Fault Tolerance Using Dynamic Reconfiguration on the POEtic Tissue

Will Barker, David M. Halliday, Yann Thoma, Eduardo Sanchez, Gianluca Tempesti, and Andy M. Tyrrell, *Senior Member, IEEE*

***Abstract*—Fault tolerance is a crucial operational aspect of biological systems and the self-repair capabilities of complex organisms far exceeds that of even the most advanced electronic devices. While many of the processes used by nature to achieve fault tolerance cannot easily be applied to silicon-based systems, in this paper we show that mechanisms loosely inspired by the operation of multicellular organisms can be transported to electronic systems to provide self-repair capabilities. Features such as dynamic routing, reconfiguration, and on-chip reprogramming can be invaluable for the realization of adaptive hardware systems and for the design of highly complex systems based on the kind of unreliable components that are likely to be introduced in the not-too-distant future. In this paper, we describe the implementation of fault tolerant features that address error detection and recovery through dynamic routing, reconfiguration, and on-chip reprogramming in a novel application specific integrated circuit. We take inspiration from three biological models: *phylogenesis, ontogenesis,* and *epigenesis* (hence the POE in POEtic). As in nature, our approach is based on a set of separate and complementary techniques that exploit the novel mechanisms provided by our device in the particular context of fault tolerance.**

***Index Terms*—Bio-inspired architectures, computer fault tolerance, evolutionary computation, evolvable hardware, reconfigurable hardware.**

I. Introduction

REDUCING the failure probability and increasing reliability has been a goal of electronic systems designers ever since the first components were developed. No matter how much care is taken designing and building an electronic system, sooner or later an individual component will fail. For systems operating in remote environments, such as space applications, the effect of a single failure could result in a multimillion pound installation being rendered useless. With safety-critical systems such as aircraft, the effects could be even more severe. Reliability techniques need to be implemented in these and many more applications. In response to this, the development of fault tolerant techniques became driven by the need for ultrahigh availability, reduced maintenance costs, and long-life applications to ensure systems can continue to function in the presence of faults.

Ensuring that computing and electronic systems provide this level of reliability has always been a challenge. As the complexity of systems increases, the inclusion of reliability measures becomes progressively more complex, but is often a necessity for VLSI circuits where a single error can potentially render an entire system useless. This fragility is in complete contrast with the great resilience of biological organisms: nature has evolved mechanisms that allow extremely complex systems to resist considerable damage and still remain operational.

As a consequence, drawing inspiration from biology in this context makes sense, but the application of bio-inspired fault-tolerance mechanisms to the design of digital hardware remains a little-studied approach. Yet, several of the issues confronting the design of reliable circuits are relatively close to the issues that an organism must confront to overcome damage. For example, the implementation of a fault tolerant mechanism in nature as in circuit design requires four stages [1].

- *Error detection*, to identify the presence of a fault in a system.
- *Error confinement*, to prevent propagation through the system.
- *Error recovery*, to remove the error from the system.
- *Fault treatment and continued system service*, to repair and return the system to operation.

In this paper, we deal in particular with two of these stages: error detection and error recovery. Nature has evolved a very complex set of mechanisms to handle these two aspects of fault-tolerance, and in this paper, we illustrate how some of these mechanisms can be used as inspiration for circuit design and, in particular, in the design of circuits on the POEtic devices introduced below.

The rest of this paper is organized as follows. Section II gives a brief outline of the POEtic project and the thinking behind the ideas within the project. The details of the POEtic device are outlined in Section III. Section IV discusses some basic ideas of fault tolerance and growth. Sections V and VI describe two methods of self-repair that make extensive use of the unique features of POEtic. A test application is outlined in Section VII and the results verifying the ideas are given in Section VIII. Finally, conclusions are drawn in Section IX.

II. The POEtic Project

Researchers studying biologically inspired systems have begun investigating both evolutionary and developmental approaches to reliable system design in the form of Evolvable Hardware [2], Embryonics [3], and Artificial Immune Systems [4]. The implementation of bio-inspired systems in silicon is

Manuscript received December 14, 2005; revised July 24, 2006 and March 1, 2007. This work was supported in part by the Future and Emerging Technologies Program (IST-FET) of the European Community under Grant IST-2000-28027 (POETIC). The Swiss participants of this project are supported in part by the Swiss Government under Grant 00.0529-1. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

W. Barker, D. M. Halliday, G. Tempesti, and A. M. Tyrrell are with the Department of Electronics, University of York, York YO10 5DD, U.K. (e-mail: dh20@ohm.yorl.ac.uk; gt512@ohm.york.ac.uk; amt@ohm.york.ac.uk).

Y. Thoma and E. Sanchez are with the Ecole Polytechnique Fédérale de Lausanne, CH-1015, Lausanne, Switzerland (e-mail: yann.thoma@hesge.ch; eduardo.sanchez@epfl.ch).

Digital Object Identifier 10.1109/TEVC.2007.896690

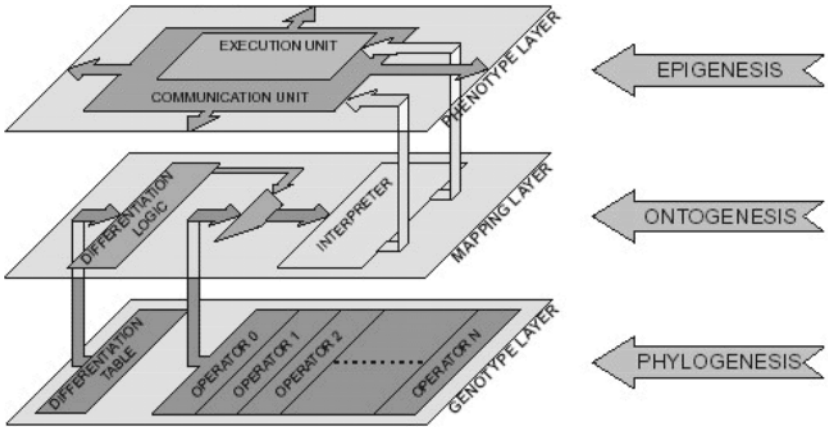

Fig. 1. The three organizational layers of the POEtic project.

quite difficult, due to the sheer number and complexity of the biological mechanisms involved. Conventional approaches exploit a very limited set of biologically plausible mechanisms to solve a given problem, but often cannot be generalized because of the lack of a *methodology* in the design of bio-inspired computing machines. This lack of methodology is due to the heterogeneity of the hardware solutions adopted for bio-inspired systems, which is itself due to the lack of *architectures* capable of implementing a wide range of bio-inspired mechanisms.

To partially overcome this problem, the goal of the "Reconfigurable POEtic Tissue" project (or "POEtic" for short) [5], funded by the European Community, was the development of a flexible computational substrate inspired by the evolutionary, developmental, and learning phases in biological systems.

To introduce the overall context of the project, biological inspiration in the design of computing machines finds its source, essentially, in three biological models [6]: *phylogenesis* (P), the history of the evolution of the species, *ontogenesis* (O), the development of an individual as orchestrated by its genetic code, and *epigenesis* (E), the development of an individual through learning processes (nervous system, immune system) influenced both by the genetic code (the innate) and by the environment (the acquired). These three models share a common basis: the *genome*.

Designed in order to simplify the hardware implementation of bio-inspired systems along the three axes, the POEtic tissue is a multicellular, self-contained, flexible, and physical substrate designed to interact with the environment, to develop and dynamically adapt its functionality through a process of evolution, growth, and learning in a dynamic and partially unpredictable environment, and to self-repair parts damaged by aging or environmental factors in order to remain viable and perform similar functionalities.

Within the POEtic project, a cell is seen as a processing unit that could be a small processor, a neuron, or a dedicated circuit executing any kind of task. The POEtic tissue is, therefore, an array of these processing units, each executing a task and communicating with its neighbors. As all cells of the tissue are identical, the genome of the entire organism implemented in the tissue (i.e., the configuration of the functional part of each *type* of cell in the organism) is stored in every cell. Mechanisms inspired by cellular differentiation in biology are then charged with identifying the specific function (i.e., the gene) to be realized by a given cell.

Following the three models of bio-inspiration, POEtic cells are designed logically as a three-layer structure [5]–[7] (Fig. 1 gives an abstract view of the cellular processors in this context).

- The phylogenetic model acts on the genetic material of a cell. Each cell is designed to contain the entire genome of the tissue and, therefore, contains an arbitrarily large and complex memory structure in its *genotype layer*.
- The ontogenetic model concerns the development of the individual. It acts on the *mapping or configuration layer* of the cell, implementing cellular differentiation (by selecting which parts of the genome to execute) and growth. Ontogenesis will also have an impact on the overall architecture of the cells where self-repair (healing) is concerned.
- The epigenetic model modifies the behavior of the organism during its operation, and is therefore best applied to the *phenotype layer*, which can be seen as the part of the processor where computation is carried out. The genes contain a description of this part of the cell.

Defining separate layers for each model has a number of advantages, as it allows the user to decide whether to implement any or all models for a given problem, and enables the structure of each layer to be adapted to the model. This adaptability, which will be detailed in the next section, is achieved by implementing the cells on a *molecular substrate*, in practice, a surface of programmable logic.

To implement any or all combinations of the three main axes of biological self-organization, the proposed tissue presents two innovative hardware-oriented aspects:

- a layered hardware structure that matches the three axes of biological organization;
- an input/output interface with the external world that allows each cell to perceive and modify its environment when and where necessary.

Moreover, the final hardware design (the VLSI device) has a set of specific novel features built into its fabric to assist with bio-inspired designs. The next section will give an overview of some of these features, with particular emphasis on how they might be used in the context of this paper, that is, for the design of fault-tolerant systems.

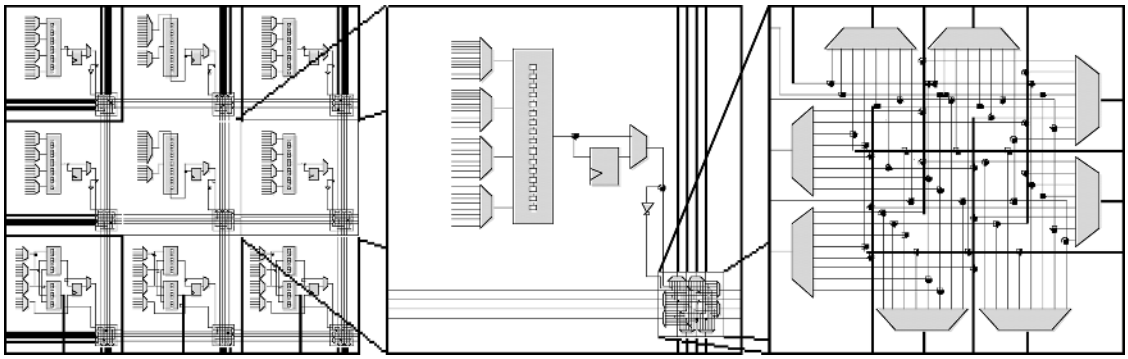

Fig. 2. The molecules.

III. POEtic Chip: Hardware Features

Following the three-layer architecture presented above, the POEtic chip had to allow the implementation of artificial organisms capable of growth, self-repair, and learning, as well as the capacity of letting a population of such organisms evolve. The chip can be considered as divided in two parts: a microprocessor and a reconfigurable array of simple elements. Among its main novel features, of particular note for the field of evolutionary and cellular computation are the following [8].

- The presence of a dedicated microprocessor that allows a rapid reconfiguration of the array.
- The facility to create, dynamically, data paths between resources on one chip, or across resources on multiple chips.
- The ability of an array to *self-reconfigure* parts of itself, a powerful feature for evolvable hardware and self-repairable systems.
- The use of specific circuit design techniques on the chip (for example, the use of multiplexers instead of tri-state buffers to implement buses) to ensure that no possible configuration bitstream could cause a short-circuit on the chip (while not used in the work reported in this paper, this feature is very useful, for example, in unconstrained evolution).

As in all programmable logic, the implementation of a bio-inspired system on a POEtic chip implies that the organisms (i.e., the arrays of cellular processors) be loaded into the hardware reprogrammable part, which in POEtic is called the "organic subsystem." This subsystem is divided logically into two separate planes: the *functional plane*, hosting the reconfigurable logic, and the *routing plane*, which creates data paths for inter-cellular communication at runtime. The functional plane is composed of basic elements that play the role of *molecules*, and can implement any digital circuit, while the routing plane implements a distributed dynamic routing algorithm. The main specificities of the organic subsystem are its capabilities of changing *at runtime* the function realized by the molecules and the connections between molecules. In the next subsections, we will describe the two planes, defining first the architecture and the modes of operation of the molecules, and then the hardware and algorithm that control the dynamic paths creation (more details in [8] and [9]).

A. Molecules

A molecule (Fig. 2) basically contains a 4-input lookup table (LUT) and a flip-flop, the output being combinational or sequential [8]. This architecture is quite similar to commercial field programmable gate arrays (FPGAs) supplied by Xilinx or Altera, but special features have been added to support bio-inspired applications. A molecule can be configured in eight different operational modes (note that some modes relate to the routing mechanism described in the next section).

- *4-LUT*: The functionality corresponds to a 4-input LUT.
- *3-LUT*: The LUT is split into two 3-input LUTs.
- *Shift memory*: The LUT is used as a 16-bit shift register.
- *Comm*: The LUT is split into an 8-bit shift register and a 3-input LUT.
- *Input*: In this mode, a molecule acts as an input element, receiving data from the routing plane. A unique identifier specifying the source of this data (see output mode) is stored in the 16-bit shift register (see shift memory mode).
- *Output*: In this mode, a molecule acts as an output element, sending data to the routing plane. The unique identifier of this molecule is stored in the 16-bit shift register (see shift memory mode).
- *Trigger*: This mode is used by the routing algorithm to synchronize the process of decoding the identifiers (see input and output modes). In addition, two inputs are responsible for resetting the routing plane and disabling molecules.
- *Configure*: A molecule in this mode can modify the configuration bits of other molecules, allowing self-reconfiguration.

As in all programmable logic, connections are a fundamental part of the circuit. Each molecule contains a switchbox composed of eight 8-input multiplexers, two for each cardinal direction (right in Fig. 2). Intermolecular communication (used for the conventional interconnection between the programmable logic elements) goes through these switchboxes. The configuration bits of the switchbox are loaded and specified at configuration time, but can be changed by a full or a partial reconfiguration of the circuit (configure mode, see next section on partial reconfiguration). In addition, the dynamic communication network that exploits the routing plane, described next, is used for intercellular communication between processors.

Partial Reconfiguration: One of the main innovations in the molecular substrate is the ability to perform the partial reconfiguration of molecules. Any molecule has the possibility to change the configuration of any other molecule(s). This feature also allows the user to decide specifically which parts of the configuration should be modifiable during runtime.

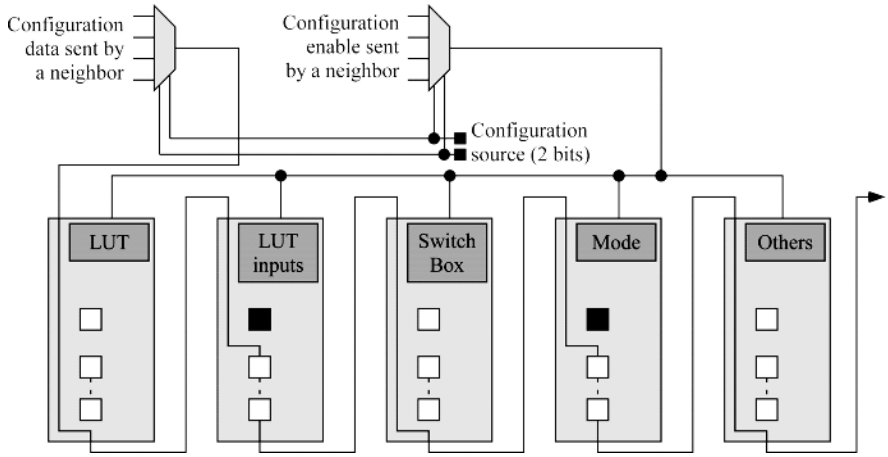

Fig. 3. Configuration bits of a molecule split into five blocks, two of them being reconfigured.

The 76 bits that represent the configuration of a molecule are split into five blocks: *LUT*, *LUT input multiplexers*, *switchbox*, *operational mode*, and *other bits*. A special bit associated to each block, indicates if a particular block in a particular molecule can be reconfigured during a partial reconfiguration process (if not, it will be bypassed during any such process). In this way, only part of a molecule can be designated as modifiable: for instance, it is possible to change only the contents of the LUT, and hence the function realized by the molecule, while keeping all of its connections to the rest of the array unchanged. Fig. 3 shows these five blocks, with their corresponding special bit, and the chain of partial reconfiguration passing through the LUT inputs and the mode. The molecule needs two configuration bits to indicate the origin of an accepted partial reconfiguration, and a bit to enable its propagation to its neighbors. These 3 bits, with five additional special bits, cannot be modified by the molecules themselves. These bits can only be modified by the microprocessor through a direct reconfiguration. It is also interesting to note that a molecule is able to transmit the configuration bits through the connection network, allowing for long-distance reconfiguration (i.e., not only of the directly connected neighbors).

The functional behavior of molecules can be modified by partial reconfiguration. This feature is necessary for self-repair, as a spare cell would be required to execute the task of a dead one. Through partial reconfiguration, it is possible to create the functional part of a new cell by copying the appropriate part of an existing cell. Furthermore, the first three blocks can be used to store information, up to 54 bits per molecule. This kind of memory, accessed serially, can be very useful to efficiently store a genome.

Molecular Enable: Another useful feature of the molecules is their ability to perform a *global molecular enable*. Trigger molecules control this process and every molecule has a configuration bit that indicates if it is sensitive or not to this process. For example, to perform self-repair, a cell might be divided in two parts: a functional part and a part responsible for the self-repair. The functional part can then be sensitive to the global enable, and be disabled while a self-repair process occurs, ensuring correct behavior of the whole system.

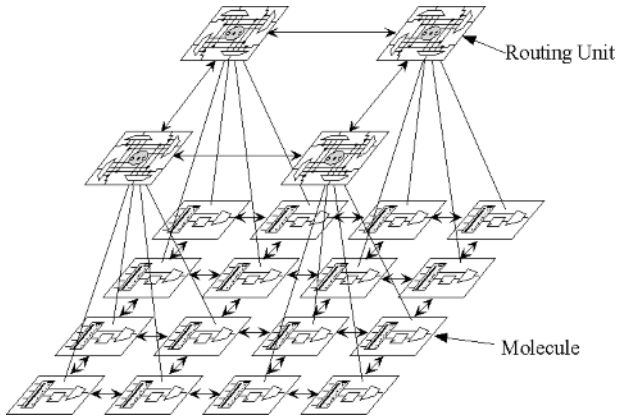

Fig. 4. Molecules and routing units.

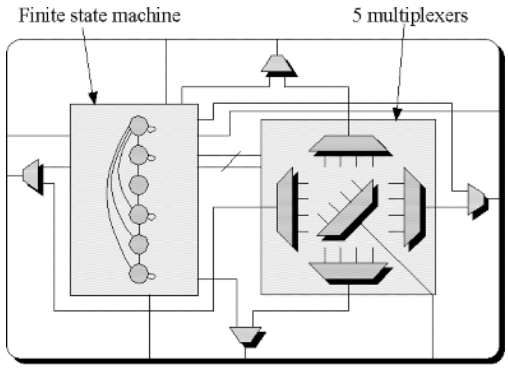

Fig. 5. Routing unit.

B. Routing Plane

The routing plane, logically superposed on the molecular plane (Fig. 4), enables dynamic configuration of paths that connect molecules across one or multiple chips. In a sense, it is the POEtic equivalent of long-distance connection buses in a conventional FPGA, with some crucial differences.

The routing plane consists of a set of regularly spaced *routing units* that use multiplexers to direct the flow of data. One routing unit connects to four molecules in the subjacent plane and is itself connected to its four cardinal neighboring

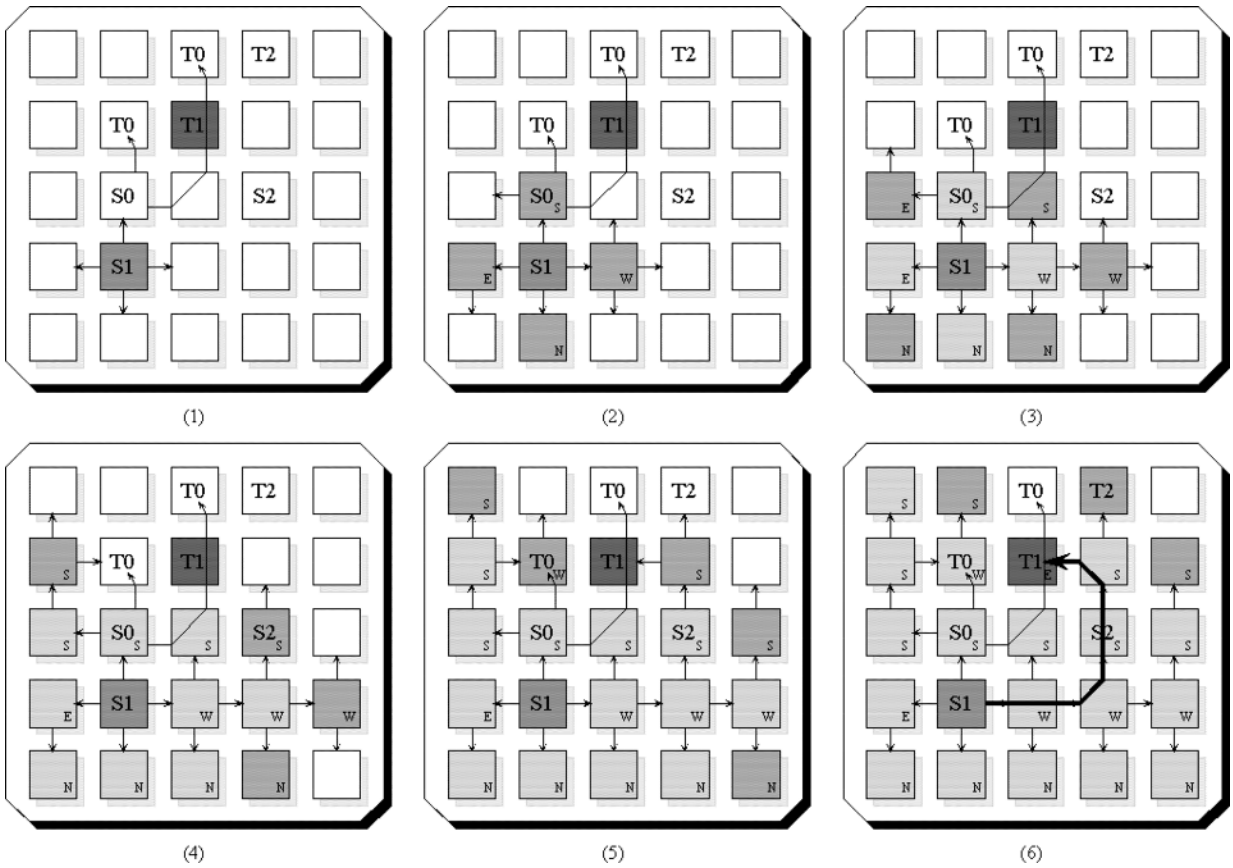

Fig. 6. Creation of a path during a routing process [7].

routing units (Fig. 4). This choice of one routing unit for each four molecules represents a compromise between area and efficiency. The routing mechanisms ensure that only one of the four subjacent molecules has access to the routing unit.

The routing units implement a distributed routing algorithm, and configure multiplexers that are then used to transmit data from a molecule configured as an output to a molecule configured as an input. The novelty of this mechanism is the absence of configuration bits: the communication paths are established dynamically at runtime, hence, the user does not need to define the connection network at design time. A detailed description of the algorithm used in the routing plane can be found in [9].

A routing unit (Fig. 5) is composed of a controller (a finite-state machine) and five 4-input multiplexers selecting the value to send in each cardinal direction and to the molecules connected directly to it. Each multiplexer is controlled by two bits that are modified by the finite-state machine during the routing process, and an additional bit indicating if the multiplexer is currently in use, in order to avoid collisions [9].

The algorithm is based on 16, 8, 4, 2, or 1 bits long IDs stored in the shift register of molecules configured in input or output mode: during the routing process, molecules connect with other molecules having the same ID. These molecules manage the creation of paths by controlling a *routing enable signal* sent to the routing units. When a molecule enables the signal, a routing process is launched and continues until the molecule is connected to the other molecules with the same ID; a disabled signal indicates that a molecule is ready to accept a connection. This approach is very well adapted for the implementation of cellular self-repair or growth mechanisms, since it allows *spare cells* to be placed on the array simply by giving them all the same ID: a cell that wants to connect to a spare cell will then simply create a path to the nearest cell with that ID, without needing to know its actual physical position [9].

A crucial feature of the algorithm is that it is distributed, and that every routing unit can try to start a routing process at any time, including many at the same time. A hardware-based algorithm grants priority to the molecule in the most southwesterly position. The routing process is serialized by allowing a single path at a time to be established. This avoids the need for explicit synchronization mechanisms.

Additionally, molecules can force a reset of the routing plane after which every connection needed by the molecules will be recreated. In this way, a self-repair mechanism can reinitiate new connections when a cell is replaced by a new one.

In a routing process, every routing unit (RU) that needs a connection propagates a signal through the chip. Priority is given to the most southwest unit, which becomes the *master*. The master unit then sends its ID to all other RUs serially. At the end of this phase, every RU knows if it is involved in the current routing process. The master then disables all molecules in the same mode (input or output), thus avoiding contentions. The next phase is a parallel implementation of Lee's algorithm [10], and is illustrated in Fig. 6, where we can see the creation of the shortest path between the source S1 and the target T1, taking into account existing paths.

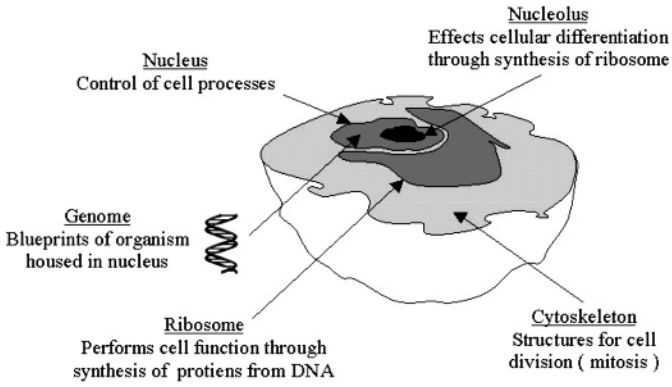

Fig. 7. Eukaryote cell structure.

IV. Growth and Fault-Tolerance

Within this project we have attempted to take considerable inspiration from biological systems [7]. One consequence of this is that we have used (and in some cases abused!) biological terminology for our electronic systems described in the following sections.

To define a general operational framework, we call *molecules* the smallest processing elements on the POEtic device. A number of molecules grouped together form what we term a *cell*. One or more cells grouped together will form an *organism* (performing the total function of one's application).

In biological systems, all multicellular organisms are composed of *eukaryote cells* (Fig. 7). These cells contain a nucleus and can differentiate to perform different functions. The genome contains bundles of information, the *chromosomes*, that define the function of each cell type within an organism. When differentiating, the *nucleolus* synthesizes the *ribosome* from a selected chromosome. The ribosome realizes the cell function by transcribing proteins from the DNA in the genome. The selected chromosome and, therefore, the structure of the ribosome produced are dependent upon the position of the cell within the organism and determine its type and behavior through its protein production.

These division and differentiation mechanisms have a fundamental importance in the growth of an organism from a single initial cell to adulthood (*ontogenesis*). These same mechanisms, moreover, are also the basis of many of the processes used by organisms to repair or replace faulty cells and are, thus, crucial for the fault tolerance of a biological organism, as described below.

The POEtic project provides a unique platform for investigating mechanisms at work in biological systems that exhibit fault-tolerant behavior [11], as we wish to demonstrate through the development of a cellular ontogenetic fault-tolerant mechanism based upon growth on the POEtic tissue.

Self-repair or healing is a critical mechanism in a living organism's response to damage, involving the growth of new resources, in the form of cells, and their integration into the organism. An electronic system cannot grow new silicon resources in response to device faults in the same way and so growth in silicon is generally emulated by having redundant resources into which the system can grow. The POEtic architecture provides novel features which are particularly useful for implementing models of growth in digital hardware, including the underlying molecular architecture, the dynamic routing plane, and the self-configuration mechanism.

A. Growth

Growth in multicellular organisms is based upon two distinct mechanisms: cellular division and cellular differentiation. Cellular division is a process of self-replication through which cells produce copies of themselves. Cellular differentiation is the process through which cells organize themselves by taking on specific functional types depending on their position within an organism.

During the healing process, some types of differentiated cells in damaged areas within an organism can divide in order to replace those lost. This is referred to as *renewal by simple duplication*. Others rely upon the *differentiation of stem cells* located within the damaged tissue [12].

Prompted by these distinct modes of growth, two cell designs have been implemented on the POEtic tissue and are described in this paper: an embryonic array emulating the processes of growth through cellular differentiation and an emulation of growth through simple duplication of differentiated cells.

B. Fault Detection

Biological mechanisms for fault detection in themselves provide a rich field of research to which the POEtic platform is applicable [12]. However, as the aim of this work is to investigate fault tolerance and by extension, growth mechanisms, a standard technique of hardware redundancy has been chosen to provide fault detection in the designs [13]. Thus, if there is a difference in output between the redundant unit(s) and the main unit, the system has detected a fault. The advantages of this method for fault detection are that it is simple, acts at the resolution of a single clock cycle, operates online, and is applicable to any cell function.

C. Fault Models

A number of different models exist for faults that can occur in programmable VLSI systems such as the POEtic device [13]. An important distinction between fault models in this work is between soft faults and hard faults. *Soft faults* are disruptions to the data contained within a programmable VLSI device. They do not affect its fabric and can be rectified by resetting or reconfiguring the device. *Hard faults* are produced by physical damage to the device and render that part of the device permanently unusable.

The redundancy-based detection system referred to above cannot distinguish between soft and hard faults and so in the following cell designs all faults are assumed to be hard. This implies that the faulty part of the device must be deactivated rather than reconfigured and implies that all faults instigate "death" (or, in biological terms, *apoptosis*). Additionally, it has been shown that soft faults (and/or transient faults) can be addressed using other bio-inspired techniques, such as development [14].

V. Embryonic Healing on the POEtic Tissue

An embryonic array consists of an array of cells implemented in reconfigurable logic. Each cell contains a set of configuration strings describing a set of functions for cells in the or-

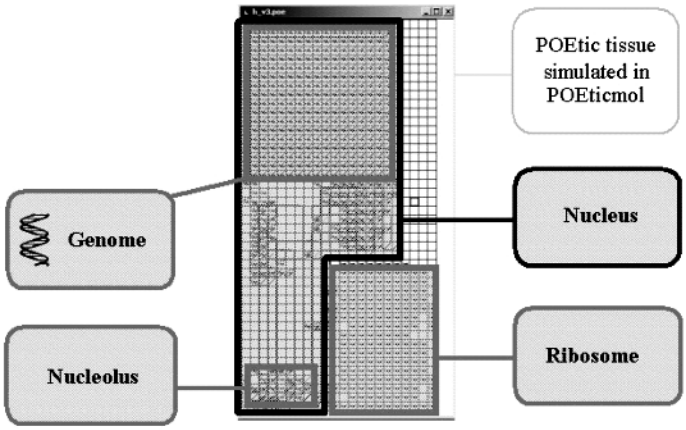

Fig. 8. Artificial embryonic cell structure on the POEtic tissue.

ganism. This set of configuration strings is analogous to the biological genome. Development is achieved through a differentiation process during which each cell identifies its configuration string with respect to its location within the array and uses it to configure its function [3], [15]. Fig. 8 shows an artificial embryonic cell on the POEtic tissue simulated using the POEtic design tool *POEticmol* [7], [16]. In the embryonic implementation presented in this paper, embryonic cell structures are preconfigured in an array on the tissue and an organism is developed and repaired by means of cell differentiation and apoptosis (cell death).

A. The Ribosome

The ribosome is the area of the cell where the functional part of the cell is implemented. Duplication of the cell's function enables fault detection using redundancy. The duplicate areas are initially blank and require configuration from a stored gene (Fig. 9, [11]). They consist of molecules which have the partial configuration inputs from their neighbors chained together, as shown in Fig. 9, and all configuration registers set for configuration. This allows an arbitrary, but defined, cell function to be configured within them. The stored gene, therefore, consists of the contents of the configuration registers for each molecule in the function listed, in the order in which they appear in the chain from the head to the tail.

B. Genome Storage

The stored genome consists of individual *gene blocks* (Fig. 10): each block can be selected by the differentiation system as the source for the ribosome within the cell [11]. The genes consist of shift memory molecules which store the configuration string in their lookup tables. The inputs and outputs of the memory molecules are chained together in the same way as the molecules in the ribosome. During configuration every gene in the stored genome shifts its contents out from its head, with the string from the gene selected by the differentiation process being channeled into the ribosome (Fig. 11, [11]). The head of each gene block is looped back into the tail by connecting the memory molecule output of the head molecule to the input of the tail molecule to preserve the contents of the gene block (Fig. 10).

C. Fault Detection in the Stored Genome: Cyclic Redundancy Code

The use of a stored genome has implications with respect to the fault-detection scheme proposed for the cell design. The stored genome requires a significant area of resources on the tissue which can potentially sustain faults. Approximately four times more molecules are required to store each gene than are used in the function block that it describes (it is possible to compress the genome and reduce this figure to 1.5 times, however, in this work, this was not used for ease of development of this application) and an embryonic cell will require as many genes as there are different cells in the system (in many applications the number of different cells may actually be small, but this is still a significant resource). As both ribosome copies are configured from the same stored gene, a fault in the gene will go undetected by the fault-detection system as both copies will be producing the same erroneous outputs.

A second fault-detection system has, therefore, been implemented in the embryonic cell design in the form of a cyclic redundancy code (CRC) which can detect faults in the gene being used to configure the cell function by means of a frame check sequence (FCS) tagged onto the end of every gene [17].

On configuration of the cell ribosome, the configuration string for the selected gene (including the FCS) is passed through the CRC register as the cell function is configured. If the stored gene is correct, then the output of all of the CRC register elements will be zero at the end of this process. Otherwise, at least one of the outputs of the register elements will be high indicating a fault in the gene (Fig. 12, [7]) and triggering cell apoptosis and system differentiation.

D. Cell Nucleus

The cell nucleus (in this design) is responsible for controlling the five main processes that make up the life-cycle of the embryonic cell [7]. These are cellular differentiation, ribosome configuration, fault detection, apoptosis, and routing.

Cellular Differentiation: Each of the embryonic cells in the array has a differentiation input and output molecule. These inputs and outputs are linked in a chain across the tissue via the dynamic routing layer. On system initialization, or in response to a fault, a differentiation signal is sent down the chain from an external source. Cells select their allocated genes depending on the signal received at their differentiation inputs.

This signal consists of a series of ones equal in length to the number of cells in the organism that are to be developed. For example, in Fig. 13, the first cell in the chain receives 3 ones before receiving a zero. At this point, the output of cell one is asynchronously reset to zero causing a propagation through which all cell differentiation outputs down the chain asynchronously reset to zero. This terminates the differentiation process in all cells, each of which will have received one less one at its differentiation input than the previous cell in the chain.

The cells then select their allocated genes depending upon the number of ones received at their differentiation inputs. Cells at the end of the chain that are not used in the organism blank their ribosomes and are left as spare cells that can be integrated into the organism in the event of cell failure higher up the chain. The

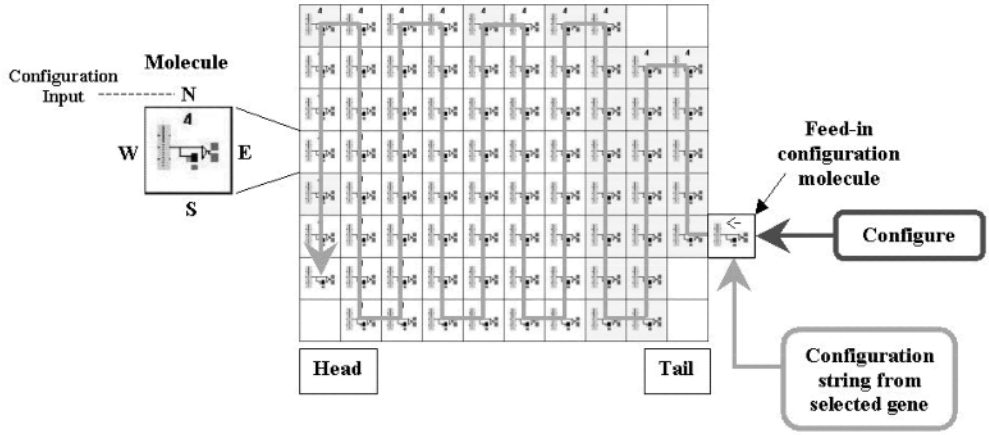

Fig. 9. Ribosome configuration chain.

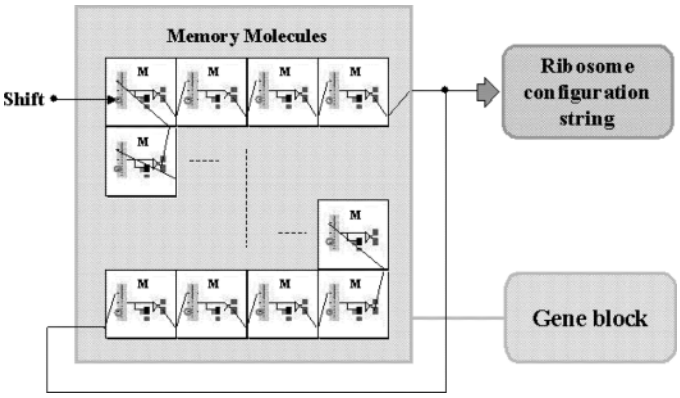

Fig. 10. Gene block.

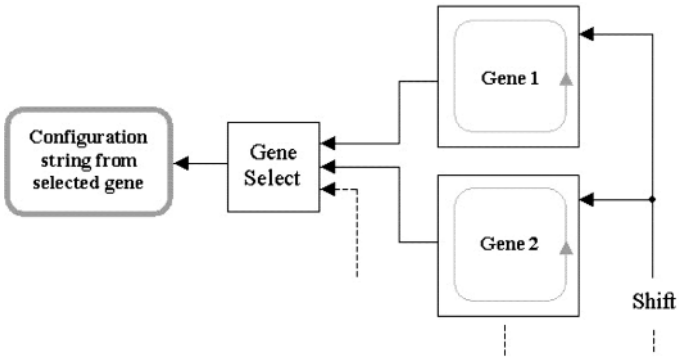

Fig. 11. Stored genome consisting of selectable gene blocks.

differentiation process can be instigated at any point by driving the differentiation signal into the chain.

Ribosome Configuration: Completion of the differentiation process triggers a *molecule configuration counter*. The counter enables the shift input to each of the gene blocks in the stored genome and enables a configuration molecule which feeds the output of the selected gene into the configuration input at the tail of the ribosome configuration chain, starting the construction of the functional areas.

When the counter indicates that the number of molecules in a ribosome have been configured, the enable to the configuration molecule is disabled and the counter resets. At this point, the function areas of the cell have been programmed with the

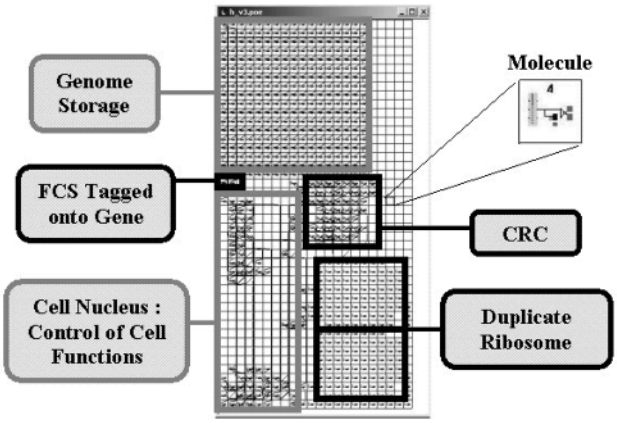

Fig. 12. Embryonic cell fault-detection mechanisms on the POEtic tissue.

selected gene and are ready for integration into the system. Before this can occur, however, the FCS must be shifted through the CRC register to check that the gene is correct. This is controlled by a second counter triggered by the overflow of the first and shifts the gene blocks by a further 32 bits driving the FCS out of the gene block through the CRC register.

Fault Detection: In the cell nucleus, the values at the outputs of the two ribosome copies are compared and a fault flagged in response to a discrepancy. The integrity of the configuration of the ribosome copies is tested by the CRC register on configuration and, if it is found to be corrupt, a fault is flagged. The cell nucleus combines these two fault flags into a single signal that triggers cell apoptosis and system differentiation.

Differentiation in response to a fault is triggered by the faulty cell activating the global molecular enable through its trigger molecule. This is detected by the external system controlling the differentiation signal input which drives the signal into the differentiation chain in response.

Apoptosis: Apoptosis in a faulty embryonic cell is achieved by selecting the blank gene, simply a source of zeros, and bypassing the delay which the cell would otherwise introduce into the differentiation chain. This shifts the differentiation values received downstream of the faulty cell one cell down the chain and causes the cell to blank its function areas removing any molecules such as input and output molecules that may interfere with the operation of the system.

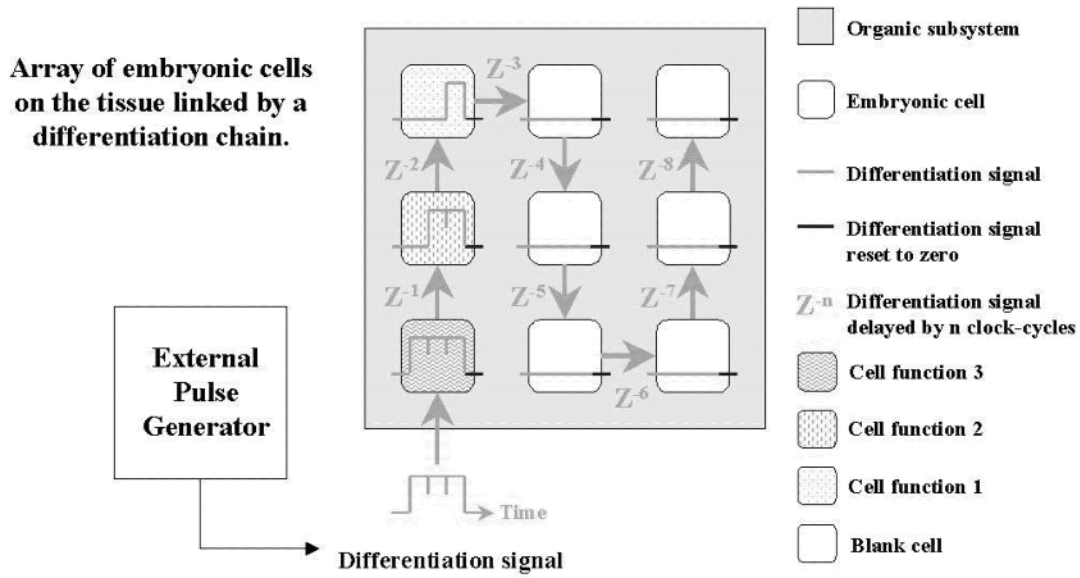

Fig. 13. Differentiation chain.

Routing: Having completed the processes of cellular differentiation and configuration, the final step in producing the functioning embryonic system is to route together any input and output molecules that have been configured in the cell ribosome.

The first healthy cell in the differentiation chain is assigned the task of triggering the routing process. Every cell contains a trigger molecule capable of this. On completing the configuration of their functional areas every cell sends a pulse on the routing enable signal to its inputs entering them into the routing process. This pulse is also sent to an input of the cell's trigger molecule via a gate to reset the routing. The output of this gate is enabled if it is the first working cell in the chain thereby triggering the routing process. This system is required as firing multiple trigger molecules on the tissue will result in multiple routing processes being instigated wasting clock cycles.

VI. Healing by Simple Duplication of Cells on the POEtic tissue

Simple duplication is the process by which differentiated cells of a specific type produce copies of themselves through division. The mother cell is the source of the genome, which is copied into the daughter cell, and the gene governing the function of the daughter cell is preselected as being the same as that of the mother.

This kind of process can be realized in the POEtic tissue by a cell connecting to a spare cell and copying the configuration string of its ribosome into the ribosome of the spare cell. Since the function of the new cell is predetermined, the genes that form the rest of the genome do not need to be stored in the cell. As the ribosome of a functioning cell contains exactly the same configuration string as the stored gene associated with that function, no entries are required in the stored genome and it can be omitted. Also, as the cell does not need to select its function through differentiation, the nucleolus can be omitted (Fig. 14).

While this approach represents a departure from direct biological inspiration, since cell division in nature preserves the complete genome in each cell, the departure is less abrupt than it might seem: unlike stem cells, differentiated cells in nature, while storing a complete genome, are not capable of generating cells that are different from themselves.

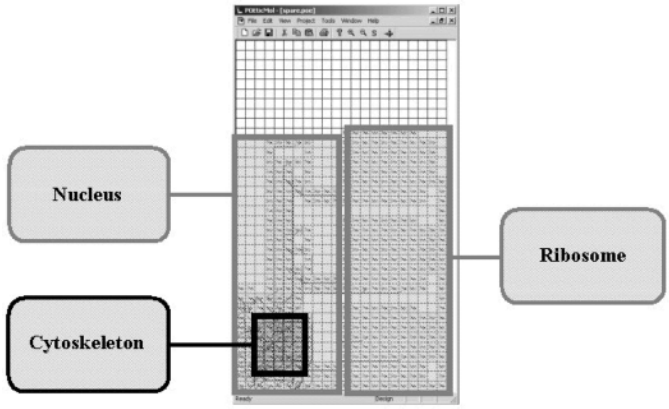

Fig. 14. Structure of a cell created through artificial simple duplication on the POEtic tissue.

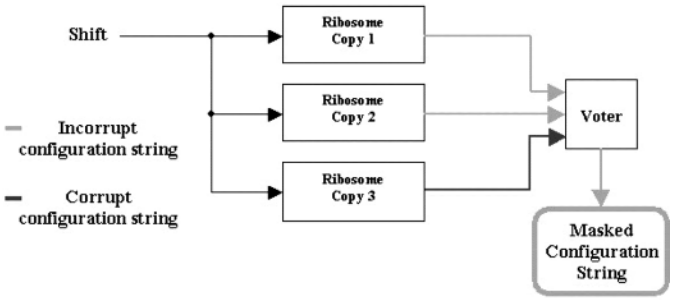

Fig. 15. Triple modular redundancy masking of a gene fault.

Growth in the liver in response to damage, for example, occurs through the simple duplication of healthy cells promoted by an imbalance in the levels of inhibitory and excitatory growth hormones caused by cell loss. In order to avoid having to simulate hormone regulatory networks within cell populations (which forms a substantial area of research in its own right, e.g., [12]), a departure from the analogy with biology has

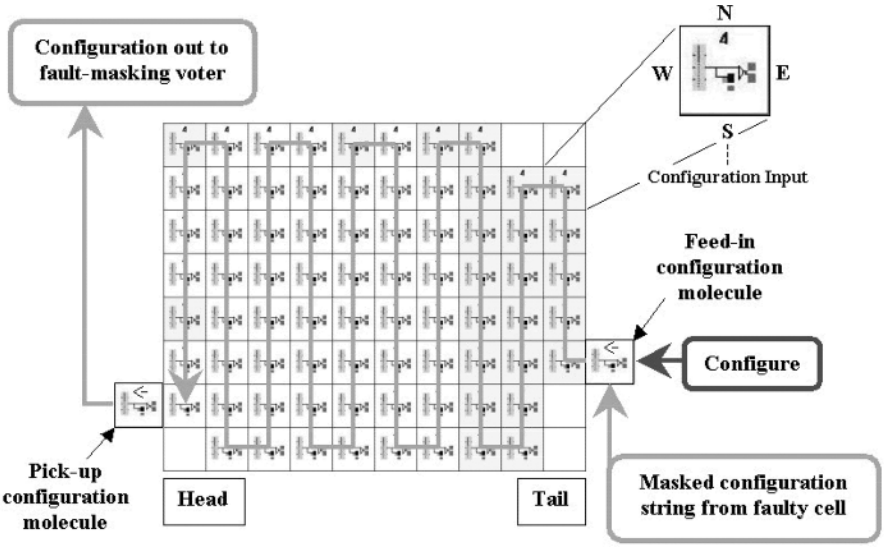

Fig. 16. Ribosome configuration chain.

been made in the POEtic implementation of growth through simple duplication. This departure consists of having the malfunctioning cell, rather than a healthy cell from a population, duplicate itself in order to generate a healthy replacement. If the fault has affected the contents of the configuration string within the cell ribosome (which holds the cell data in the form of the molecule LUT contents and flip–flop outputs, as well as the contents of the molecule configuration registers), then this will result in the growth of another faulty cell. The malfunctioning cell must, therefore, be able not only to identify that a fault has occurred in its function but be able to correct its configuration string when creating a duplicate copy of itself. This is a process known in engineering as *fault-masking* and can be achieved based upon the same assumptions made for the hardware redundancy fault-detection system, by adding a third ribosome copy to the cell in a system referred to as triple modular redundancy (TMR) (Fig. 15).

Under the assumption of a fault condition in a single ribosome, the majority consensus on the configuration string output gives the masked output for configuring a healthy new cell.

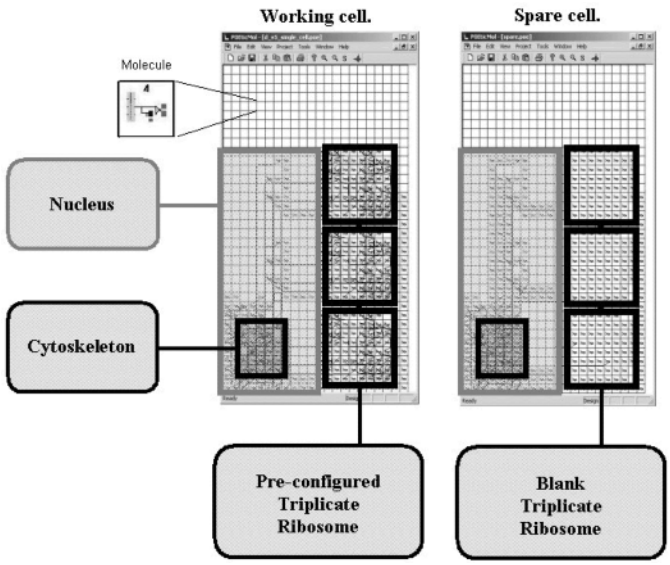

Fig. 17. Simple duplication cell: initialized working and spare cells.

A. The Ribosome

Triplication of the ribosome enables fault detection and masking by the hardware redundancy method described previously. The ribosome areas in the simple duplication cell are identical to those in the embryonic cell except that a configuration molecule is placed at the head of the configuration chain to pick up the configuration string as it is shifted out of the ribosome (Fig. 16). Cell ribosomes are either preconfigured with a cell function or left blank on system initialization. Blank cells can be used as universal spare cells accepting the function and data from any working cell in the system (Fig. 17).

B. Cell Nucleus

The cell nucleus (in this design) is responsible for controlling the four main processes that make up the life-cycle of the simple duplication cell: fault detection, routing, configuration of a spare cell, and apoptosis. Note that while the cell nucleus has the overall controlling function in both implementations (Sections V and VI), the details of the functions required will be different since each implementation is using different forms of bio-inspiration. For example, the cytoskeleton in this section performs similar, but different, functions to that of the nucleolus in Section V.

Fault-Detection: Fault-detection within the simple duplication cell simply consists of a comparison between the outputs of the three copies of the ribosome. If a discrepancy occurs a fault is flagged and the processes of cell replacement configuration are triggered.

Routing: Two routing tasks are performed by the cell nucleus: routing to a spare cell for configuration in response to a fault and routing inputs and outputs in the function areas of a newly configured cell into the system. Each cell has a dedicated configuration input and output molecule and a trigger molecule located within its nucleus. The input and output molecules all

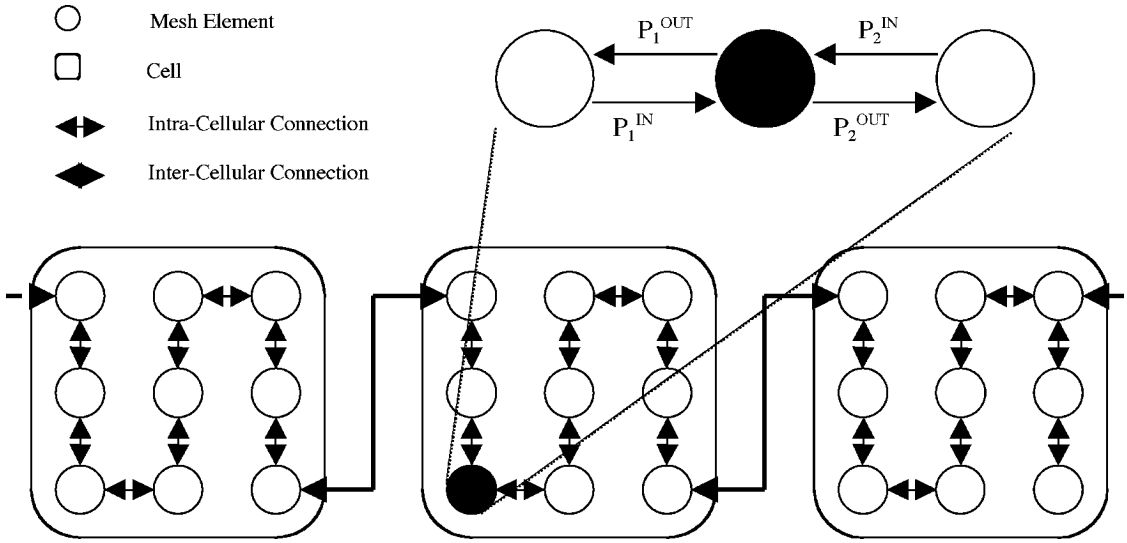

Fig. 18. Nine mesh-element one-dimensional waveguide cell with left and right input streams.

have the same identification address which is reserved as a configuration connection.

When a fault is flagged, the cell activates the *routing enable input* to its configuration output and sends a pulse to its trigger molecule initiating a routing process. This process connects the configuration output of the faulty cell to the nearest active configuration input. All cells that are preconfigured with a function on system initialization have their configuration inputs disabled to prevent faulty cells from routing to working cells and also from routing to themselves when searching for a spare cell to configure. All other cells on the tissue are initialized as spare cells with active configuration inputs.

Routing of the inputs and outputs in the function configuration areas of a newly configured cell is achieved by the new cell sending a pulse on the routing enable signal to its inputs entering them into the routing process. This pulse is also sent to the reset routing input on the cell's trigger molecule initiating the routing process.

Configuration of a Spare Cell: Once a faulty cell has established a connection between its configuration output and the configuration input of the nearest spare cell it "pings" the spare cell by sending a signal down the connection and begins configuring it. The configuration for the spare cell is sourced from the fault masking voter in the faulty cell (Fig. 15). This voter accepts the three configuration output streams which are being flushed out of the triplicate ribosome as its inputs and outputs the consensus for each bit shifted through it. The ribosome copies in the faulty cell are flushed out by a zero source effectively deleting the ribosome and removing any molecules from the faulty cell which may interfere with the system when the cell has died (for example, I/O molecules). When the ribosome copies of the spare cell are fully configured and ready for integration into the system, the newly configured cell destroys its configuration input to prevent connection from other faulty cells or itself and triggers the routing of its ribosome into the system.

Apoptosis: Apoptosis of the faulty cell is achieved through the blanking of its ribosome during the configuration of a spare cell and by setting the fault detection and routing trigger systems to an inert state. As the cell's configuration input has already been destroyed this means that the cell has no means of connecting to other cells and no configured cell function. It has, therefore, ceased to exist from the point of view of the cellular system.

VII. Test Application

In order to investigate issues regarding the implementation of the fault-tolerant mechanisms described on the POEtic tissue, a nontrivial test application has been defined and implemented on the substrate. This application is based upon an audio system presented in [18] that simulates the human vocal track and requires real-time processing.

Our system (Fig. 18) consists of an array of nine one-dimensional waveguide mesh elements [11], [18]. For those not familiar with waveguide meshes, each mesh element can be considered as a particular type of processing element, performing a simple computation on 16-bit two's complement integers. Each element consists of many molecules and exchanges information (the Ps in the figure) with its neighbors. Arrays can be chained together to form a one-dimensional waveguide with length an arbitrary multiple of nine, as shown in Fig. 18.

Two kinds of processing elements are present in the system.

- *Scattering elements* have up to four inputs and one or more outputs (dependent upon the dimension of the track), and at each time-step of length Δt perform the computation described in (1)

$$p_i(t) = \frac{1}{2} \sum_{n \in \mathcal{N}} p_n(t - \Delta t) p_i(t - 2\Delta t). \tag{1}$$

- *Reflective elements* have one input, one output, and a coefficient of reflection λ. At each time step they perform the computation described in (2)

$$p_i(t) = (1 + \lambda) p_j(t - \Delta t) - \lambda p_i(t - 2\Delta t). \tag{2}$$

It was decided to use cellular routing for intercellular connections (which can, thus, bridge the boundaries between POEtic

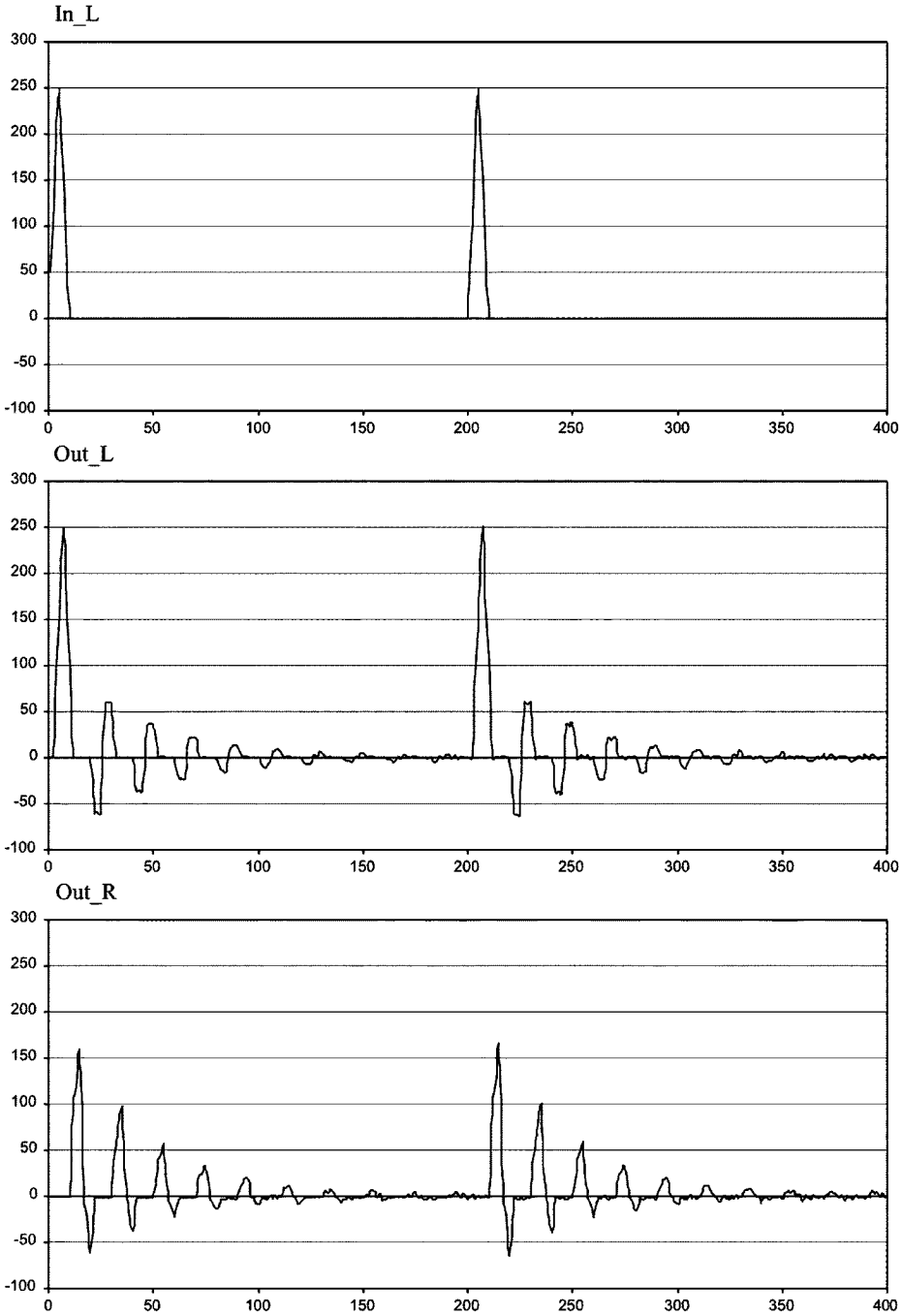

Fig. 19. Cell output in response to pulses applied to one input.

devices) and molecular routing for local, intracellular connections. This also means that the mesh elements can be considered as "cells" in a cellular application. In this way, a simulated vocal tract can be configured by creating cells of the appropriate type in a position on the tissue that maps directly to the two-dimensional structure they represent [18]. The individual "cell types" were designed in the *POEticmol* design tool.

Fig. 19 illustrates the behavior of a cell with a pulse applied to one of its inputs (In_L) and what response is expected at its left (Out_L) and right (Out_R) outputs [7]. Note that in this figure the timescale shows two input pulses (at $\mathrm{t} = 0$ and $\mathrm{t} = 200$), while in the following experiments more input pulses were applied and, hence, the timescales have been shortened to allow multiple input pulses and the resulting output waveforms to be shown.

VIII. Simulations and Results

The cell designs described have been simulated in the presence of randomly generated faults using the POEtic design tool *POEticmol*. Since *POEticmol* simulates the behavior of the POEtic device using the same VHDL description from which the VLSI device was fabricated, accurate simulations of the effects of faults on system behavior can be made.

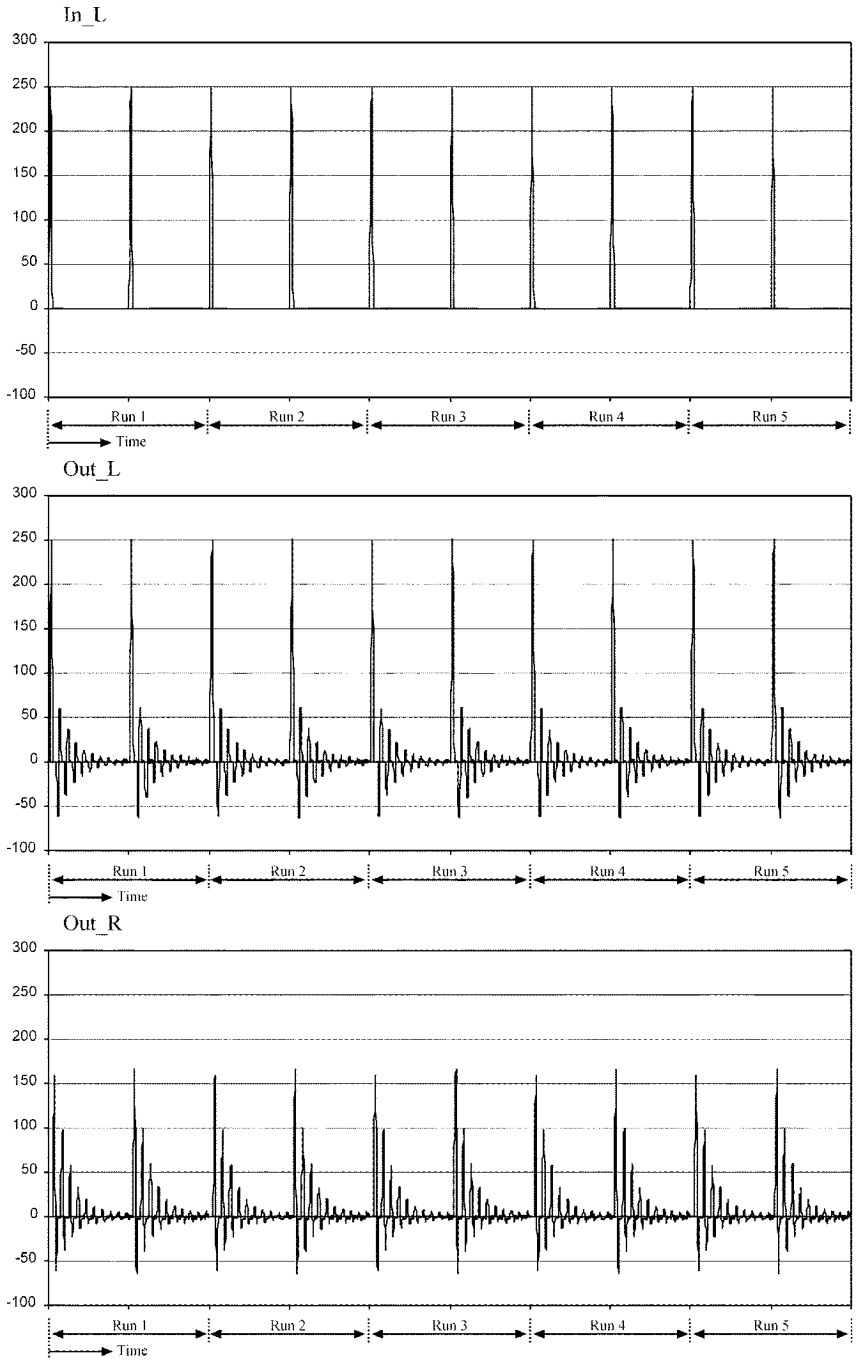

Fig. 20. Embryonic organism: no faults injected.

After the elaboration of the VHDL description of the POEtic tissue, *POEticmol* scans through the entire tissue description compiling a list of signal elements. This scan gives a full list of all possible locations where a fault can be injected into the application. A number of signal elements are then randomly chosen from this list and will be forced into a fault condition during the execution of the application. Each of these signal elements is allocated a "time stamp" which indicates when the fault will be injected (this stamp is created by specifying a random number of clock cycles from the start of the simulation).

The fault model used is the *"stuck-at" fault*, or *single hard error* (SHE) [13]. The number of faults forced in the simulation is set high enough to guarantee a satisfactory yield of terminal cell faults (i.e., to ensure that a significant and observable effect on the output of the application will occur). Some example results showing the behavior of the embryonic cell design in response to randomly generated faults can be seen in Figs. 20–22 [7]. Similarly, Figs. 23–25 show analogous results for the simple duplication cell design (note that the faults are randomly generated and that as a consequence the simple duplication and embryonic cell simulations do not use the same set of faults).

Each figure shows the input and output data for two cells, a working cell and a spare cell (note that both kinds of cells are subject to faults), simulated over five runs of a fixed number of

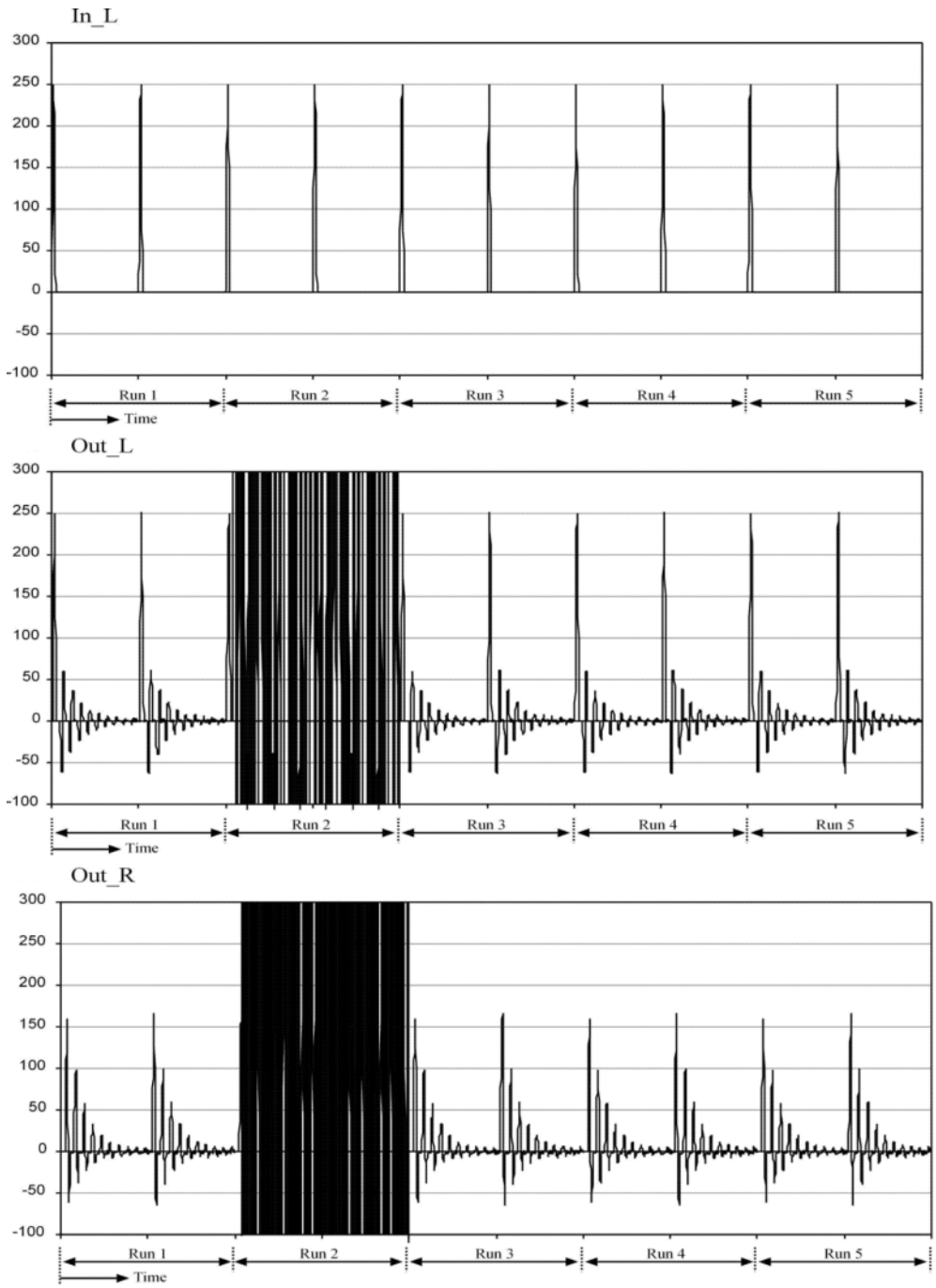

Fig. 21. Embryonic organism: faults injected. Fault-tolerance off.

output words with our test application (described in Section VII) in the ribosome. Each run has a different set of randomly generated faults which are injected into the tissue. The design is reset after each run and all faults are removed in order to delete all traces of previous executions).

Each cell design is simulated under three conditions. In the first set of simulations (Figs. 20 and 23), no faults are forced into the tissue, illustrating the target output which the fault-tolerant system is aiming to achieve. In the second set (Figs. 21 and 24), faults are forced into the tissue but the fault-detection and growth mechanisms are disabled. In the third set (Figs. 22 and 25), the same faults are forced into the tissue with the fault-detection and growth mechanisms enabled.

Embryonic Cell: In Fig. 21, it can be seen that unprotected embryonic cell sustains a terminal fault in run 2. In run 2 of Fig. 22, on the other hand, it can be seen that with the fault-tolerant mechanisms enabled, the system has detected the induced fault and instigated apoptosis of the faulty cell and regrowth of the system. Data loss in the embryonic system (since when regrowth is triggered system functionality is recovered, but not system state) is illustrated by the zero output produced by the newly grown system. At the start of run 2, the system is repaired and fully functional but its response to an input pulse previous to the fault being detected has been wiped by regrowth. Part way through run 2, the correct system response to this input has become negligible and a new input pulse stimulates the repaired

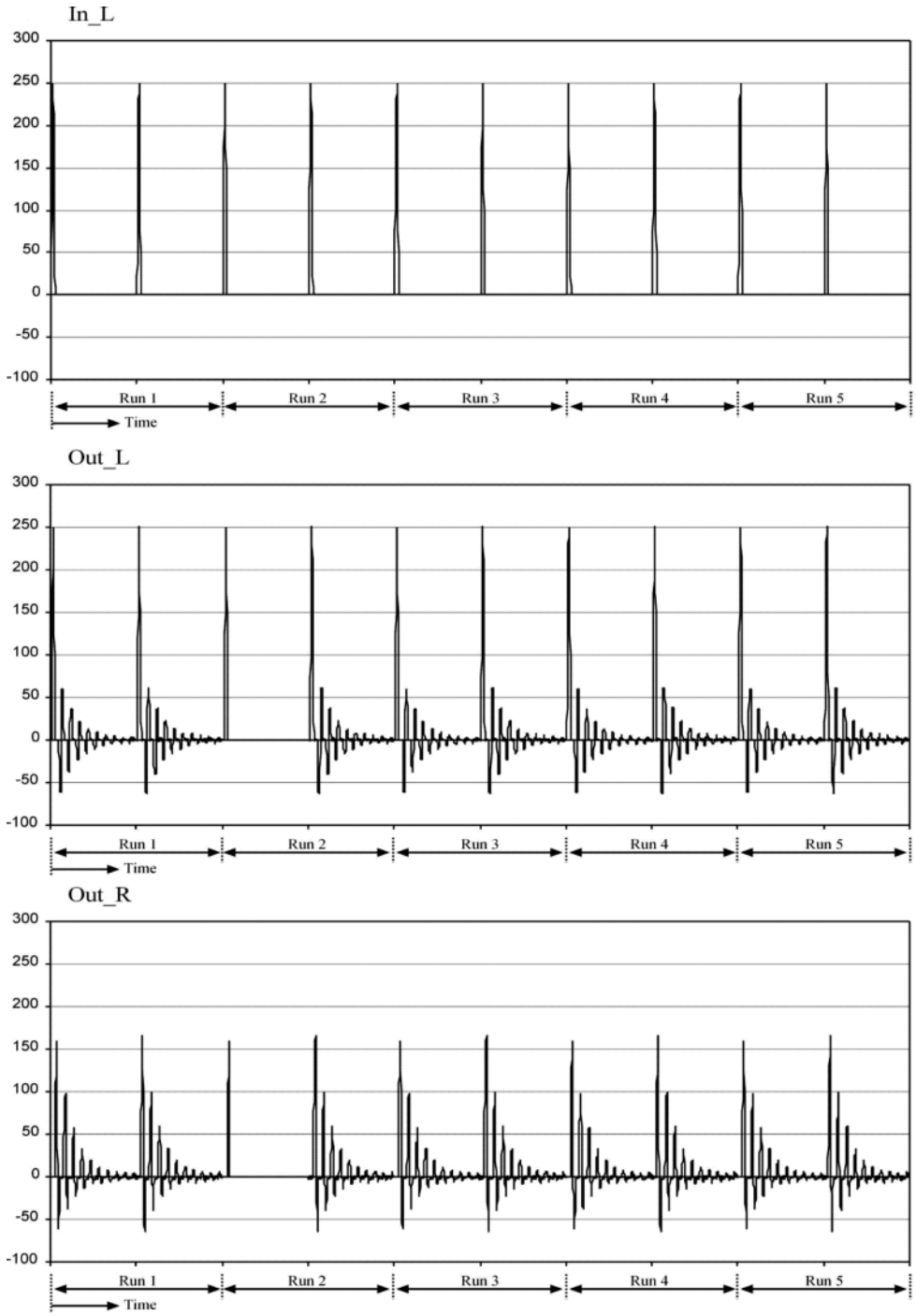

Fig. 22. Embryonic organism: faults injected. Fault-tolerance on.

embryonic system producing incorrupt output data. This issue would obviously not exist in applications that do not require previous state information for their operation.

Simple Duplication Cell: In Fig. 24, it can be seen that in runs 1, 2, and 4 the faults forced into the tissue have had no adverse effects on the output. In runs 3 and 5, however, the faults have caused the unprotected system to fail. Fig. 25 shows that in both these cases the fault-tolerant mechanisms have enabled the system to detect the induced fault and successfully export the cell function and data to the spare cell. This cell has then been integrated into the system and the correct organism output has been seamlessly maintained.

IX. Conclusion

The goal of the POEtic project was the development of a hardware platform capable of implementing bio-inspired systems in digital hardware. In particular, the final hardware device, while similar to conventional FPGAs in that it is a reconfigurable logic device capable of implementing any digital circuit, was designed with a number of novel features which fa-

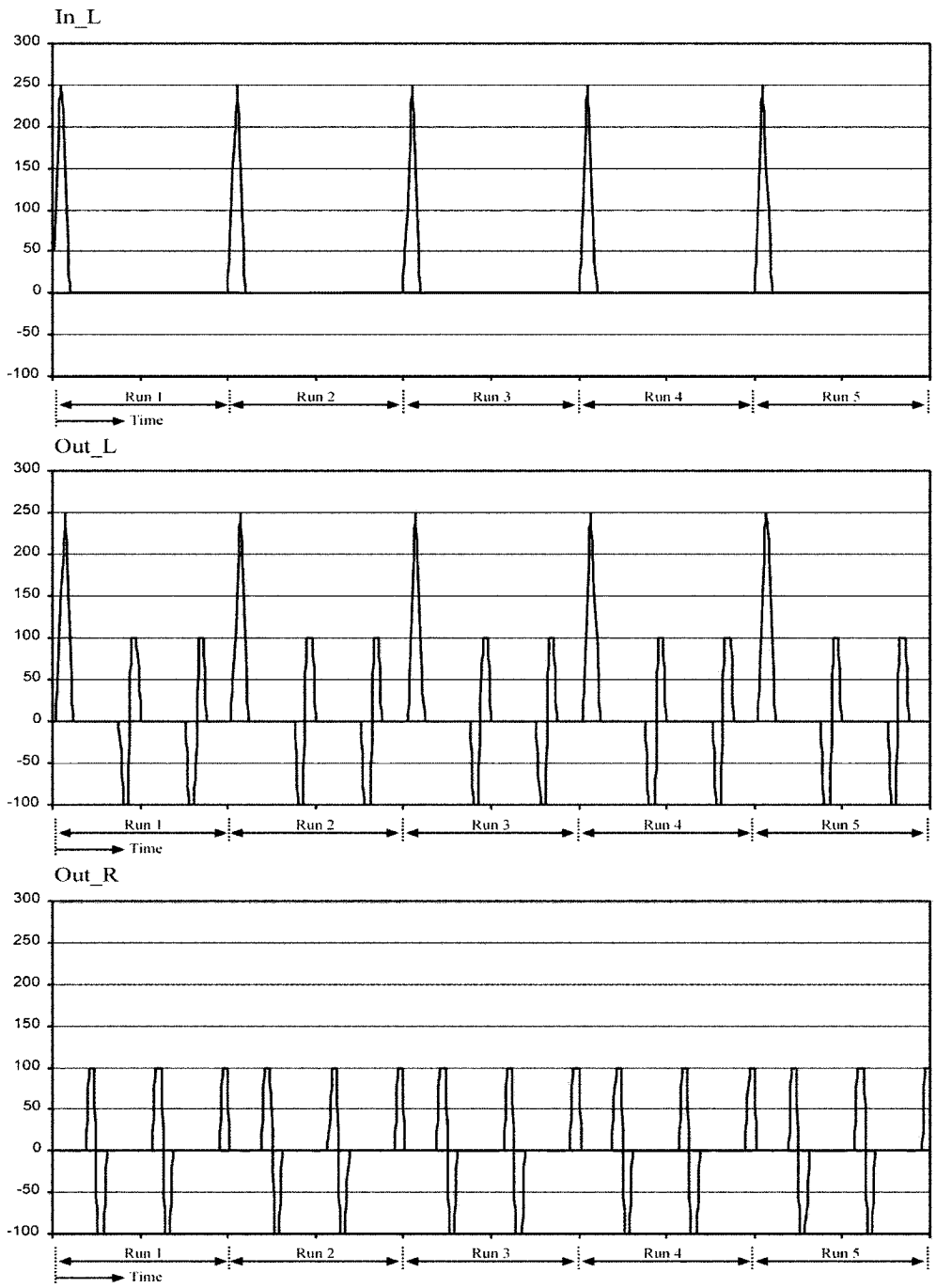

Fig. 23. Simple duplication organism: no faults injected.

cilitate the realization of bio-inspired systems in general and of fault-tolerant circuits in particular. The work reported in this paper aimed to illustrate the effectiveness of features such as dynamic reconfiguration and on-chip reprogramming for the implementation of fault-tolerant systems using an ensemble of different, but often complementary techniques. Two approaches, based on different bio-inspiration techniques, have been implemented and assessed using the same real-time audio application: embryonics and duplication.

An embryonic cellular fault-tolerant mechanism has been successfully implemented in simulation of the POEtic tissue. Unlike implementations on generic FPGA architectures that require complex stages of synthesis and careful tailoring of the embryonic architecture for the target device, compact embryonic designs can be built directly on the POEtic tissue at the molecular level. Results of preliminary simulations using randomly introduced faults show that the cell design is capable of successfully detecting, repairing, and recovering from terminal faults in the cell function.

A cellular fault-tolerant mechanism inspired by simple duplication in mammalian liver cells has also been successfully demonstrated in simulation on the POEtic tissue. The mechanism is applicable to almost any cell function and makes extensive use of the novel features for self-configuration and dynamic routing provided by the POEtic architecture. Results of preliminary simulations in the presence of randomly introduced faults

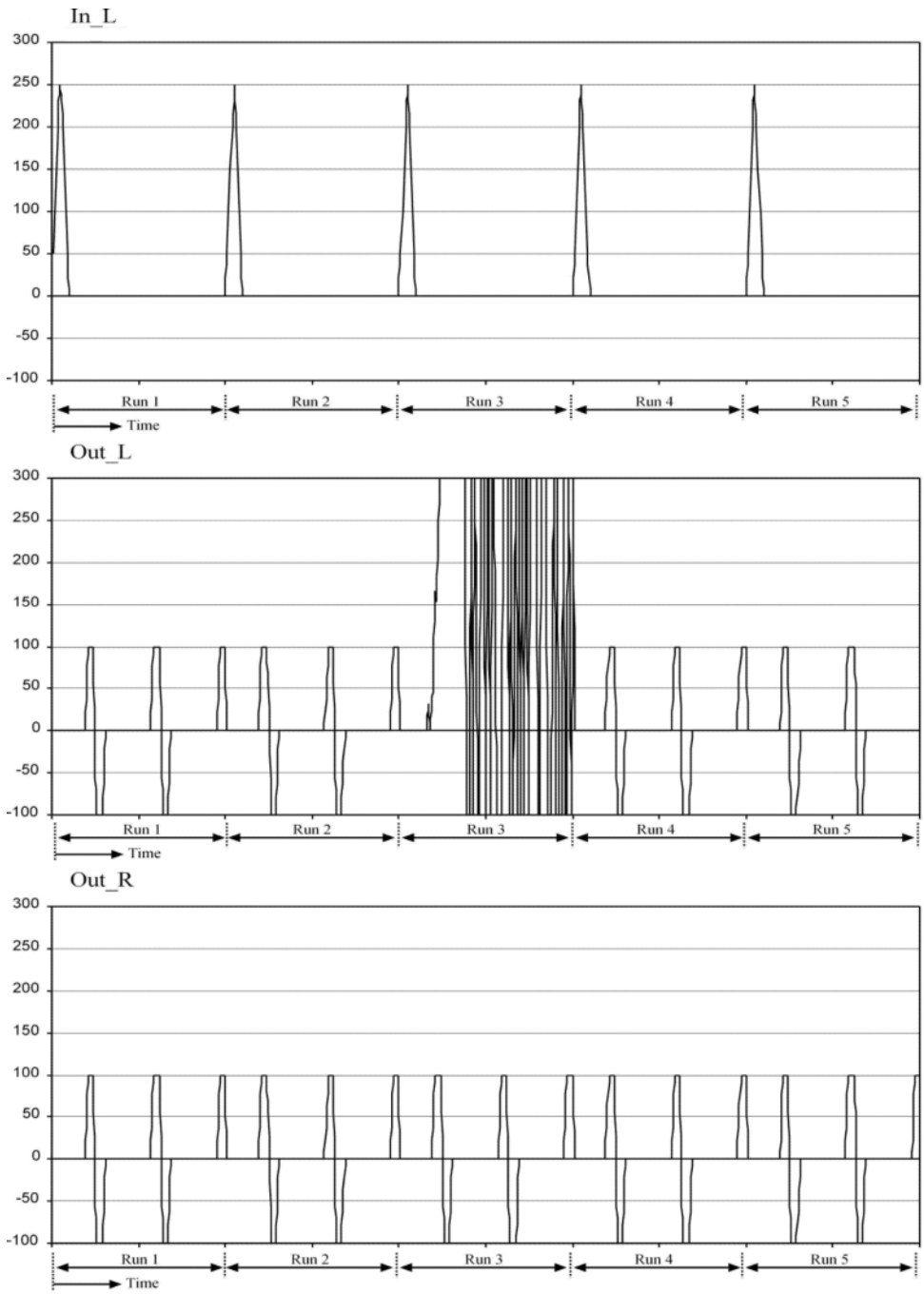

Fig. 24. Simple duplication organism: faults injected. Fault-tolerance off.

show that the cell design is capable of successfully detecting and repairing terminal faults in the cell function. The redundancy in function required for fault masking has been maintained through growth of a new cell. This process can continue for as long as healthy spare cells exist on the tissue allowing cells to survive multiple nonsimultaneous terminal faults.

The results presented in this paper belong to a broader context within the development of a design methodology for bio-inspired systems, which is finding increasing interest in computer science and electronic engineering. Systems inspired by biological mechanisms are carving themselves a niche in several areas of computation, but the solutions adopted often cannot be generalized for the lack of a universal architecture capable of integrating the different approaches. This paper has shown the effectiveness of the POEtic tissue in the specific area of fault tolerance. However, the possibilities of the tissue for research in bio-inspired hardware is not limited to this area: the POEtic device, for the first time, gives researchers in the field of bio-inspired engineering the real chance to evolve, adapt, develop, grow hardware systems, where currently they can only effectively simulate them.

The POEtic project proposes a hardware substrate capable of implementing this kind of architecture. It has developed the *molecules* necessary for the development of cellular systems. The usefulness and the practicality of the molecular surface are being validated as the number of bio-inspired applications im-

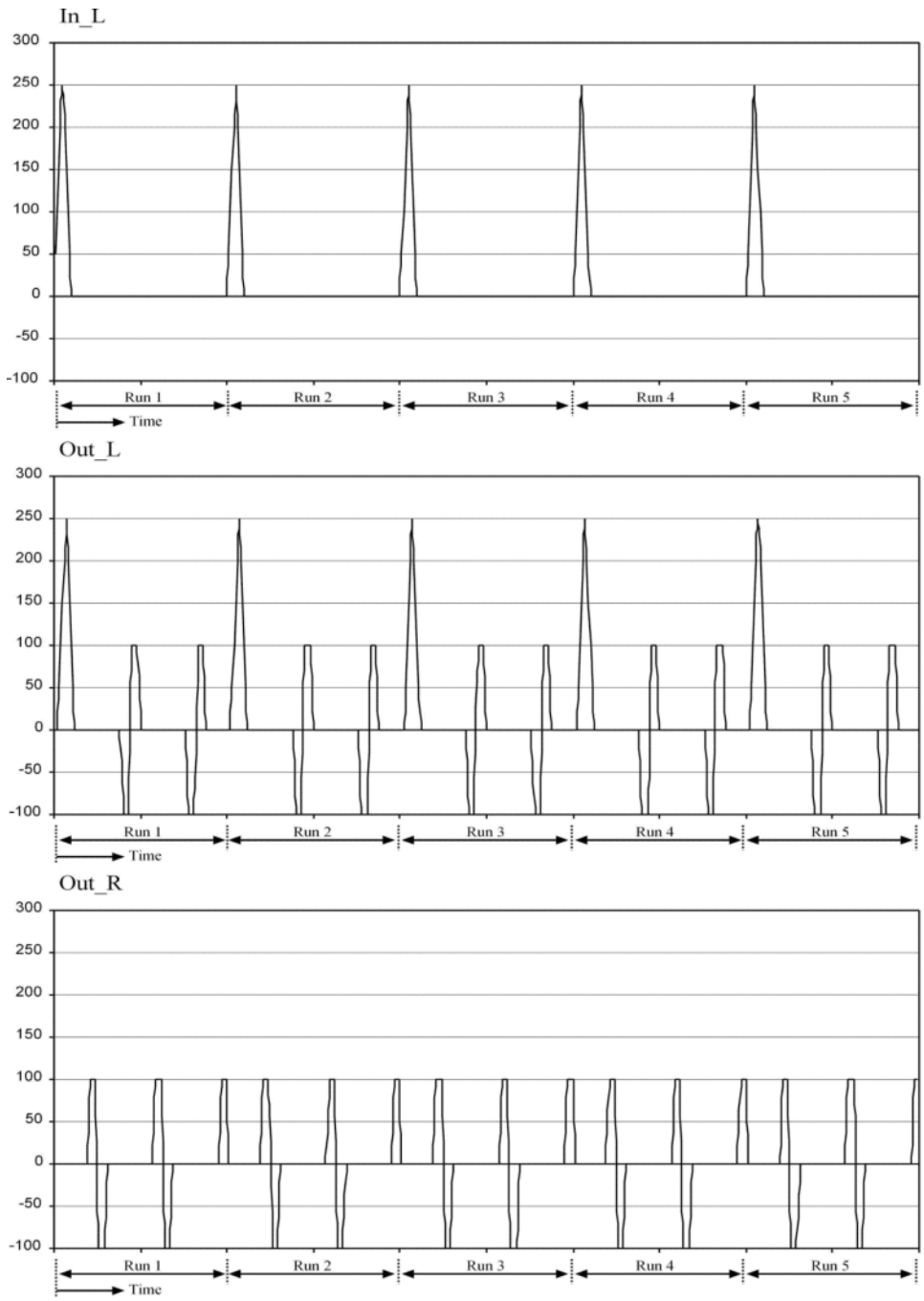

Fig. 25. Simple duplication organism: faults injected. Fault-tolerance on.

plemented on it increase. However, in addition to the applications developed within the project, the POEtic tissue will allow the implementation of a variety of bio-inspired systems, and it is our hope that the circuits created in this project will find users well beyond our group.

ACKNOWLEDGMENT

The authors would like to acknowledge the contribution of all the other members of the POEtic project, who all contributed to the material presented herein. They would also like to thank the reviewers for their helpful and incisive comments that certainly improved the paper.

REFERENCES

[1] P. Lee and T. Anderson, *Fault Tolerance Principles and Practice*, 2nd ed. Berlin, Germany: Springer-Verlag, 1990.

[2] A. M. Tyrrell, G. Hollingworth, and S. L. Smith, "Evolutionary strategies and intrinsic fault tolerance," in *Proc. 3rd NASA/DoD Workshop on Evolvable Hardware*, Jul. 2001, pp. 98–106.

[3] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti, "Towards robust integrated circuits: The embryonics approach," *Proc. IEEE*, vol. 88, no. 4, pp. 516–541, Apr. 2000.

[4] D. W. Bradley and A. M. Tyrrell, "Immunotronics: Novel finite state machine architectures with built in self test using self-nonself differentiation," *IEEE Trans. Evol. Comput.*, vol. 6, no. 3, pp. 227–238, Jun. 2002.

[5] A. M. Tyrrell, E. Sanchez, D. Floreano, G. Tempesti, D. Mange, J. M. Moreno, J. Rosenberg, and A. E. P. Villa, "POEtic Tissue: An integrated architecture for bio-inspired hardware," in *Proc. 5th Int. Conf. Evolvable Syst.*, Trondheim, Mar. 2003, pp. 129–140.

[6] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, and A. Stauffer, "Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 35–54.

[7] A. M. Tyrrell and W. Barker, "The POEtic hardware device: Assistance for evolution, development and learning," in *Evolvable Hardware (Invited Chapter)*, T. Higuchi, Y. Liu, and X. Yao, Eds. Berlin, Germany: Springer-Verlag, Jun. 2006, pp. 99–120.

[8] J.-M. Moreno, Y. Thoma, E. Sanchez, O. Torres, and G. Tempesti, "Hardware Realization of a Bio-inspired POEtic tissue," in *Proc. 2004 NASA/DoD Conf. Evolvable Hardware*, , 2004, pp. 237–244.

[9] Y. Thoma, E. Sanchez, J.-M. Moreno, and G. Tempesti, "A dynamic routing algorithm for a bio-inspired reconfigurable circuit," in *Proc. 13th Int. Conf. Field Program. Logic Appl.*, 2003, vol. 2778, pp. 681–690, LNCS.

[10] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Electron. Comput.*, vol. EC-10, no. 3, pp. 346–365, Sep. 1961.

[11] W. Barker and A. M. Tyrrell, in *Proc. 6th Int. Conf. Evolvable Syst., Hardware Fault Tolerance Within the POEtic System*, Barcelona, Spain, Sep. 2005, pp. 25–36, LNCS 3637.

[12] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. Watson, "Molecular biology of the cell," *Garland Science*, 2002.

[13] P. K. Lala, *Digital Circuit Testing and Testability*. New York: Academic, 1997.

[14] H. Liu, J. F. Miller, and A. M. Tyrrell, "A biological development model for the design of robust multiplier," in *Proc. 7th Eur. Workshop Evol. Comput.*, Lausanne, Switzerland, Mar. 2005.

[15] C. Ortega and A. M. Tyrrell, *MUXTREE revisited: Embryonics as a Reconfiguration Strategy in Fault-Tolerant Processor Arrays*. Berlin, Germany: Springer-Verlag, 1998, vol. 1478, Lecture Notes in Computer Science, pp. 206–217.

[16] Y. Thoma, E. Sanchez, C. Hetherington, D. Roggen, and J.-M. Moreno, "Prototyping with a bio-inspired reconfigurable chip," in *Proc. 15th Int. Workshop Rapid Syst. Prototyping (RSP 2004)*, Geneva, Switzerland, Jun. 2004.

[17] D. J. Costello Jr. and S. Lin, *Error Control Coding*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004, pp. 136–188.

[18] C. Cooper, D. Howard, and A. Tyrrell, "Singing synthesis with an evolved waveguide mesh model," *IEEE Trans. Speech and Audio Process.*, vol. 14, no. 4, pp. 1454–1461, Jul. 2006.

Will Barker received the Degree in mechatronics (Hons.) from the University of Leeds, Leeds, U.K., in 1997. He is currently working towards the Ph.D. degree in electronic/computing disciplines at Leeds Metropolitan University, Leeds.

From October 2003 to October 2004, he worked as a Research Assistant in the Bio-Inspired Architectures Laboratory, University of York, where he carried out research into fault-tolerant cellular implementations as part of the POEtic project. His main research interests are in fault tolerance in biologically inspired computing architectures, evolvable hardware, evolutionary computing, inverse problems, and optimization.

David M. Halliday received the B.Sc. and Ph.D. degrees in electronics and electrical engineering from the University of Glasgow, Scotland, U.K.

He has held research positions at the University of Glasgow, University of Strathclyde, and NTT Basic Research Laboratories, Tokyo. His research interests are in computational neuroscience and neural computing.

Yann Thoma received the M.Sc. degree in computer science and the Ph.D. degree in science from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, in 2001 and 2005.

He is currently a Lecturer and a Postdoctoral at the University of Applied Sciences Western Switzerland and at EIG, Geneva, and HEIG-VD, Yverdon, respectively. His research interests include reconfigurable hardware, bio-inspired systems, and embedded digital systems.

Eduardo Sanchez received the diploma in electrical engineering from the Univerdsidad del Valle, Cali, Colombia, in 1975, and the Ph.D. degree from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, in 1985.

He is currently a Professor of Computer Science at the EPFL and Head of the Computer and Communications Department of the HEIG-VD (Yverdon, Switzerland), where he is engaged in teaching and research. His chief interests include computer architecture, reconfigurable computing, and bio-inspired computing.

Gianluca Tempesti received the B.S.E. degree in electrical engineering from Princeton University, Princeton, NJ, in 1991, the M.S.E. degree in computer science and engineering from the University of Michigan at Ann Arbor, in 1993, and the Ph.D. degree from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, in 1998, with a thesis on the design of fault-tolerant bio-inspired FPGAs.

He joined the Department of Electronics of the University of York, York, U.K., as a Reader in 2006. His research interests include bio-inspired digital hardware, built-in self-test and self-repair, programmable logic, and cellular automata. He is the author of over 75 articles in these areas.

Dr. Tempesti received a Young Professorship Award from the Swiss National Science Foundation (FNS) in 2003, and created the Cellular Architecture Research Group (CARG). He was Program Chair of the Von Neumann Day (Lausanne, 1997), and Program Co-Chair of the Sixth International Conference on Evolvable Systems (Barcelona, September 2005), as well as member of the program and scientific committees of several international conferences.

Andy M. Tyrrell (SM'96) received a first class honors degree in 1982 and the Ph.D. degree in 1985, both in electrical and electronic engineering, from Aston University, U.K.

He joined the Electronics Department, University of York, in April 1990, where he was promoted to the Chair in Digital Electronics in 1998. Previous to that he was a Senior Lecturer at Coventry Polytechnic. From August 1987 and August 1988, he was Visiting Research Fellow at the Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, where he was researching into the evaluation and performance of multiprocessor systems. From September 1973 to September 1979, he worked for STC at Paignton Devon, on the design and development of high-frequency devices. He is Head of the Intelligent Systems research group at the University of York. His main research interests are in the design of biologically inspired architectures, artificial immune systems, evolvable hardware, FPGA system design, parallel systems, fault tolerant design, and real-time systems. In particular, over the last eight years his research group at York have concentrated on bio-inspired systems. This work has included the creation of embryonic processing array, intrinsic evolvable hardware systems, and the immunotronics hardware architecture.

Prof. Tyrrell is a Fellow of the IET. He was General Program Chair for ICES 2003 and Program Chair for IPCAT 2005. He has published over 200 papers in these areas, and has attracted funds in excess of £2.6M.