Information Society Technologies

Project Number:	**034632**
Acronym:	**PERPLEXUS**
Title:	**PERVASIVE COMPUTING FRAMEWORK FOR MODELING COMPLEX VIRTUALLY-UNBOUNDED SYSTEMS**
Instrument:	SPECIFIC TARGETED RESEACH PROJECT
Thematic Priority:	INFORMATION SOCIETY TECHNOLOGIES
Start date:	01/09/2006
Duration:	36 months

D2.1: BIO-INSPIRED FEATURES OF THE UBICHIP

Nature:[1] R

Dissemination level:[2] PU

Due date: Month 5

Date of delivery: Month 5

Partners involved (leader in bold): **HEIG-VD**, UPC, CNRS, UNIL, UJF

Authors: Andres Upegui, Yann Thoma, Juan Manuel Moreno

[1] R = Report, P = Prototype, D = Demonstrator, O = Other

[2] PU = Public, PP = Restricted to other programme participants (including the Commission Services), RE = Restricted to a group specified by the consortium (including the Commission Services), CO = Confidential, only for members of the consortium (including the Commission Services)

Contents

Chapter 1

Introduction

The PERPLEXUS project aims to develop a scalable hardware platform made of custom reconfigurable devices endowed with bio-inspired capabilities that will enable the simulation of large-scale complex systems and the study of emergent complex behaviors in a virtually unbounded wireless network of computing modules.

At the heart of these *ubiquitous computing modules (ubidules)*, we will use a custom reconfigurable electronic device capable of implementing bio-inspired mechanisms such as growth, learning, and evolution. This *ubidule bio-inspired chip* (ubichip) will be associated to rich sensory elements and wireless communication capabilities. Such an infrastructure will provide several advantages compared to classical software simulations: speed-up, an inherent real-time interaction with the environment, self-organization capabilities, simulation in the presence of uncertainty, and distributed multi-scale simulations.

The strong interaction between our hardware infrastructure and the real environment circumvent the need to simulate the environment and ease the occurrence of unexpected emergent phenomena. The observation of such emergent phenomena will be now facilitated by the shorter simulation time, brought by the hardware speed-up.

One of the major difficulties of a complex system simulation is to define the structural organization of the modules composing the model. The self-organization and bio-inspired capabilities of our platform will bring an innovative solution to this problem: an evolving and hierarchical structure. The function of each ubidule can be dynamically and autonomously determined by the simulation itself: it can be an independent agent or a part of a largest entity.

We have identified three domains where our modeling infrastructure will prove its usefulness as a powerful and innovative simulation tool: biologically-plausible developing neural networks modeling, culture dissemination modeling, and cooperative collective robotics modeling. We will perform qualitative and quantitative comparisons between classical software implementations of these three target modeling applications and their implementation running on a net-

work of ubidules, our PERPLEXUS platform.

The PERPLEXUS platform will thus provide an unprecedented modeling framework thanks to the pervasive nature of the hardware platform, its bio-inspired capabilities, its strong interaction with the environment, and its dynamical topology.

We envision three strategic objectives to be addressed

1. Design and development of a scalable hardware platform made of custom reconfigurable devices endowed with bio-inspired capabilities that will enable the simulation of large-scale complex systems and the study of emergent complex behaviors in a virtually unbounded wireless network of computing modules.

2. Simulation of complex phenomena in the domains of realistic neural models, social sciences, and collective cooperative robotics, taking advantage of the features of our modeling hardware platform: speed-up, distributed computing, an inherent real-time interaction with the environment, self-organization capabilities, and simulation in the presence of uncertainty.

3. To study the emergent phenomena arising from the strong interaction between our ubiquitous computing modules and the real environment.

Chapter 2

Bio-inspired hardware

2.1 About Bio-inspiration

Living organisms, from bacteria to giant sequoias, including animals such as insects and humans, have successfully survived on earth for billions of years. If one were to propose but one key to explain such a success, it would certainly be *adaptation*. In contrast with nature, adaptation has been very elusive to human technology. The most relevant examples of adaptive systems are not among human's creations, but among nature's. Biological organisms show a striking capacity to adapt to changing circumstances, thus ensuring their continued functionality.

Nature has always stimulated the imagination of humans, but it is only very recently that technology is allowing the physical implementation of bio-inspired systems. They are man-made systems whose architectures and emergent behaviors resemble the structure and behavior of biological organisms [22]. Artificial neural networks (ANNs), evolutionary algorithms (EAs), and fuzzy logic are some representatives of a new, different approach to artificial intelligence. Names like "computational intelligence", "soft computing", "bio-inspired systems", or "natural computing", among others, are used to denominate the domain involving these and other related techniques. Whatever the name, these techniques exhibit the following features: (1) their role models, to different extents, are natural processes such as evolution, learning, development, or reasoning; (2) they are intended to be tolerant of imprecision, uncertainty, partial truth, and approximation; (3) they deal mainly with numerical information processing using little or no explicit knowledge representation.

How to model life? How to integrate all these bio-inspired techniques to a single model? How to merge these techniques in order to create an entity able to mimic living beings? These are open questions that are still far from being completely answered. There exist several research fields deeply studying and proposing computational models of specific aspects of biological systems. Neu-

rocomputing, evolutionary computation, and fault-tolerant systems are some examples of them. However, modelling life implies including them all in a single model, for which the POE model proposes a well structured framework, which is also well suited to the implementation of real systems.

2.1.1 The POE model

If one considers life on Earth since its very beginning, then the following three levels of organization can be distinguished [37]: (1) Phylogeny, concerning the temporal evolution of a certain genetic material in individuals and species, (2) Epigenesis, concerning the learning process during an individual's lifetime, and (3) Ontogeny, concerning the developmental process of multicellular organisms.

Analogous to nature, the space of artificial bio-inspired systems can be partitioned along these three axes: phylogeny, ontogeny, and epigenesis; we refer to this as the POE model [37, 38]. The distinction between the axes cannot be easily drawn where nature is concerned. We therefore define each of the above axes within the framework of the POE model as follows: the phylogenetic axis involves *evolution*, the ontogenetic axis involves the *development* of a single individual from its own genetic material, essentially without environmental interactions, and the epigenetic axis involves *learning* through environmental interactions that take place after formation of the individual. As an example, consider the following three paradigms, whose hardware implementations can be positioned along the POE axes: (P) EAs are the simplified artificial counterpart of phylogeny in nature, (O) self-replicating and self-repairing cellular automata are based on the concept of ontogeny, where a single mother cell gives rise, through multiple divisions, to a multi-cellular organism, and (E) ANNs embody the epigenetic process, where the system's synaptic weights change through interactions with the environment. Within the domains collectively referred to as bio-inspired systems, which often involves the solution of ill-defined problems coupled with the need for continual adaptation or evolution, the above paradigms yield impressive results, frequently improving upon those of traditional methods.

2.1.2 Phylogeny

The first level concerns the temporal evolution of the genetic program, the hallmark of which is the evolution of species, or *phylogeny*. The multiplication of living organisms is based upon the reproduction of the program, subject to an extremely low error rate at the individual level, so as to ensure that the species of the offspring remain unchanged. Mutation (asexual reproduction) or mutation along with recombination (sexual reproduction) gives rise to the emergence of new organisms. The phylogenetic mechanisms are fundamentally nondeterministic, with the mutation and recombination rate providing a major source of diversity. This diversity is indispensable for the survival of living species, for their continuous adaptation to a changing environment, and for the appearance of new species.

The idea of applying the biological principle of natural evolution to artificial systems, introduced more than three decades ago, has seen impressive growth in the past few years. Usually grouped under the term *evolutionary algorithms* (EAs) or *evolutionary computation*, we find the domains of GAs, evolution strategies, evolutionary programming, and genetic programming [4, 10, 21, 27]. EAs can be also considered as a family of stochastic global optimization algorithms, mainly differing from their deterministic counterparts [32] in lower knowledge requirements of the problem at hand and in the absence of mathematical proofs of convergence given their stochastic nature. For highly non-linear search spaces, EAs have exhibited faster convergence than deterministic methods, given their population-based approach.

Evolutionary computation makes use of a metaphor of natural evolution according to which a problem plays the role of an environment wherein lives a population of individuals, each representing a possible solution to the problem. The degree of adaptation of each individual to its environment is expressed by an adequacy measure known as the fitness function. The phenotype of each individual, i.e., the candidate solution itself, is generally encoded in some manner into its genome (genotype). EAs potentially produce progressively better solutions to the problem. This is possible thanks to the constant introduction of new "genetic" material into the population, by applying so-called genetic operators which are the computational equivalents of natural evolutionary mechanisms.

As they combine elements of directed and stochastic search, evolutionary techniques exhibit a number of advantages over other search methods. First, they usually need a smaller amount of knowledge and fewer assumptions about the characteristics of the search space. Second, they are less prone to get stuck in local optima. Finally, they strike a good balance between exploitation of the best solutions, and exploration of the search space.

EAs are common at present, having been successfully applied to numerous problems from different domains as diverse as optimization, circuit design, disease diagnosis assistance, precision agriculture, self-organizing systems, automatic programming, machine learning, economics, immune systems, ecology, population genetics, studies of evolution and learning, and social systems [21].

2.1.3 Ontogeny

Upon the appearance of multi-cellular organisms, a second level of biological organization manifests itself. This level constitutes the developmental process of multi-cellular organisms, best known as *ontogeny*. The successive divisions of the mother cell, the zygote, into newly formed cells each possessing a copy of the original genome, is followed by a specialization of the daughter cells in accordance with their surroundings, i.e. their position within the ensemble. This latter phase is known as cellular differentiation. The ontogenetic process is essentially deterministic: an error in a single base within the genome can provoke an ontogenetic sequence that results in notable, possibly lethal, malformations.

Ontogeny comprises several mechanisms of high interest for inclusion in human-designed systems. Self-replication and self-reparation are two key char-

acteristics of living beings that are still far from being implemented in engineered systems with an efficiency comparable to nature. However, some key factors from multicellular beings have been identified for use in the design of ontogenic machines: a cell's function depends upon its relative position, the physical neighborhood is relevant for chemical interactions between cells, time scales are determinant during cellular reproduction, and the fundamental role played by protein's regulation and cell's differentiation, which is driven by regulatory and differentiation genes.

Research projects as *Embryonics* [25] (embryonic electronics) and *POEtic* [47, 42, 41, 45] have studied the issues related to hardware implementations of such mechanisms.

The Embryonics project take inspiration from the genome interpretation done by each cell composing living beings. This project aims to build robust integrated circuits endowed with two fundamental properties of living beings: self-repair and self-replication. For achieving this, they propose a hardware system with several levels of organization. The lowest level is a molecule consisting in a multiplexer. The next level consists in a cell, represented by a set of molecules forming a processor with its memory. Then, a set of cells forms an organism, or in hardware, a multiprocessor system. Finally, this organism can itself replicate, generating in this way a population of organisms.

2.1.4 Epigenesis

The ontogenetic program is limited in the amount of information that can be stored, thereby rendering the complete specification of the organism impossible. A well-known example is that of the human brain with some 10^{11} neurons and 10^{14} synapses, far too large a number to be completely specified in the four-character genome with an approximate length of 3×10^9. Therefore, upon reaching a certain level of complexity, there must emerge a different process that permits the individual to integrate the vast quantity of interactions with the outside world. This process is known as *epigenesis* and primarily includes the nervous system, the immune system, and the endocrine system. These systems are characterized by the possession of a basic structure that is entirely defined by the genome (the innate part), which is then subjected to modification through lifetime interactions of the individual with the environment (the acquired part). The epigenetic processes can be grouped under the heading of *learning* systems and, in bio-inspired systems, it is mainly represented by the domain of ANNs.

Artificial neural networks (ANNs) are massively parallel distributed computing units made up of very simple basic elements. They provide the feature of storing experiential knowledge making it available for future use. ANNs takes inspiration from animals' brains in several aspects: they benefit from a massively parallel cellular architecture, a learning process allows acquiring a certain knowledge, and this knowledge is stored in the form of synaptic weights interconnecting neurons. Among other computation features, ANNs provide nonlinearity (an ANN made up of nonlinear neurons has a natural ability to compute nonlinear input-output functions), they are universal approximators (ANNs can

approximate input-output functions to any desired degree of accuracy, given an adequate computational complexity), they are adaptable (adjustable synaptic weights and network topology can adapt to its operating environment and track statistical variations), they are fault tolerant (an ANN has the potential to be fault-tolerant, or capable of robust performance, in the sense its performance degrades gradually under adverse operating conditions), and they can mimic real neurons (neurobiologists look to neural networks as a research tool for the interpretation of neurobiological phenomena. By the same token, engineers look to the human brain for new ideas to solve difficult problems) [13].

2.2 POEtic chip

There has been considerable research involving one or more of these three life-axes. Most of it is done in software, due to the lack of a real hardware platform specifically designed for such applications. However, hardware implementations can dramatically improve the speed of neural networks and fuzzy logic systems, for instance, by taking advantage of the inherent parallelism of hardware systems. Furthermore, self-repair mechanisms can only be realized in hardware, as a single microprocessor is intrinsicaly not fault-tolerant, due to its centralized calculation.

The POEtic chip [29] is a reconfigurable hardware platform for rapidly prototyping bio-inspired systems that employ POE principles, developed in the framework of the european project POEtic. In this section we briefly describe the chip, with a special attention on its bio-inspired capabilities. We then describe four applications which take advantage of the special features of POEtic, and conclude by addressing the potential improvements that could be proposed concerning its architecture. The special features can be summarized as:

- combination of a microprocessor and a reconfigurable array
- parallel configuration
- partial reconfiguration
- dynamic routing

The POEtic chip has been specifically designed to ease the development of bio-inspired applications. It is composed of two main parts: a microprocessor, in the environmental subsystem, and a 2-dimensional reconfigurable array called the organic subsystem (figure 2.1). This array is made of small elements, called molecules, that are essentially composed of a 4-input look-up table and a flip-flop.

Although being oriented for bio-inspired systems, its architecture makes it an appropriate candidate for any general design as the microprocessor can access the reconfigurable array very rapidly. This can be useful for both configuration and state retrieval.

The final chip contains an array of 18x8 molecules and the microprocessor. This ASIC prototype uses a CMOS 0.35 μm 1P-5M technology.

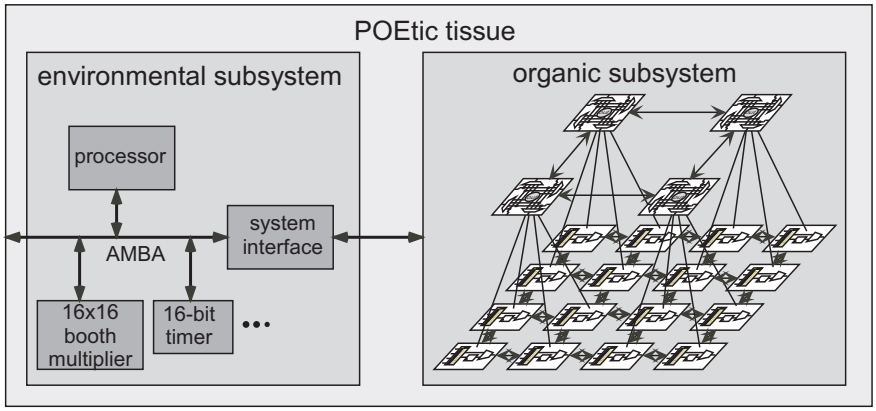

Figure 2.1: The POEtic chip, showing the microprocessor and the reconfigurable array. In the organic subsystem, the molecular plane (bottom) is connected to the routing plane (top). Many elements connected to the AMBA bus, (another timer and serial and parallel ports) are omitted in order to simplify the schematics.

2.2.1 The Microprocessor

The microprocessor is a 32-bit RISC processor, specifically designed for the POEtic chip. Its purpose is to control the organic subsystem, including the configuration of molecules, as well as to execute evolutionary processes. During the design process, particular attention was paid to the microprocessor size so as to leave more room for the reconfigurable array.

The main features of the POEtic microprocessor are as follows:

- The architecture is LOAD/STORE.
- Every instruction is 32 bits.
- Every instruction is executed in one clock cycle.
- A five-stage pipeline implements the datapath, with the following states: Fetch, Decode, Execute, Memory, and Writeback.
- 57 instructions are defined, two of which give access to a hardware pseudo-random number generator (a read instruction, and the load of an initial seed) which can be very useful for evolutionary processes. This generator has been implemented using a 32-bit linear feedback shift register.
- Up to 5 interrupt sources can be handled by the microprocessor.
- An AMBA bus [2] allows communication with all internal elements, as shown in figure 2.1, as well as with external devices. It also permits the interconnection of many POEtic chips, in order to realize a larger virtual reconfigurable array.

The microprocessor can configure the array, and also retrieve its state. Access is made in parallel, so configuration and partial reconfiguration are very fast. The retrieved state can be used to calculate the fitness of an individual, in the case of an evolutionary process, or simply to debug any design running on POEtic. For genetic algorithms, evolution can be performed by the microprocessor. This obviates the need for slow data transmission to and from a host computer.

2.2.2 The Reconfigurable Array

The organic subsystem is composed of two layers: the molecular layer, that is reconfigured by the microprocessor, and the routing layer which implements a dynamic routing algorithm managed by the molecules.

The molecular layer is a grid of basic elements, called molecules. Although being similar to standard FPGA elements, molecules have special features which are useful for bio-inspired systems. The main components are a 4-input look-up table, a flip-flop, and a switch box, as depicted in figure 2.2.

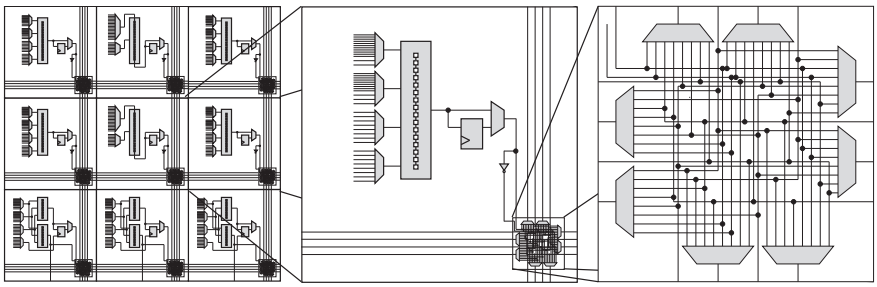

Figure 2.2: On the left, 9 molecules of the reconfigurable array. In the centre, a molecule in 4-LUT mode. On the right, the switch box of a molecule.

The switch box allows the connection of molecules that are not adjacent to one another. It is composed of eight multiplexers — two in each direction. Each multiplexer can select from the two signals coming from each direction, the output of the molecule, or a second output. The second output is, in most cases, the inverse of the first. The switch box has been designed with multiplexers, rather than with anti-fuse or RAM bits, in order to avoid any short-circuit. This feature means that a developer can use POEtic as platform for evolvable hardware without any risk, as no randomly generated bitstream configuration can destroy the chip.

The molecule can act in different operating modes (figure 2.3):

- In 4-LUT mode, the output is any function of the four inputs.
- In 3-LUT mode, two outputs are computed, each from any 3-input function.

- In Comm mode, the LUT is split into a 8-bit shift register and a 3-input LUT.
- In Memory mode, the LUT is used as a 16-bit shift register, and can be used to implement a serial access memory.
- In Input mode, the molecule retrieves a value from the dynamic routing layer.
- In Output mode, the molecule sends a value to the dynamic routing layer.
- In Configure mode, the molecule can partially reconfigure a neighbouring molecule.
- In Trigger mode, the molecule serves as a trigger to synchronize the dynamic routing process.

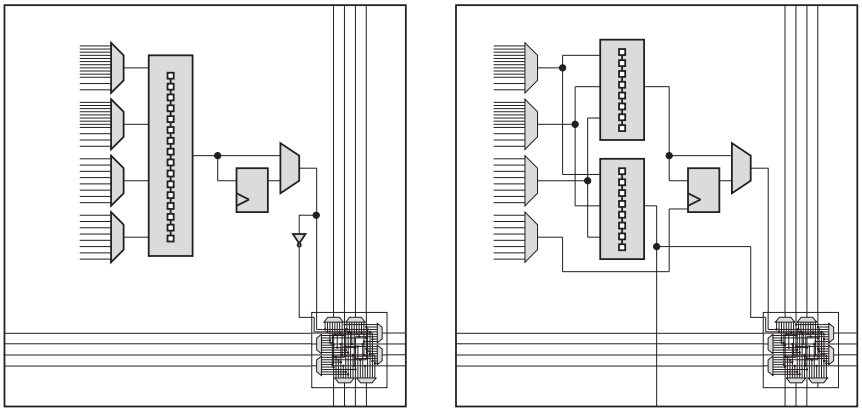

Figure 2.3: 3 bits define the operational mode of a molecule. On the left, a molecule in 4-LUT mode. On the right, a molecule in 3-LUT mode.

One of the special features of the reconfigurable array is that molecules can be partially reconfigured without microprocessor intervention. A molecule configuration is described by 76 bits, split into 5 blocks. The molecule can allow a reconfiguration of each of its blocks, and chooses the source of the configuration data. A partial reconfiguration is processed when a molecule in Configure mode is active; this is when its first input is active. In this state, configuration bits are shifted on every clock cycle, with the second input of the Configure molecule being sent out as the new configuration bitstream. This feature allows LUT content or dynamic routing addresses to be changed at runtime. It can be very useful for self-repair systems, in which the array can partially reconfigure itself without needing an external agent.

The routing layer is a grid of routing units, which can dynamically create paths between different points of the circuit at runtime. It implements a distributed pseudo-dynamic routing algorithm, based on addresses (interested readers

can see a description of this algorithm in [46]). This algorithm is dynamic in the sense that a process creates the paths at runtime, but it is not fully dynamic as when a path has been defined, the data from the source to the target always follow the same path. It can be used to create connections between any parts of the circuit, by using the input/output molecules. In a cellular system (e.g. a neural network), for instance, cells could be identified by a unique ID and then connected to other cells by means of this mechanism. As the path creation is made at runtime, and can be made incrementally, POEtic is an appropriate architecture on which to grow neural networks, or any other system involving a changing topology.

2.2.3 Sample applications

The POEtic circuit has been evaluated with different applications bringing cellular systems into play. We briefly describe five of them that show phylogenetic, ontogenetic and epigenetic concepts.

1. Evolvable hardware. Evolvable hardware consists of letting a system find a solution to a given problem, without human intervention. It can be done by supplying inputs and analysing the outputs to evolve to a design satisfying all cases. A demonstration of how POEtic can be used for gate-level evolvable hardware is proposed in [43]. A subset of the possible configurations allow to see POEtic as if it was a XC6200 FPGA, the most often used reconfigurable circuit for these kinds of applications.

2. Evolvable hardware and ontogeny. Concerning ontogeny, the concept of hardware evolution mixed with growth and differentiation of a cellular system has been developed in [35]. In this application, for the evaluation of each individual, the reconfigurable part of the circuit is loaded with identical cells. The microprocessor launches the growth process by connecting to a cell, and then the cells are able to differentiate to acquire a specialization that corresponds to a part of the genome stored in the cell (identical in every cell).

3. Self-reproducing system. Partial self-reconfiguration allow the molecules to change the configuration bits of other molecules of the circuit. A cell, composed of molecules, can therefore build a copy of itself, somewhere in the circuit, at the only condition of the presence of some molecules (around 10) that are capable to retrieve a configuration chain and to configure their neighbors. A demonstration of this concept can be found in [36], where the reconfigurable part of POEtic has been implemented on the BioWall [40], a giant wall of FPGAs with interaction capabilities.

4. Artificial Neural networks. The model of an artificial spiking neuron capable of learning has been presented in [28]. Implemented onto the molecules of the POEtic chip, it allows for the observation of unsupervised learning performed by the neural network.

5. Vocal tract synthesis. As a last example, [8] describes how to exploit the capabilities of POEtic to implement a voice tract model using a mesh. Composed of cells connected to their four neighbors, evolution is applied to the shape of the individual. The mesh receives noise as input, and has to create a

sound corresponding to the voice of a human. Evolution is therefore exploited to find a good solution for each different sound, and the cellular system is implemented in the reconfigurable part of POEtic.

2.2.4 Conclusion

Common reconfigurable circuits lack adaptability, as their configuration is fixed once, by an external processor, for a specific application. In this section we presented POEtic, an electronic circuit that proposes special features particularly useful for applications that bring cellular applications into play, and that require hardware adaptation. This adaptation can act at different levels: a microprocessor has the possibility of applying evolutionnary algorithms to the system and the reconfigurable part can partially self-configure itself and change its connection topology at runtime.

The circuit, as a prototype, does not contain an impressive amount of logic (144 molecules). However, it is possible to connect many chips on a board to have a virtually larger array at one's disposal. On the neural networks side, the chip allows for the realization of neural networks that would need to change their topology at runtime. Taking advantage of the pseudo-dynamic routing algorithm of POEtic, neurons can ask for the creation of new datapaths within the chip, and by using self-reconfiguration, they can change their behavior by directly act on the content of look-up tables contained in the basic building blocks (the molecules) of the neuron.

Although being really different from the the commercial devices, this reconfigurable circuit doesn't constitutes the absolute chip for bio-inspired application:

1. The pseudo-dynamic routing algorithm requires long-distance combinational links. For scalability reasons, an algorithm working with more local connectivity would better suite our goal of developing a scalable platform.

2. The partial self-reconfiguration of the molecules allows for replication of part of the circuit. However, as there are 8 configuration bits per molecules that cannot be modified, a replication process can be executed only if the 8 bits of the receiving molecules are already programmed. For a real self-replication, the modification of all configuration bits of the programmable elements would be mandatory.

3. Concerning the implementation of artificial neural networks, some special hardware operations could be facilitated.

Taking into account these possible improvements, the *u*bichip could therefore go further into the ultimate bio-inspired chip, allowing to better perform the mechanisms needed for phylogenetic, ontogenetic and epigenetic applications.

Chapter 3

Bio-inspired features

The limitations exhibited by the POEtic tissue suggest several architectural and configurability features to be improved. These improvements may lead us to a reconfigurable platform better suited for supporting the bio-inspired principles that we want our devices to mimic. Before discussing the hardware mechanisms that we will be considered for the design of our *u*bichip, we will present the desired features that we want them to support.

In the next sections we will delineate the bio-inspired features that will be supported by our *u*bichip. Each of them is presented in the framework of the POE model, more precisely, in the space of bio-inspired hardware systems.

3.1 Phylogeny

When we refer to the phylogenetic axis of bio-inspired hardware systems, we are certainly talking about "evolvable hardware" (EHW). If one carefully examines the work carried out to date under the heading EHW, one can identify several levels at which the evolution is performed. According to [38], one can identify four taxonomic subdivisions according to the level of bio-inspiration: extrinsic, intrinsic, complete, and open-ended evolution. Our goal, concerning the phylogenetic features of the *u*bichip, is to provide the mechanisms for tackling these four taxonomic subdivisions. It is important thus to examine the scope of each of these subdivisions:

- At the bottom of this axis we find **extrinsic evolution**, what is in essence evolutionary circuit design, where all operations are carried out in software, with the resulting solution possibly loaded into a real circuit. Though a potentially useful design methodology, this falls completely within the realm of traditional evolutionary techniques.
- Moving upward along the axis one finds **intrinsic evolution**, where a real circuit is used during the evolutionary process for fitness computation,

though most operations are still carried out offline, in software. Examples are [48, 49, 17], where fitness calculation is carried out on a real circuit.

- Still further along the phylogenetic axis, one finds systems in which genetic operations (selection, crossover, mutation), as well as fitness evaluation, are carried out intrinsically, in hardware. This category has been called **complete evolution** by Haddow and Tufte [12]. The main motivation is to attain adaptive systems that are able to accomplish difficult tasks, possibly involving real-time behavior in a complex, dynamic environment. This approach has a special interest since it greatly enhances the autonomy of the circuit, allowing the EHW to adapt to a changing environment during its lifetime. The major aspect missing, compared with biological evolution, concerns the fact that evolution is not open ended, i.e., there is a predefined goal and no dynamic environment to speak of. In this category we find two subdivisions: *centralized* and *population-oriented*.

  The main characteristic of the *centralized* approach is the existence of a single evolvable circuit and a single evolvable algorithm computation. The centralized approach implements an on-chip genetic machine. This genetic machine can be implemented in the form of a hardwired EA or an on-chip processor running the EA [52].

  A hardware implementation of the full population, and not only of one individual (as was the case for previous categories), is the distinctive feature of the *population-oriented* approach. An example of this approach is the work of Goeke et al. [11], where an evolving cellular system was implemented in which evolution takes place completely on-chip, and each cell comprises a genetic machine that updates its own genetic description.

- **Open-ended evolution**, the last subdivision situated at the top of the phylogenetic axis, involves a population of hardware entities evolving in an open-ended environment. When the fitness criterion is imposed by the user in accordance with the task to be solved (currently the rule with artificial evolution techniques), one attains a form of guided, or directed, evolution. This is to be contrasted with open-ended evolution occurring in nature, which admits no externally imposed fitness criterion, but rather an implicit, emergent, dynamical one (that could arguably be summed up as reproducibility). Open-ended undirected evolution is the only form of evolution known to produce such devices as eyes, wings, and nervous systems and to give rise to the formation of species.

The *u*bichip must provide thus the capabilities for performing each of the aforementioned levels of evolution. The capability of evolving at these four levels will allow our ubidules to evolve, in a completely autonomous way, under a real-time interaction with the environment and under the presence of uncertainty.

3.2 Ontogeny

As described in subsection 2.1.3, ontogenetic features correspond to the way an organism develops from a single cell to an entire body (self-replication), as well as to the capability of self-repair. Both processes of self-replication and self-repair require cellular replication and differentiation. Although differentiation can act at system level, to simply express a particular functionality depending on some factors, replication requires specific hardware mechanisms.

3.2.1 Replication

In order to implement cellular growth or self-repair, some parts of the circuit must be replicated. It could be an entire cell, or part of a cell, and for both cases there must be a unit responsible for the control of the replication. Previous work on replicating circuits have been done with the POEtic chip [3, 36]. However, in POEtic, replication was possible with two limitations:

1. A configuration path had to be pre-configured in the receiving molecules. This path determined the cellular morphology.
2. Two reconfiguration units had to be loaded in the circuit by the microprocessor, one for the replicating cell, and one for the creation of the new cell.

Although being quite useful for growth and self-repair, simple replication is limited in the sense that the circuit has to be preprogrammed to accept this replication. A full replication, without any requirements will therefore be a plus in our new PERPLEXUS device.

3.2.2 Self-replication

The concept of self-replication, in the case of a reconfigurable circuit, is illustrated by the following example:

We have to configure a section of the circuit that we will call a cell. Then, based on ontogenetic processes, this cell will self-replicate, by creating a real copy of itself somewhere else on the reconfigurable array. The main advantage of self-replication over replication is that there is no need to prepare the remaining part of the reconfigurable array, and that it would be possible to create a Von Neumann universal constructor [53]. The difference with the realization of Von Neumann is that instead of using a tape describing the constructor, the replication is performed through self-inspection of the cell.

In the Von Neumann cellular automata, a universal constructor is composed of a functional part and a tape containing its description. The replication process acts in two steps: (1) the functional part creates a copy of itself by using the tape information, and (2) the cell duplicates the tape and inserts it at the same relative place to the new functional part.

The concept of self-inspection acts in a different way. Instead of using a tape describing the cell, the replication directly *inspects* the content of the cell. In

the case of reconfigurable circuits, this content corresponds to the configuration bits of the reprogrammable elements. The advantage of this approach is that there is no need to duplicate information, because the cell content is directly scanned. However this gain in term of data storage leads to a more complex hardware, capable of supporting this inspection.

Considering the limitation of simple replication, the architecture of the PERPLEXUS chip will allow to perform real self-replication. This self-replication process can be viewed at three different abstraction levels:

1. At organism level, a cell simply creates a copy of itself.
2. At cellular level, cellular structure is decomposed in three organelles, mandatory to further analyse the requirements of a self-replication by self-inspection.
3. Finally, at molecular level, special hardware mechanisms are needed to allow the implementation of real self-replication.

The self-replication at organism level corresponds to the high-level vision of the self-replication - i.e. the final goal of our mechanism - and does not require more description. The molecular level will be treated in detail in section 4.3, and we here propose a description of self-replication at cellular level.

3.2.3 Self-replication at cellular level

In a general way, self-replication implementations can be considered at different levels of abstraction. It is not easy to define a level where everyone is satisfied. A good example of this is the case of self-replicating robots, where a robot must build its exact copy after providing him the necessary building blocks. The highest level can be considered as a robot able to assemble two blocks: *a battery* and *an unpowered robot*. The result from this assemblage will be a functional robot. On the other extreme we find the lowest level case: the unrealistic scenario where a robot builds its exact copy by assembling atoms from scratch.

In the world of configurable digital systems one can also envision two extreme cases. The highest level can be an FPGA able to configure another FPGA with the same configuration bitstream. And the lowest level can be a logic cell able to fully configure its neighbor logic cells with its own configuration, and still provide a differentiation mechanism allowing it to perform any useful computation. Although the lowest level approach could lead to an easy use of such mechanisms by an end user, it would impose an impressive hardware overhead.

In our case, we consider an intermediate level where a molecule contains the basic functionalities required for the self-inspection process, letting the control to be executed by other molecules not affected by this self-inspection. This approach implies also a less important hardware overhead. Its basic problem, if there is no duplicated information in the cell, is that it is not possible to replicate itself while managing its own replication. A subdivision of the cell is therefore needed to allow decomposition of the replication process.

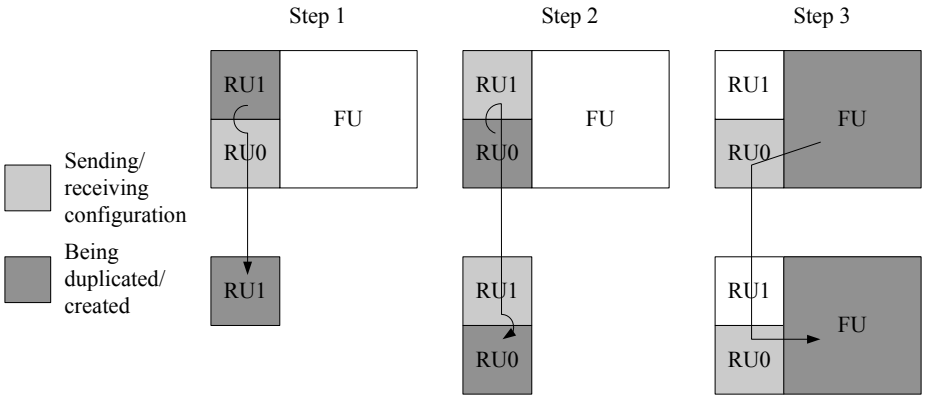

Figure 3.1: Self-replication process divided in three phases.

We split the cell into three organelles, as shown on figure 3.1: a functional unit (FU), and two replication units (RU0 and RU1) responsible for the self-replication process.

The self-replication algorithm is decomposed into three steps:

1. RU0 creates a copy of RU1 somewhere on the reconfigurable array. The choice of the place where to put RU1 is not considered in this paper, as it is closely related to the dynamic routing algorithm that will be included in a further work.
2. RU1 creates a copy of RU0, by connecting to the newly created RU1.
3. RU0 creates a copy of FU, by connecting to the newly created RU0.

This algorithm, while being quite simple, requires special hardware support (for instance, in the POEtic chip, it was not possible to implement it):

1. Connections have to be created at runtime, letting the old cell connect to the new one, to send the configuration bitstreams. This mechanism could correspond to the dynamic routing.
2. The programmable elements have to allow for a self-inspection process to retrieve all the configuration bits, and to allow the creation of the configuration path (the way the new cell is created).

3.3 Epigenesis

From an implementation point of view, an artificial neural network (ANN) can be seen as a system that maps a function from an input vector to an output vector. It consists of a set of simple units called artificial neurons, where each neuron has an internal state which depends on its own input vector. From this

state the neuron maps an output that is sent to other units through parallel connections. Each connection has a synaptic weight that multiplies the signal travelling through it. So, the final output of the network is a function of the inputs and the synaptic weights of the ANN.

Most neural models are conceived for being implemented in software platforms, making them unsuitable for hardware implementations. Most of these models, such as perceptrons or radial basis functions, encode information by using continuous values, which are processed using logistic or gaussian functions as activation function [13]. They don't take care about data resolution, about floating point operations overhead, or about the number of multiplications to be executed, since the overall overhead in software implementations due to these facts is negligible or nonexistent.

However, these aspects turn out to be very expensive when one considers their implementation as a hardware architecture. There are two main critical levels to be considered when proposing a model: *node complexity* and *inter-neural connectivity*. *Node complexity* is directly related to the arithmetic operations to be computed in the node: a simple exponential function or a multiplication can require a prohibitive amount of logic resources. In addition to this, the fact of using a floating point representation instead of fixed point can triplicate the amount of logic resources. The *inter-neural connectivity* constitutes also a critical aspect in terms of routing resources. The fact of coding information as a continuous value of n-bits requires a bus of size n in order to connect every two neurons. Such implementation requires an important amount of routing resources, which are indeed the most valuable resource in commercial FPGAs.

Some previous works have focused on optimizing the implementation of such types of models, and they end up being a coarse approximation to the original model. Other works have focused on proposing original models that exploit better the hardware specificity of the implementation. A good example to take inspiration is that of biological neurons, where information is coded in spikes instead of continuous values. Spiking neuron modelling, which is one of the three applications proposed in the PERPLEXUS project, provide a good framework for these implementations. Spiking neurons allow resource-efficient implementations at both levels: *node complexity* and *inter-neural connectivity*. At *node complexity* level, they allow simplistic hardware implementations [34, 50, 51]. At *inter-neural connectivity* level, the simple fact of coding a spike as an event allows to interconnect neurons through a single line instead of a bus.

Pulse stream neural models can be considered as a superset of spiking neurons. These are quasi-periodic streaming signals, where information is coded in timing instead of amplitude [33]. Several types of coding are allowed, according to the type of modulation used to represent data. Among the most common types of modulation one can find *pulse frequency modulation*, *pulse width modulation*, *stochastic pulse modulation*, and *pulse code modulation*.

Pulse stream models feature the same advantages exhibited by spiking neurons concerning the resource-efficiency given *node complexity* and *inter-neural connectivity* [14, 15, 16, 23]. Additionally, unlike spiking neural models, these models support the application of a vast number of supervised, reinforcement,

and unsupervised learning algorithms. The enhanced adaptability provided by learning, along with the low-resource requirements of the implementation, make pulse stream neural models a good candidate for being used in PERPLEXUS applications. More specifically in the collective robotics and culture dissemination applications.

It would be desirable thus for our *u*bichip to provide the resources for implementing the aforementioned neural models in an efficient way. This efficiency will be provided by a neural-friendly architecture, that will allow the implementation of important amounts of neural units in a single *u*bichip.

Chapter 4

Hardware mechanisms

4.1 Dynamic placement

One of the major limiting factors for the development of actual self-adaptive hardware is given by the lack of programmable architectures endowed with self-configuration abilities. Even if the dynamic reconfiguration capabilities present in current programmable devices allow for a partial on-line modification of the system functionality, they are still far from permitting to redefine in real time the system structure. This is due to the fact that the processes that determine the physical location (placement) and the physical connectivity (routing) of the elementary building blocks that constitute a system are mostly based on complex and time consuming compilation processes that take place off-chip.

The dynamic routing features present in the POEtic chip offer a partial solution for achieving actual self-configurable hardware, since the cells that constitute the system may establish their connections autonomously and in real time without the need for an external compiler. However, the placement of the cells within the physical substrate has still to be defined externally.

A new on-chip autonomous dynamic placement mechanism is being developed within the framework of the AETHER project [1]. This mechanism is being investigated as one of the possible features to be included in future hardware substrates able to support actual self-adaptive hardware.

The dynamic placement algorithm has been conceived for a homogeneous architecture constituted by a regular array of elementary programmable cells. These cells provide some basic programmable functionality and contain communication resources that permit to establish both dedicated and programmable paths between them.

The dynamic placement method is totally distributed, since it is based on local interactions between the elementary cells. A few global communication resources (basically a shared bus) are included in order to facilitate some of the processes on which it is based. The goal of this placement method is to

automatically determine the best position in the array for the cells that define a given functionality. In determining the position of a cell within the array two major metrics are considered: the distance to the already placed cells with which the cell has to connect (in order to minimize the overall delay of the system) and the utilization of the communication resources at a given position (so as to minimize the possibility of congestion problems during the routing process). Once a cell has been placed in the array it is connected to the cells that have been previously placed using the dynamic routing mechanism presented in [30]. Therefore, the combination of dynamic self-placement and dynamic self-routing features in the same substrate will permit the autonomous construction of a given functionality without the need for an external compiler.

Additionally, since both the dynamic placement and routing processes rely on dedicated resources that do not affect the functional part of the system, they may constitute the basis for supporting some of the bio-inspired features (growth, self-replication and self-repair) to be included in the Ubichip.

The AETHER project is only concerned with the development and test of the dynamic placement method and the proposal of a suitable hardware architecture able to support it. Within the framework of the PERPLEXUS project, this dynamic placement method will be carefully assessed from a physical implementation point of view, and the feasibility of developing the necessary hardware resources for its integration in the final Ubichip will be evaluated.

4.2 Dynamic routing

As previously presented, ontogenetic processes require the ability of creating paths at runtime, in order to connect newly created cells. Epigenetic systems such as growing neural networks would also need to build connectivity during the lifetime of the artificial network. Therefore the Ubichip has to propose a hardware mechanism to handle this kind of *dynamic* routing.

For this intra-chip communication, two kinds of approaches could be exploited:

1. Packet switching or wormhole routing
2. Pseudo-dynamic routing

Packet switching or wormhole routing [31] are based on targets identified by a unique identifier. Each routing node is connected to its nearest neighbors, and owns a mechanism to identify where to send a message forward. It can be a look-up table with the directions corresponding to each identifier, or simply a system where the identifier is the coordinate of the target. In the first case, a significant amount of storage elements are required to store the look-up table, while the second is more simple in term of logic elements. If we consider the hardware implementation of one of this routing mechanism, the hardware overhead seems to be unacceptable, regarding the size of the final Ubichip. This is also due to the size of the reprogrammable elements of the Ubichip. If each one

is a full processor, then the ratio between the size of a routing node and the reprogrammable element stays acceptable. However, the Ubichip elements will be really smaller than a processor, and so the ratio would not allow to implement a reasonable number of elements.

Pseudo-dynamic routing is therefore a good alternative to packet switching or wormhole routing. Instead of dynamically routing the information from a source to a target everytime a message has to be sent, a path between a source and a target is built once, and exploited by each message. It is dynamic in the sense that the paths are built at runtime, and pseudo-dynamic because the messages always follow the same route, after the creation of a path. This approach has two main advantages:

1. First, it requires less logic than packet switching or wormhole routing.
2. Second, the routing process of path building is executed once for each path. After that, sending a message only requires to send the data, as no header is needed.

The hardware implementation of dynamic routing will be based on multiplexers passing information to the neighbors of a routing node. Following a previous study of the neighborhood [44], we can argue that a 8-neighborhood is a better candidate than a 4-neighborhood in terms of hardware per routing capabilities (congestion troubles/number of transistors).

Finally, for ontogenetic processes such as the one described in section 3.2, another routing feature is mandatory. A cell has to be capable of building a copy of itself somewhere else in the circuit, and the abovementioned mechanism only allows to connect two existing cells. Two options can be envisaged:

1. Possibility to create a path in a given direction, that is to indicate only this direction, and to connect to the first empty reprogrammable element.
2. Possibility to indicate the coordinates of the target. By using serial arithmetic it would be possible to find the shortest path between the source and the destination.

Both options can solve the problem of cellular self-replication. However, the first one, although being easier to implement, needs to be sure that the path is available in order not to erase an existing path, and so the second option is more flexible, considering the potential congestion of the network.

4.3 Self-reconfiguration

The process of self-replication, as presented in section 3.2, requires specialized configurability mechanisms on the replicator and replica side. The replicator cell needs to inspect itself to retrieve its genome, while the replica needs to create itself. We start with the proposal of a creation mechanism, and we sketch some ideas about the self-inspection mechanism.

For clarifying terms we will line-out some analogies between the reconfigurable logic and the biologically-inspired terminology (Figure 4.1). We consider a *molecule* to be the equivalent of a *logic cell*, being part of a *programable logic cell array*. Analogous to biology an *organelle* will be considered as a set of molecules, where each organelle has a specific task in the *cell*. Finally, a *cell* (very different from a *logic cell*) will be considered as a set of organelles performing a certain computation. In our case, two main types of organelles will be considered within the cell: a replication organelle and a computation organelle. A set of cells will form a complete organism, which is equivalent to the configuration of the whole device or several of them.

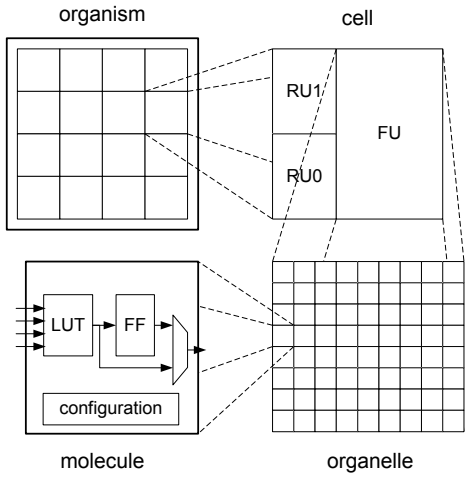

Figure 4.1: Hierarchical decomposition of an organism.

A cell's functionality is defined by a genome, which describes the complete organism. This genome provides the genetic material that will be further expressed by the organism in the form of a phenotype as a function computation. In terms of reconfigurable hardware this genome is equivalent to the logic cells' configuration bit-string.

The Ubichip will logically allow for cellular development and self-repair. The idea behind these two concepts is to let the reconfigurable part of the chip to self-organize, and to allow it to potentially support fault-tolerance mechanisms. The developmental features of a cellular organism basically require two processes: growth and differentiation, which interact during the organism construction.

The developmental process, derived from nature, allows to fully identify the requirements for our reconfigurable circuit. Initially, a single cell is programmed in the circuit. This is done by an external agent, that could be a microprocessor capable of configuring the programmable elements. This single cell can then self-replicate to start the construction of an organism. Keeping in mind the concept of a genome present in every cell, this genome can contain the number of cells of the organism, and so the self-replication can manage to end up with

the correct number of cells. After the creation of the complete organism, or when at least two cells ar present in the circuit, a differentiation process is mandatory to let the cells express a different functionality depending on, for instance, their place in the organism (An example of differentiation based on unique identifiers is described in [35]). When all cells have been created and differentiated, the artificial organism is ready to operate, and can be considered as a fully functional system.

The configuration mechanism used by this developmental process can be exploited by self-repair. A genome in every cell means that a faulty cell could be replaced by another one, simply by creating a new cell and by differentiate it.

The reconfigurable array is intrinsically a distributed system, each molecule functioning in parallel with the others. In a standard device, an external agent is responsible to load a configuration bit-string describing the entire circuit at startup. The idea behind ontogenetic hardware is to only configure a part of the organism, a cell, letting this artificial organism to grow on the electronic substrate. So, the reprogrammable array itself has to manage the dynamic incremental configuration of the cellular array. For this purpose we borrow an idea from the Tom Thumb algorithm [26] in order to manage the morphogenetic development of cell's organelles. The idea: to create a configuration path by means of morphogenetic flags.

Basically, the path serving to define the shape of the organelle is serially configured, and the configuration bits describing the functional part of the molecules are further sent. The configuration bit-string must contain thus the flags allowing the organelle construction and the molecules' functionality.

If we consider the examples of figure 4.2, the flags required for building org_0 would be:

$$\begin{aligned} F(org_0) &= [f(m_0), f(m_1), f(m_2), f(m_3)] \\ &= [\Uparrow, \Rightarrow, \Downarrow, \dashv] \end{aligned}$$

and the flags for building org_1 would be:

$$F(org_1) = [\Uparrow, \Uparrow, \Uparrow, \Rightarrow, \Downarrow, \Downarrow, \Downarrow, \Rightarrow, \Uparrow, \Uparrow, \Uparrow, \Rightarrow, \Downarrow, \Downarrow, \Downarrow, \dashv]$$

This feature can lead to the construction of an organelle of any shape, not only rectangles. When the morphology path is built, the configuration bits of the functional part of the molecules can be sent. However, the order in which the molecular functionality configuration is sent is the inverse as that for the flags. The organelle's construction would look like the process depicted in figure 4.3, where the configuration bits of molecule 3 are the first to be sent, the ones of molecule 0 are the last ones. As this configuration mechanism can be integrated in any kind of reconfigurable circuit, the number of configuration bits of a molecule is not relevant, and we simply represent it by $c(m_i)$ for the i$^{\text{th}}$ molecule. The set of configuration bits of an organelle can therefore be described

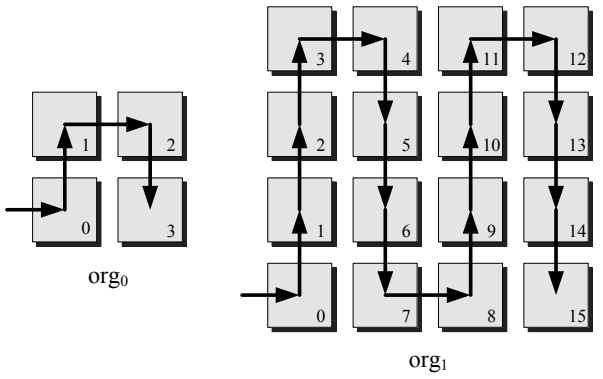

Figure 4.2: Possible paths for different organelle shapes

by a list $C(org)$:

$$C(org) = [c(m_{n-1}), c(m_{n-2}), \cdots, c(m_0)]$$

The information contained in $F(org)$ allows for the creation of the cell morphology, and the data in $C(org)$ are the configuration bits of the cell. The concatenation of these two lists completely describes the organelle. This complete description corresponds to the genome $G(org)$, and can be used for serially configuring the circuit:

$$\begin{aligned} G(org) &= [F(org), C(org)] \\ &= [f(m_0), f(m_1), \cdots, f(m_{n-1}), \\ &\quad c(m_{n-1}), c(m_{n-2}), \cdots, c(m_0)] \end{aligned}$$

As an example, figure 4.3 presents the eight steps of the creation of a 4-molecule organelle described by the genome

$$G(org) = [\Uparrow, \Rightarrow, \Downarrow, =|, c(m_3), c(m_2), c(m_1), c(m_0)]$$

The creation of an exact copy of the organelle requires the system to be able to recover the genome in the exact order that it was sent. So the result of pulling the genome must be the obtention of the same genome $G(org)$, that has been previously introduced. It would be possible to achieve this by virtually creating a shift register following the path used for the cell shape creation, and to traverse it in both directions.

This genome retrieval process provides the possibility of implementing a self-inspection mechanism. The replicators, present at the original cell, will be able to recover this string for creating a new cell in a remote set of molecules accessed through dynamically routed signals.

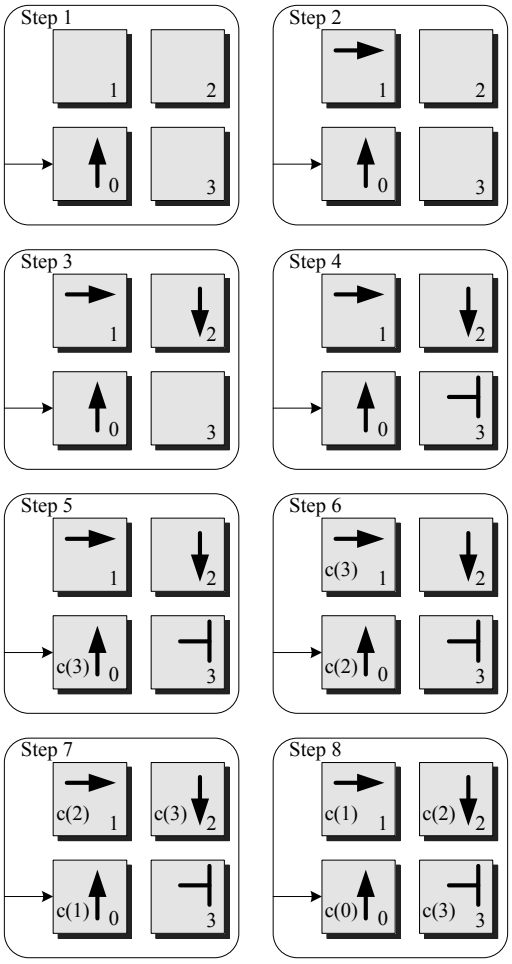

Figure 4.3: Creation of a 2×2 organelle, decomposed into 8 steps.

4.4 Neural friendliness

The ubichip must be a general purpose reconfigurable platform. However, it must be noted that, given its bio-inspired specificity and the target applications, the ubichip architecture must favor the implementation of neural architectures. This is what we call *neural friendliness*. We cannot claim that the ubichip will be able to implement systems which are impossible to do with commercial FPGAs (since one can implement any logical system on a commercial FPGA, or a set of them); but we can perform the same computation in a most efficient and flexible way.

We must propose thus a reconfigurable architecture bearing in mind, not a neuron model, but at least a family of them. Without going as far as considering the whole domain of neural networks. Consequently, we considered two families

of neural models, spiking and pulse-stream, for the reasons exhibited in 3.3. However, we are not constrained to the use of them.

If one examines the implementations of spiking and pulse stream neural models, one can find an important amount of architectural features in common.

- In both cases inter-neural connectivity is done through a single line. Unlike software oriented neural models that code inter-neural information by means of continuous values, pulse-stream and spiking models code this information in terms of pulses. This coding allows a striking simplicity when implementing flexible network topologies in hardware architectures.

  This feature is even more relevant when one considers the cost of routing resources. Even though the dynamic routing capabilities, described in 4.2, provide a huge architectural flexibility, they imply also an important silicon overhead. Making a reasonable use of such resources is thus an important issue, since it can limit the number of neurons to be included in our device. The implementation of neuron models that considers each input and output as an n-bit bus would require a prohibitive amount of routing resources, if one wants to provide topological flexibility.

- Another common aspect between spiking and pulse-stream neurons is the fact that in most cases synaptic weight multiplication is implemented in the form of an *AND* logic gate. In this way, the neural response to this input is, in some way, enabled, disabled, or weighted, by such *AND* gate.

- A final aspect which is common to both approaches is that one can identify the repeated use of additions and subtractions used to compute the internal state of the neuron. In both cases one find repeatedly counters and accumulators for computing the neuron internal state and for computing synaptic contributions. Among these models one finds both: parallel and serial arithmetic implementations.

The silicon overhead exhibited by implementations on reconfigurable devices is always important (around 40×). The use of general purpose configurable cells, based on LUTs, does not offer the most optimal configurable platform, especially when there is a set of arithmetic and logic operations that are repeatedly used. Three possible solutions are currently envisioned to tackle this issue:

1. To use a uniform logic cell array, where each cell can be used as a 4-LUT, or as two 3-LUT. When used as two 3-LUT, it should be possible to configure it as 1-bit adder and a carry, and it should be well suited also for serial implementations of both in a single logic cell. This feature would reduce to a half the amount of logic cells required to implement an adder.

2. To use a non-uniform logic cell array, where each logic cell can directly implement, for instance, an n-bits adder, a set of AND gates, or a general purpose LUT.

3. To use a uniform logic cell array, where each cell is a configurable finite state machine that can be programmed to compute a complete neuron or a part of the whole computation (i.e. a neuron state or a synapse).

Whichever solution is chosen, it will remain a general purpose architecture, and it will be also useful for implementing neural models other than the two targeted ones. Even though some features from other neurons could be not directly optimized, like sigmoid or gaussian function implementations, they will certainly still benefit from the provided features, which may still increase their implementation efficiency.

4.5 Evolvability

According to the four taxonomic levels presented in 3.1, we identify here the implications of including each of them in our *u*bichip architecture.

Extrinsic evolution does not implies any special architectural feature to be included in the reconfigurable circuit. Consequently, it will not be considered for determining the evolvability features of the platform.

Intrinsic evolution requires a number of architectural features for being supported by a reconfigurable architecture. *Unconstrained evolution* [49] constitutes the lowest level evolution to be implemented in a reconfigurable device, since it directly evolves the configuration bitstream. Though in the PERPLEXUS project the goal is not necessary to perform this unconstrained evolution, the fact of supporting it by the *u*bichip provides a more general evolutionary framework. Current commercial reconfigurable devices are not well suited for such type of implementations for two main reasons: (1) the high complexity exhibited by logic cells leads to a huge configuration search-space to be explored by the EA, and (2) the risk of internal contentions due to the routing matrices implemented with switches. For overcoming these issues, our reconfigurable circuit will remain as simple as possible for reducing the search space of possible circuit-configurations, and the routing matrices will be implemented in the form of multiplexors for avoiding hazardous configurations.

The bio-inspired chip contained in our ubidules must support the possibility of performing *complete evolution*. For guaranteeing this, the programable substrate must permit to be reconfigured by an "on-chip genetic machine". This complete evolution can be executed in a centralized manner, where a "processor" would execute the EA and would have access to a configuration port of the reconfigurable circuit. The evolution can be also done in a distributed manner, where several "genetic machines" implemented on the reconfigurable circuit will have access to the configuration bitstream of the device. This capability would be guaranteed by the mechanism described in section 4.3. In this manner a "genetic machine" will read the configuration bitstream of a circuit composed of several logic cells, and then will let an EA to modify the bitstream before writing it back to the configuration registers. In this way, the evolution will be able to modify the individual at a functional and morphogenetic level, since the bitstream modifications can also affect the organism construction.

Finally, *open-ended evolution* does not depend directly on the hardware architecture supporting it, but it is more related to the task at hand. Open-endedness is closely related to the concept of emergence, where the goal is to find unexpected behaviors. A natural application area for such systems is within the field of autonomous robots, which involves machines capable of operating in unknown environments without human intervention. Specifically, the field of collective robotics exhibits a population of individuals interacting in a common environment: they can learn to cooperate or to compete for achieving their goal, exhibiting a high level of emergence as a first step to open-endedness. Societal robotics fits also very well in this category. Societal simulations are well known because of the high degree of emergency that they exhibit. In our setup, this societal aspect will be combined with a real world interaction that may favor emergent behaviors to arise.

4.6 Scalability Issues

One of the most salient features of complex systems is the dense interaction scheme established between their constituent components. This implies that special attention has to be paid to the scalability properties of any hardware platform envisioned for the efficient implementation of complex systems. That is, the main figures of merit of the platform should be kept irrespective of the number of physical units (chips in the case of the PERPLEXUS platform) that constitute it and also irrespective of the partitioning done (i.e., the number of components that are simulated on a single physical unit).

For instance, in the case of the neural application considered within the framework of the PERPLEXUS project [19, 20] the hardware platform should be able to simulate the functionality of a spiking neural network constituted by 10000 neurons, and each neuron establishes on average 300 synaptic connections with other neurons. This means that, if 100 neurons could be physically mapped in a single chip (a number that still has to be verified), the number of I/O pins required per chip would exceed 20000, far more than can be attained with any foreseeable packaging technology.

Therefore, one of the major hardware issues to be faced by the PERPLEXUS project is related to the scalability of the basic building blocks (*u*bichips) that will constitute the PERPLEXUS platform. Among the different approaches that may provide a feasible solution for the I/O scalability problem it is foreseen to analyze and adapt the principles involved in the Address Event Representation (AER) scheme, initially proposed in [24, 39] and developed later in [5, 9, 7, 6]. This communication mechanism was developed in order to overcome the bottlenecks that appear when information has to be exchanged within a system composed of massively interconnected components. The principle of the AER scheme consists in converting an ordered sequence of events (spikes in the case of a spiking neural network) into a sequence of addresses that encode the source of the event and that are broadcasted to the rest of the system. In the receiver side, the sequence of addresses are converted again into a sequence of events

that are transferred to the corresponding destinations. Figure 4.4 represents the basic mechanism on which AER is based.

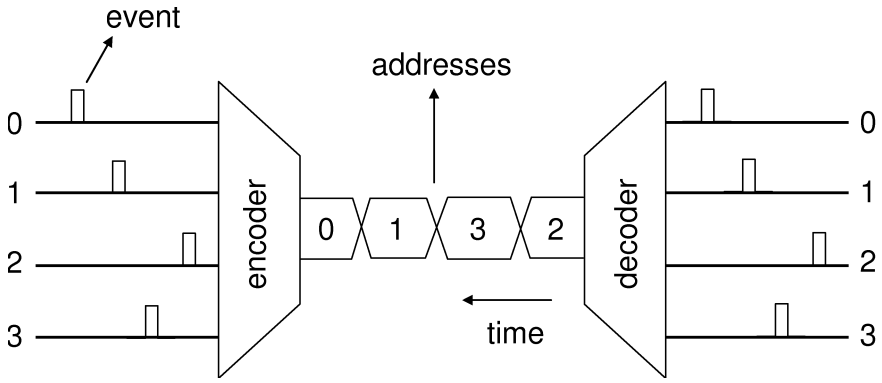

Figure 4.4: AER principle

The AER communication scheme can be easily adapted to the computational needs of a distributed system such as that constituted by the PERPLEXUS platform. A global bus containing the address of the source components that generate events at a given time is shared between all the *u*bichips. Every *u*bichip contains an encoder unit that converts the events generated by the components contained in it into addresses to be placed in the shared bus, and also a decoder unit that translates the addresses present in the shared bus into events for its implemented components. The arbitration for the access to the bus can be established in a sequential way between all the *u*bichips present in the system. In this way, every *u*bichip will indicate to the next one by means of a specific signal, *start_frame*, that it is accessing the shared bus and broadcasting the addresses corresponding to the events generated by its components. Another signal, *end_frame*, would indicate that its access to the bus has finished and that the next *u*bichip is granted to access the bus. When the last *u*bichip generates the *end_frame* signal, the first one will activate a global signal, called *frame_update*, so that all the components included in all the *u*bichips may update their outputs from the inputs received in the current simulation frame. Figure 4.5 represents the system organization for the implementation of this communication scheme. The boxes labelled as C in the figure are the components implemented in the different *u*bichips.

The scalability of the proposed principle is quite simple to establish. In the frame associated with every *u*bichip for broadcasting its own addresses, the first address would be an integer whose value is that generated by the previous *u*bichip decremented by one unit. Therefore, if the number of *u*bichips in the system is N, the first address generated by the first *u*bichip would be N-1, and the first address generated by the last one will be 0. In this way, the first *u*bichip is able to generate the overall *frame_update* signal after receiving an *end_frame* signal corresponding to a frame whose first address is 0. In the case the emulation platform consists of only one Ubidule there is no need for the

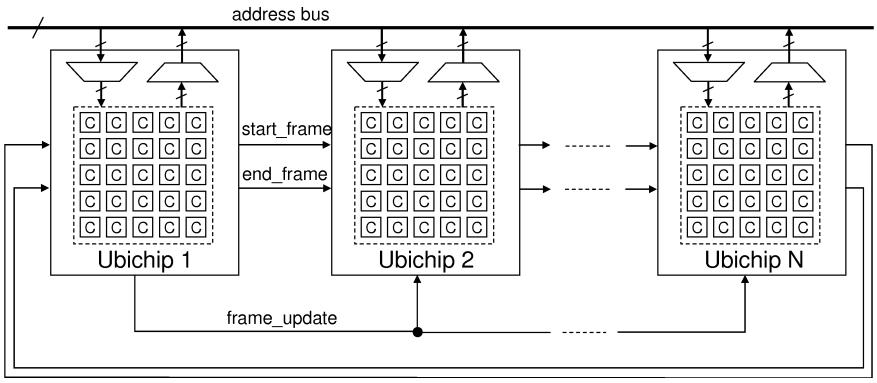

Figure 4.5: Possible AER communication implementation in the PERPLEXUS platform

*u*bichips to issue the first address in every frame, since the first *u*bichip will activate the *frame_ update* signal upon receiving the *end_ frame* signal from the last *u*bichip.

It is worth noting that, because it basically consists on a multiplexed broadcasting of information, this communication scheme is valid either for a single-*u*bichip per Ubidule scenario or for a multi-*u*bichip per Ubidule scenario. Furthermore, this communication scheme may provide enough bandwidth for the communication needs of the applications considered in the PERPLEXUS project, even in the single-*u*bichip per Ubidule scenario and a wireless physical link between Ubidules. If we consider the neural application (the most restrictive one in terms of capacity and bandwidth), and assuming 100 neurons are implemented in each *u*bichip and a 54 Mbits/second wireless link, this would permit a neuron firing rate of around 300 spikes/second, something that is in line with the simulation experiments already performed for the application [18]. In the case of a multi-*u*bichip per Ubidule scenario with a shared bus running at 10 MHz (a quite conservative approach) this would imply a firing rate of around 1000 spikes/second, far exceeding the application needs.

Finally, it is also worth noting that this communication scheme permits a local synchronous implementation of the target functionality and an asynchronous information exchange, something that fits well with the scalability features to be attained by the PERPLEXUS platform. Additionally, the proposed communication scheme permits to synchronize the overall emulation of the target system in the platform, a strict requirement in some applications like the spiking neural network that is being considered within the framework of the project.

Chapter 5

Conclusion

There is a main reason for the success of living beings perpetuation on Earth: *flexibility*. This flexibility is presented in the form of adaptation, at long and short term, and development. It is thanks to this flexibility that living beings are able to learn, evolve, reproduce, grow, and self-repair. This flexibility allows systems to self-modify in order to change different aspects as their behavior, their size, or their morphology. Unlike living beings, electronic circuits use to have fixed architectures and use to have functionalities pre-defined at design-time. Some circuits allow to be programmed in order to exhibit a certain level of behavioral adaptation. However, their adaptation capabilities remain constrained by the level of pre-programmed flexibility .

This document presented a set of architectural features that will allow a logical circuit to learn, evolve, reproduce, grow, and self-repair. The key idea borrowed from nature to do this is *flexibility*. In our case, this flexibility is provided in the form of reconfigurability. The reconfigurability features presented here (dynamic placement, dynamic routing, and distributed self-reconfigurability), along with the bio-inspired techniques facilitated by our architecture (evolvability, neural friendliness, and scalability), will allow our applications to mimic their biological analogs in a more proper way than current state of the art bio-inspired systems.

The *u*bichip will feature these configuration capabilities, becoming an outstanding platform for implementing POE systems. The phylogenetic aspect will be provided by the evolvability feature, which along with the self-reconfiguration, will allow the implementation of population-based self-reconfigurable evolving systems in a completely autonomous way. The ontogenetic axis is guaranteed with the inclusion of dynamic placement, dynamic routing, and self-reconfigurability, which will allow the implementation of systems with growing and self-repairing capabilities. And, finally, the epigenetic aspect is covered by two features: (1) the neural friendliness that will allow an efficient implementation of neural architectures, and (2) the scalability provided by the *u*bichip, which will allow the design of large networks contained in multiple *u*bichips.

Bibliography

[1] AETHER project webpage. http://www.aether-ist.org.

[2] ARM. *AMBA Specification, Rev 2.0*. Advanced RISC Machines Ltd (ARM), http://www.arm.com/armtech/AMBA_Spec, 1999.

[3] W. Barker, D. M. Halliday, Y. Thoma, E. Sanchez, G. Tempesti, J.-M. Moreno, and A. M Tyrrell. Fault tolerance using dynamic reconfiguration on the poetic tissue. 2007. To be published.

[4] T. Bäck. *Evolutionary algorithms in theory and practice : evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, New York, 1996.

[5] K.A. Boahen. Point to point connetivity between neuromorphic chips using address events. *IEEE Trans. on Circuits and Systems II*, 47(5):416–434, 2000.

[6] K.A. Boahen. A burst-mode word-serial address-event Link-I: Receiver design. *IEEE Trans. on Circuits and Systems I*, 51(7):1281–1291, 2004.

[7] K.A. Boahen. A burst-mode word-serial address-event Link-I: Transmitter design. *IEEE Trans. on Circuits and Systems I*, 51(7):1269–1280, 2004.

[8] C. H. V. Cooper, D. M. Howard, and A. M. Tyrrell. Using GAs to create a waveguide model of the oral vocal tract. In G. R. Raidl et al., editors, *Proc. Applications of Evolutionary Computing 2004*, volume 3005 of *LNCS*, pages 280–288, Berlin Heidelberg, 2004. Springer-Verlag.

[9] E. Culurciello and A.G. Andreou. A comparative study of access topologies for chip-level address-event communication channels. *IEEE Trans. on Neural Networks*, 14(5):1266–1277, 2003.

[10] D. B. Fogel. *Evolutionary computation : toward a new philosophy of machine intelligence*. IEEE Press, New York, 2nd edition, 2000.

[11] M. Goeke, M. Sipper, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini. Online autonomous evolware. In *Evolvable Systems: From Biology to Hardware, LNCS*, volume 1259, pages 96–106. Springer-Verlag, 1997.

[12] P. Haddow and G. Tufte. Evolving a robot controller in hardware. *Proceedings of the Norwegian Computer Science Conference (NIK99)*, pages 141–150, 1999.

[13] S. Haykin. *Neural Networks, A Comprehensive Foundation*. Prentice-Hall, Inc, New Jersey, 2 edition, 1999.

[14] H. Hikawa. Frequency-based multilayer neural network with on-chip learning andenhanced neuron characteristics. *IEEE Transactions on Neural Networks*, 10(3):545–553, 1999.

[15] H. Hikawa. A digital hardware pulse-mode neuron with piecewise linear activation function. *IEEE Transactions on Neural Networks*, 14(5):1028–1037, 2003.

[16] H. Hikawa. A new digital pulse-mode neuron with adjustable activation function. *IEEE Transactions on Neural Networks*, 14(1):236– 242, 2003.

[17] G. Hollingworth, S. Smith, and A. Tyrrell. The intrinsic evolution of Virtex devices through internet reconfigurable logic. In *Evolvable Systems: From Biology to Hardware, LNCS*, volume 1801, pages 72–79. Springer-Verlag, 2000.

[18] J. Iglesias. *Emergence of Oriented Circuits driven by Synaptic Pruning associated with Spike-Timing-Dependent Plasticity (STDP)*. Phd thesis, University Grenoble I Joseph Fourier, University of Lausanne, 2005.

[19] J. Iglesias, J. Eriksson, F. Grize, and M. Tomassiniand A.E.P. Villa. Dynamics of pruning in simulated large-scale spiking neural networks. *Biosystems*, 79(1-3):11–20, 2005.

[20] J. Iglesias, J. Eriksson, B. Pardo, M. Tomassini, and A.E.P. Villa. Emergence of oriented cell assemblies with spike-timing-dependent plasticity. In W. Duch, J. Kacprzyk, E. Oja, and S. Zadrozny, editors, *Artificial Neural Networks: Biological Inspirations, LNCS*, volume 3693, pages 11–20. Springer-Verlag, 2005.

[21] J. R. Koza. *Genetic programming : on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, Cambridge, Mass., 1992.

[22] C. G. Langton. *Artificial life : an overview*. Complex adaptive systems. MIT Press, Cambridge, Mass., 1995.

[23] P. Lysaght, J. Stockwood, J. Law, and D. Girma. Artificial neural network implementation on a fine-grained FPGA. In R. Hartenstein and

M. Z. Servit, editors, *Field-Programmable Logic: Architectures, Synthesis and Applications. 4th International Workshop on Field-Programmable Logic and Applications*, pages 421–431, Prague, Czech Republic, 1994. Springer-Verlag.

[24] M.A. Mahowald. *VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function*. Phd thesis, California Institute of Technology, Pasadena, 1992.

[25] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The embryonics approach. *Proceedings of the IEEE*, 88(4):516–540, April 2000.

[26] D. Mange, A. Stauffer, E. Petraglio, and G. Tempesti. Self-replicating loop with universal construction. *Physica D*, 191(1-2):178–192, apr 2004.

[27] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, Mass., 1996.

[28] J.-M. Moreno, Y. Thoma, E. Sanchez, J. Eriksson, J. Iglesias, and A. Villa. The poetic electronic tissue and its role in the emulation of large-scale biologically inspired spiking neural networks models. In *ComPlexus*, volume 3, pages 32–47, aug 2006.

[29] J.-M. Moreno, Y. Thoma, E. Sanchez, O. Torres, and G. Tempesti. Hardware realization of a bio-inspired POEtic tissue. In R. S. Zebulum, D. Galtney, G. Hornby, D. Keymeulen, J. Lohn, and A. Stoica, editors, *Proc. 2004 NASA/DoD Conference on Evolvable Hardware*, pages 237–244, Los Alamitos, California, 2004. IEEE Computer Society.

[30] J. M. Moreno Arostegui, E. Sanchez, and J. Cabestany. An in-system routing strategy for evolvable hardware programmable platforms. In *Proc. 3rd NASA/DoD Workshop on Evolvable Hardware*, pages 157–166. IEEE Computer Society Press, 2001.

[31] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[32] J. Pinter. *Global optimization in action (Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications)*. Kluwer Academic Publishers, Dordrecht / Boston / London, 1996.

[33] L.M. Reyneri. A performance analysis of pulse stream neural and fuzzy computingsystems. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 42(10):642–660, 1995.

[34] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano. Hardware spiking neural network with run-time reconfigurable connectivity. In *5th NASA / DoD Workshop on Evolvable Hardware (EH 2003)*, pages 199–208. IEEE Computer Society, 2003.

[35] D. Roggen, Y. Thoma, and E. Sanchez. An evolving and developing cellular electronic circuit. In J. Pollack et al., editors, *Proc. Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pages 33–38, Cambridge, Massachusetts, USA, 2004. The MIT Press.

[36] J. Rossier, Y. Thoma, P.-A. Mudry, and G. Tempesti. MOVE processors that self-replicate and differentiate. In A.J. Ijspeert et al., editors, *Proc. Biologically Inspired Approaches to Advanced Information Technology (BioADIT 2006)*, number 3853 in LNCS, pages 160–175, Berlin Heidelberg, 2006. Springer-Verlag.

[37] 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 *Evolvable Systems: From Biology to Hardware, LNCS*, volume 1259, pages 35–54. Springer-Verlag, 1997.

[38] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, and A. Stauffer. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, 1997.

[39] M. Sivilotti. *Wiring Considerations in Analog VLSI Systems With Applications to Field Programmable Networks*. Phd thesis, California Institute of Technology, Passadena, 1991.

[40] G. Tempesti, D. Mange, A. Stauffer, and C. Teuscher. The BioWall: An electronic tissue for prototyping bio-inspired systems. In *Proc. of the 2002 NASA/DOD Conference on Evolvable Hardware*, pages 221–230, 2002.

[41] Y. Thoma. *Tissu Numérique Cellulaire à Routage et Configuration Dynamiques*. Phd thesis, EPFL, 2005.

[42] Y. Thoma and E. Sanchez. A reconfigurable chip for evolvable hardware. *Proc. Genetic and Evolutionary Computation Conference (GECCO 2004)*, 1:816–827, 2004.

[43] Y. Thoma and E. Sanchez. A reconfigurable chip for evolvable hardware. In K. Deb et al., editors, *Proc. Genetic and Evolutionary Computation COnference (GECCO 2004), Part I*, number 3102 in LNCS, pages 816–827, Berlin, Heidelberg, 2004. Springer Verlag.

[44] Y. Thoma and E. Sanchez. An adaptive FPGA and its distributed routing. In *Proc. ReCoSoc '05 Reconfigurable Communication-centric SoC*, pages 43–51, Montpellier - France, June 2005.

[45] Y. Thoma, E. Sanchez, J. M. M. Arostegui, and G. Tempesti. A dynamic routing algorithm for a bio-inspired reconfigurable circuit. In *Field-Programmable Logic and Applications, LNCS*, volume 2778, pages 681–690. Springer-Verlag, 2003.

[46] Y. Thoma, E. Sanchez, J.-M. Moreno Arostegui, and G. Tempesti. A dynamic routing algorithm for a bio-inspired reconfigurable circuit. In P. Y. K. Cheung et al., editors, *Proc. of the 13th International Conference on Field Programmable Logic and Applications (FPL'03)*, volume 2778 of *LNCS*, pages 681–690, Berlin, Heidelberg, 2003. Springer Verlag.

[47] Y. Thoma, G. Tempesti, E. Sanchez, and J. M. M. Arostegui. POEtic: an electronic tissue for bio-inspired cellular applications. *Biosystems*, 76(1-3):191–200, 2004.

[48] A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *Evolvable Systems: From Biology to Hardware, LNCS*, volume 1259, pages 390–405. Springer-Verlag, 1997.

[49] A. Thompson, I. Harvey, and P. Husbands. Unconstrained evolution and hard consequences. *Towards Evolvable Hardware, The Evolutionary Engineering Approach, LNCS*, 1062:136–165, 1996.

[50] O. Torres, J. Eriksson, J. M. Moreno, and A. Villa. Hardware optimization of a novel spiking neuron model for the POEtic tissue. In *Artificial Neural Nets Problem Solving Methods, Pt II*, volume 2687, pages 113–120. Springer-Verlag, 2003.

[51] A. Upegui, C. A. Peña-Reyes, and E. Sanchez. An FPGA platform for on-line topology exploration of spiking neural networks. *Microprocessors and Microsystems*, 29(5):211–223, 2005.

[52] A. Upegui and E. Sanchez. Evolving hardware with self-reconfigurable connectivity in Xilinx FPGAs. In *Proceedings of the 1st NASA /ESA Conference on Adaptive Hardware and Systems(AHS-2006)*, pages 153–160, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[53] J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Illinois, 1966. Edited and completed by A. W. Burks.