Published in International Journal of Knowledge-based and Intelligent Engineering Systems, 2008, vol. 12, no. 3, pp- 201-212, which should be cited to refer to this work.

DOI: https://doi.org/10.3233/KES-2008-12303

The PERPLEXUS Bio-Inspired Hardware Platform: A Flexible and Modular Approach

Andres Upegui, Yann Thoma, Eduardo Sanchez, Andres Perez-Uribe
HEIG-VD, Yverdon-les-Bains, Switzerland
(andres.upegui, yann.thoma, eduardo.sanchez, andres.perez-uribe)@heig-vd.ch

Juan Manuel Moreno, Jordi Madrenas
UPC, Barcelona, Spain
(moreno, madrenas)@eel.upc.edu

Gilles Sassatelli
CNRS, France
sassatelli@lirmm.fr

Abstract

This paper introduces the Perplexus hardware platform, a scalable computing substrate made of custom reconfigurable devices endowed with bio-inspired capabilities. This platform 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. The modularity and flexibility of the platform are the key for tackling the diverse hardware setup needs of the different applications, mainly in the form of a pervasive distributed computing platform and a bio-inspired chip architecture. The Perplexus platform will provide a novel modeling framework thanks to the pervasive nature of the hardware platform, its bio-inspired capabilities, its strong interaction with the environment, and its dynamic topology.

The final infrastructure will be used as a simulation tool for three applications: neurobiological modeling, culture dissemination modeling, and cooperative collective robotics.

1. Introduction

Perplexus is a project funded by the European Commission through the 6th framework programme and regroups eight research institutions from four different countries. PERPLEXUS runs for three years, starting from September 1st, 2006.

The Perplexus project aims to develop a scalable hardware platform composed of a set of custom reconfigurable devices. These devices will be endowed with bio-inspired capabilities that will enable the simulation of large-scale complex systems [3]. They will also allow the study of emergent complex behaviors in virtually unbounded wireless networks 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*) [20] 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 [3]. 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 larger entity.

We have identified three domains where our modeling infrastructure will prove its usefulness as a powerful and innovative simulation tool: neurobiological modeling, culture dissemination modeling, and cooperative collective robotics. We will perform qualitative and quantitative comparisons between classical implementations of these three modeling applications and their implementation running on a network of *ubidules*. For doing so, we envision three strategic aspects to be addressed:

(1) Design and development of the hardware platform. It will consist of a set of *ubidules* that will support the simulation of our complex systems.

(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.

(3) Study of the emergent phenomena arising from the strong interaction between our ubiquitous computing modules and the real environment.

These three aspects constitute an important contribution to fields as diverse as pervasive systems, complex systems, distributed computing, culture modeling, collective robotics, and bio-inspired hardware systems, among others. For tackling these challenges we have fixed a set of objectives toward which our efforts will be focused. These objectives, which are explained in section 2, cover technological and scientific aspects. Section 3 introduces the *ubidule*: the hardware platform that will support our modeling framework. Then, section 4 describes the *ubichip*: the bio-inspired chip being the key component of the hardware platform. And, finally, section 5 summarizes the current state of the project and the contributions presented here.

2. Perplexus Objectives

The simulation of large-scale complex systems requires huge amounts of computing resources. Even if the Moore law yet guarantees increasingly more powerful chips every year, the capacity of a single device is not always able to provide the optimal solution for running this kind of application. Supercomputing, grid computing and distributed computing are possible solutions to this problem. However, a new kind of distributed computing could appear in the near future, with the advent of a new era of computing, after the mainframe era and the PC era. In this 3rd era of computing, we are likely to see a myriad of ubiquitous devices with large computing and sensory capabilities in our environment. Such devices will interact with each other using wireless communication and coordinate to seamlessly help people in their daily tasks. Anticipating this new era of computing, we can imagine a solution for simulating complex systems, inspired from the distributed computing projects, but this time using the computation, communication, and sensing resources of ubiquitous devices in our environment: the alarm clock radio on our night table, our children's electronic toys and robots, our microwave oven, our cellular phones, PDAs, etc.

In this project we will develop a ”scale model” framework for ubiquitous computing. We will implement a series of models that will take advantage of the modeling framework provided by the Perplexus platform, and we will study the emergent behaviors arising from such models.

The design and development of the Perplexus platform rises challenging technological issues. We will develop a scalable network of *ubidules* equipped with wireless communication capabilities and rich sensory elements. The platform will be modular for allowing application developers to customize their platform set-up. In this way, application developers can easily build their system by selecting what to plug to the *ubidule* from a set of peripherals. These peripherals can be a wireless communication interface (Wi-Fi or bluetooth), sensors, actuators, cameras, or flash memories. A more complete description is given in section 3.

The *ubidule* will also be equipped with a set of *ubichips* [20]. The *ubichip* will feature bio-inspired reconfigurability capabilities that will support the implementation of bio-inspired hardware systems. Reconfigurability mechanisms like dynamic routing, self-replication by means of self-reconfiguration, and a simplified connectivity, will allow the implementation of bio-inspired mechanisms such as development, learning, and evolution, favoring in this way the on-line and adaptability of the hardware platform to the task at hand. A more complete description of the *ubichip* is given in section 4.

We can distinguish two approaches to deal with complex systems. On the one hand, the top-down approach which functions in a divide-and-conquer manner. A problem is divided into sub-problems that are individually considered. Such an approach has the problem of missing the fact that complex behaviors need not to have complex roots, and models are not scalable. On the other hand, bottom-up approaches do consider explanations of complex behavior via simple, local component interactions in a decentralized manner, generally providing a scalable model of the complex system. In this project, we consider challenging modeling problems following a bottom-up approach.

As far as the computer models of complex systems are concerned, we can also distinguish two approaches. On the one hand, there are continuous models based on differential equations. These models generally suppose a high abstraction level and do not integrate a spatial dimension. On the other hand, there are discrete models that introduce individual entities called ”agents” or ”particles”. Agent-based modeling is very often based on discrete ”grid-world” models. In this project we will explore the use of ”perplex-world” models, virtually unbounded discrete models realized by interconnecting many ubiquitous computing modules (*ubidules*) and by interacting with real-world sources of information. We will determine the range of applications and the advantages of such a framework for simulating complex systems in the domains of realistic neural models, the dynamic emergence of culture, and collective robotics.

Another major objective is to study the emergent phenomena arising from the strong interaction between our hardware infrastructure and the real environment. The Perplexus platform can be seen as a complex system itself: it is a population of artificial organisms interacting with the real environment. We will study emergent organization mechanisms that will enable the formation of two kinds of structures: a single distributed complex organism or a population of collaborating-competing individual organisms. For instance, a growing network setup of *ubidules* could form a ”small world” network.

A third scientific objective is the study of the emergence of an ”artificial culture”. The *ubidule* ubiquitous computing modules learn by interacting with the environment and with the user. This learning process might lead to the acquisition of ”concepts” that might be communicated to neighbor *ubidules* and diffused through the Perplexus platform. The sharing of such concepts might prosper and reach a stable state throughout the platform. Such a stable state or concept consensus might be seen as an ”artificial culture” shared by the *ubidule* modules of the Perplexus platform.

3 The Ubidule

The *ubidule* constitutes the hardware substrate that will support the Perplexus applications. It is a customizable computing unit that can be tailored to each of the target applications in a plug & play manner. An application requiring a specific wireless communication protocol, a given set of sensors and actuators, or extended memory, will be supported by the *ubidule* thanks to a key concept: modularity. In the case of the *ubidule*, this modularity will be present in the form of external hotplug peripherals that will allow the application designer to tailor the platform according to its needs. The same concept of modularity will be applied for physically interconnecting ubidules. An example is the AER bus implementation (further described in subsection 4.2.4), which permits to run larger neural simulations by interconnecting several *ubichips*.

Today, we have already implemented the first *ubidule* prototype. It is an electronic board mainly featuring a processor running Linux, a 5Mgates FPGA for the /it ubichip prototyping, and support for several peripherals. One of the major features of the *ubidule* prototype is its modularity and flexibility. It can be easily customizable for each of the target applications in order to allow a complete and efficient modeling platform.

Figure 1 depicts the schematic of the *ubidule* prototype board. A mini-PCI socket supporting a Colibri board from Toradex [18] provides the possibility of including a high-end embedded processor. The Colibri board contains either an Xscale PXA270 or PXA320, and enough memory resources for running well supported operating systems such as Linux or Windows CE. The Colibri board constitutes thus the first step toward the desired flexibility and modularity of our *ubidules*, by providing the advantages of a performant processor, a well supported operating system, gcc tools, and a large number of software resources as application programs, services, and drivers.

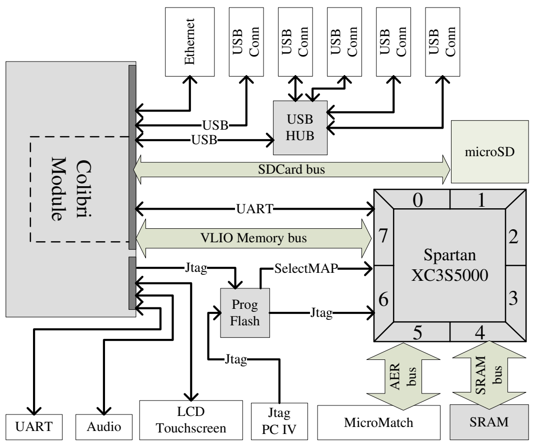

Figure 1. Schematic of the Ubidule Prototype

At hardware level, the modularity in the system is mainly provided in the form of standard peripheral interfaces. For this purpose, the Colibri board supports USB, SDcard, and CompactFLASH devices. However, given the board area and number of off-the-shelf peripherals available, the *ubidule* contains five USB ports and a microSD card reader. Additionally, two UART ports can be used in order to connect additional devices.

3.1 Peripherals

The aforementioned standard peripheral interfaces can be exploited in several ways in order to provide the communication and sensing capabilities required by the Perplexus applications.

3.1.1 Communication

The communication capabilities are a key component of ubiquitous systems in order to circumvent typical issues of distributed systems such as synchronization, data coherence, system monitoring, system upgrade, etc. The *ubidule* integrates a single way of communication through an Ethernet port included in the Colibri board. Ethernet is thus a straightforward interface for providing communication with other nodes; indeed, it is directly supported by standard operating systems.

The system modularity, in the form of USB and microSD ports, provides the mechanisms to enhance these communication capabilities in an important way. Wi-Fi, Bluetooth, and Zigbee are some of the wireless communication standards available by means of external peripherals. According to the application requirements, the *ubidule* platform can be tailored for supporting any required communication platform available in the form of USB dongle or SDcard if the respective driver is available.

3.1.2 Sensors and Actuators

Neither sensors nor actuators are directly implemented on the *ubidule* board. However, commercial USB sensors and actuators are provided by several companies such as Phidgets [9] and Toradex [18]. Most of them do not need special drivers, but just a set of libraries. An important number of sensors is available from them: force, vibration, accelerometer, RFID reader, distance, and temperature, among others, as well as some actuators such as servomotors and LCDs. Other commercial USB gadgets are not discarded like webcams, joysticks, mini-fan, mini- vacuum cleaner, etc. or even a complete robotic kit such as the LEGO Mindstorms NXT.

The *ubidule* includes also some embedded interfaces: an audio interface allows to plug input and output audio devices, and an LCD-touchscreen interface, along with a mechanical support, allows to connect the *4.3" SHARP WQVGA Touch TFT* from Toradex.

3.2 Hardware Reconfigurability

The *ubidule* board features a XC3S5000 FPGA from Xilinx, the largest device of the low-cost family Spartan-3. It is capable of implementing digital circuits of up to 5 million gates. This device offers a very reasonable trade-off of resources vs cost for the prototyping purposes of the ubidule.

At the startup of the system, the FPGA will be configured by an external Platform FLASH solution from Xilinx through the interface SelectMAP. Both, the Platform FLASH and the FPGA, can be also programmed via Jtag from a PC or from the Colibri board.

3.3 Ubichip Prototyping

The main goal of the FPGA contained in the *ubidule* board is to prototype the *ubichip* [20] that will be included in the final *ubidule* platform. The goal of implementing a prototype of the *ubichip* is to test and validate all its functionalities before fabricating the final chip. The most critical architectural features to be validated are the ones concerning the configurability of the system. These configurability issues include the possibility of configuring the *ubichip* from a parallel and a serial interface from the Colibri board, the self-reconfiguration mechanism for implementing self-replicating systems [17], and the dynamic routing mechanism.

4. The Ubichip

At the heart of the *ubidules*, we will use a custom reconfigurable electronic device capable of implementing bio-inspired mechanisms such as growth, learning, and evolution. These bio-inspired mechanisms will be possible thanks to reconfigurability mechanisms like dynamic routing, distributed self-reconfiguration, and a simplified connectivity.

Recent work in this field is the POEtic tissue [16], a reconfigurable hardware platform for rapidly prototyping bio-inspired systems that employ POE principles [13], which has been developed in the framework of the European project POEtic. The POEtic chip has been specifically designed to ease the development of bio-inspired applications.

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 will be considered for the design of our *ubichip*, in subsection 4.1 we will introduce some concepts and issues concerning bio-inspired hardware, also known as POE hardware [13], and we will also present the desired bio-inspired features to be supported by the *ubichip*. Then, in subsection 4.2, we will describe the hardware mechanisms that will allow the implementation of such bio-inspired systems.

4.1 POE Hardware

Living beings, unlike engineered systems, exhibit a high level of adaptability and robustness thanks to several biological mechanisms: reproduction, learning, self-repair, growth, and evolution. It would be thus desirable to include such mechanisms in human-designed systems in order to increase their lifetime and to improve their adaptability to the environment and to the user. If one considers life, as we know it, the following three levels of organization can be distinguished [13]: (1) Phylogeny, concerning the temporal evolution of a certain genetic material in individuals and species, (2) Ontogeny, concerning the developmental process of multicellular organisms, and (3) Epigenesis, concerning the learning process during an individual's lifetime. Analogous to nature, the space of bio-inspired hardware systems can be partitioned along these three axes, to which we refer as the POE model.

When we refer to the *phylogenetic* axis of bio-inspired hardware systems, we are talking about "Evolvable Hardware" (EHW). EHW makes use of evolutionary algorithms in order to define a description of a hardware system. From a desired behavior specification of a circuit, an evolutionary algorithm will search for a circuit configuration describing a satisfactory solution to the specification. If one examines the work carried out to date under the heading EHW, one can identify four taxonomic subdivisions according to the level of bio-inspiration: extrinsic, intrinsic, complete, and open-ended evolution [13]. The *ubichip* 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.

The *ontogenetic* axis comprises several mechanisms of high interest for inclusion in human-designed systems. Self-replication and self-repair are two key characteristics of living beings that are still far from being exploited by engineered systems with an efficiency comparable to nature. However, some key factors from multicellular beings have been identified for use in the design of ontogenetic machines: the dependence of cell's functionality upon its relative position, the relevance of the physical neighborhood for chemical interactions between cells, the importance of time scales 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 like *Embryonics* [6] (embryonic electronics) and *POEtic* [1, 11, 12] have studied the issues related to hardware implementations of such mechanisms.

The *epigenetic* axis of bio-inspired hardware systems mainly refers to hardware implementations of artificial neural networks, also known as "neural hardware". Most neural models are conceived for being implemented in software platforms, making them unsuitable for hardware implementations. These models don't take care about data resolution, floating point operations overhead, or multiplications, since their overall overhead in software is negligible or nonexistent. These aspects turn out to be very expensive when one considers their implementation as a hardware architecture. Some previous works have focused on optimizing the implementation of such types of models, and other works have focused on proposing original models that exploit better the hardware specificity of the implementation [10, 19].

The implementation of such bio-inspired features on a hardware substrate requires some special hardware mechanisms to be provided by the underlying reconfigurable architecture. These mechanisms must allow an efficient use of hardware resources when designing POE circuits.

4.2 Ubichip mechanisms

The system architecture envisioned for the *ubichip* is represented in figure 2, and it is composed of four main parts. (1) The configurable array consists in a bi-dimensional regular array of reconfigurable cells called *Macrocells*. A *macrocell* is composed of a pair of self-replication (SR) and dynamic routing (DR) units (to be further described in subsections 4.2.2 and 4.2.3), associated to four *ubicells*. A *macrocell* constitutes thus the minimal unit to be addressed (for reading or configuration) by a self-replication unit. Figure 3 depicts a top level view of a *macrocell*, which is composed of three layers: a ubicell array layer, a dynamic routing layer, and a self-reconfiguration layer. This configurable array can be further extended by interconnecting several ubichips. (2) The encoder/decoder is in charge of managing the shared address bus that implements the inter-chip communication with an address event representation (AER) scheme (to be further explained in subsection 4.2.4). (3) The memory controller takes care of handling the data RAM needed to store system parameters as well as the CAM (Content Addressable Memory) that will be used to implement the AER communication scheme between parallel applications. Finally, (4) the system manager handles the overall configuration of the *ubichip* and its interface with the main controller of the *ubidule*. An example of implementation is illustrated by the neurobiological modeling application, where a SIMD-like solution is envisioned with a centralized sequencer and a set of reconfigurable neural units. The sequencer included in this subsystem interprets the code corresponding to the execution of the neurons to be implemented in the configurable section of the *ubichip*. A small instruction set has been specifically designed for this sequencer. This instruction set is general enough to permit the implementation of any parallel system. Furthermore, it includes conditional store instructions so as to permit a linear execution of the code, thus improving the concurrence of the system.

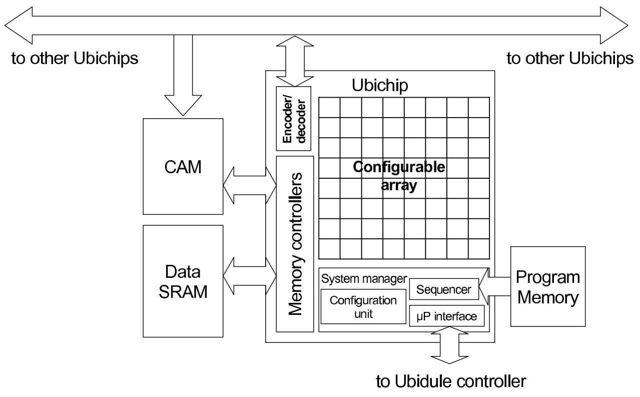

Figure 2. System Architecture of the ubichip.

Bio-inspired circuits require a number of reconfigurability and architectural features in order to be implemented in an efficient way. In this section we describe the hardware mechanisms that will allow the *ubichip* to implement circuits able to adapt by means of evolution, development, and learning.

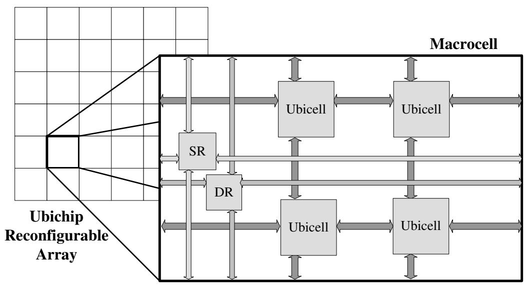

Figure 3. Composition of a Macrocell

4.2.1 Neural-friendly Architecture

Among the applications being targeted in Perplexus, the neurobiological modeling application imposes the more critical computing power requirements. In the specific neural application considered within the framework of Perplexus [5], the hardware platform should be able to simulate the functionality of a spiking neural network constituted by 10000 neurons, where each neuron establishes on average 300 synaptic connections with other neurons.

A careful analysis has been carried out in order to define the internal organization of the reconfigurable cells, in order to allow an efficient implementation of such neural network. Taking into account the trade-off to be achieved between the integration density and the granularity of the basic cells, it has been decided to define their structure around four 4-LUT units that can be configured as four independent 4-input functions or as a 4-bit ALU. This 4-bit ALU will allow the implementation of the neural processing units that will be controlled by the sequencer. The memory elements required to implement the LUT units will be designed so that they can also be used as an 8 x 4-bit register file for the ALU, so as to optimize the implementation of the arithmetic and logic instructions. The architecture of a basic cell in 4-bit ALU mode is depicted in figure 4.

The neural-friendly architecture is provided thus in the form of a reconfigurable unit array that can be configured as an efficient neural SIMD multiprocessor. A single 4-LUT logic unit can act as a 4-bit SIMD processing element (PE), or it can be assembled to neighbor units to form an n-bit PE (being n a multiple of 4). A neural-oriented instruction set, specifically defined for these PEs, guarantees an optimal implementation of spiking neural systems, allowing a single processing element to compute a plurality of neurons and synapses.

4.2.2 Self-reconfiguration

For cellular systems, ontogenetic features correspond to the way an organism develops from a single cell to an entire organism (self-replication), as well as to the capability of self-

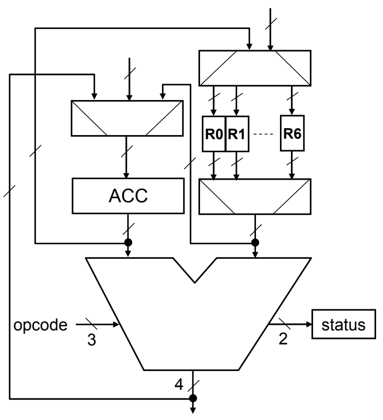

Figure 4. Configurable cell in 4-bit ALU mode.

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, self-replication requires specific hardware mechanisms.

Our concept of self-replication considers the case of a given circuit, lets call it a *cell*, configured on a reconfigurable array of logic units, lets call them *molecules*. This cell is able to self-replicate, by creating a complete and exact copy of itself somewhere else on the reconfigurable array. In our case, self-replication is performed by means of *self-inspection* [7]. The self-inspection concept performs the replication by directly self-inspecting the content of the cell in order to replicate it somewhere else. In the case of a reconfigurable circuit, 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. This gain in terms of data storage leads to a more complex hardware, capable of supporting this inspection.

In our self-replication process, we split the cell into three organelles, each performing a different function in the cell during self-replication, as shown in figure 5: 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, (2) RU1 creates a copy of RU0, and (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): connections have to be created at runtime, molecules must allow a self-inspection process to retrieve configuration bits, and molecules must permit also the creation of the configuration path in order to

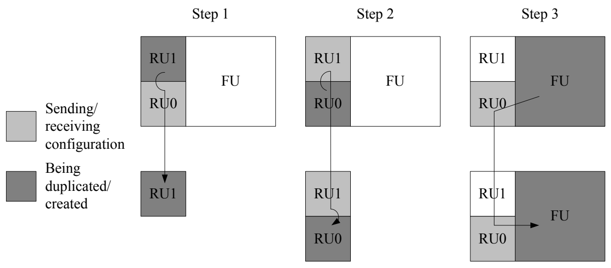

Figure 5. Self-replication principle.

determine the cell's morphology.

This process of self-replication requires specialized configurability mechanisms on the replicator and replica side. The replicator cell needs to inspect itself to retrieve its configuration bits, while the replica needs to build itself.

For this purpose, we propose the THESEUS mechanism (standing for *THeseus-inspired Embedded SElf-replication Using Self-reconfiguration*) in order to build the cell [17]. In THESEUS, an organelle is defined by a *genome* that is composed of 2 parts: (1) a set of special flags for defining the cell's morphology and (2) the configuration for each of the molecules forming the cell. We have borrowed an idea from the Tom Thumb algorithm [7] in order to manage the morphogenetic development of cell's organelles by creating a configuration path. The idea we borrowed is the way a flag is loaded into a molecule and serves then to indicate where to continue the cell construction.

The creation of an exact copy of the organelle requires to be able to recover the genome in the exact order that it was sent. So the result of pulling the configuration string must be the obtention of the same genome, that has been previously introduced. Our solution to this problem is to virtually create a shift register following the path used for the cell shape creation, and to traverse it in both directions.

An example of a 2x2-molecules organelle construction process is depicted in figure 6. The morphogenetic construction flags are initially shifted to the first molecule in order to define the organelle's shape. Thereafter, the configuration bits determining the functionality of molecule 3 are shifted in, followed by the subsequent molecules' configurations.

The flags used in the creation of such an organelle are coded in three bits and constitute the construction header for the organelle. This header is the first string to be introduced to the target reconfigurable area, followed by the configuration bits. Figure 7 shows the shift register path that is created when the construction flags are introduced on the aforementioned example. The flags are stored twice in each macrocell before filling the configuration register and sending the next flag to the next macrocell. This redundant flag storage is required in order to keep the morphology in-

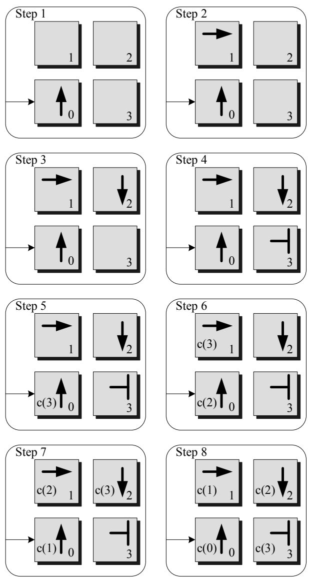

Figure 6. Creation of a 2×2 organelle, decomposed into 8 steps.

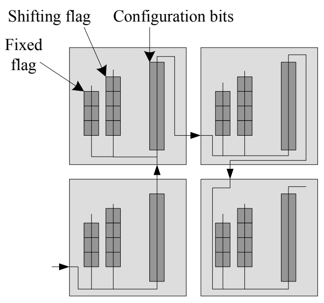

Figure 7. Shift register during the organelle configuration

formation during the recovery process.

In order to construct an organelle by means of replication, it is necessary to initially recover the configuration bitstring. This recovering process is initiated by the ubicell, allowing it to recover the bits in a serial way. Figure 8 shows the internal connectivity of the configuration bitstring in an organelle during the recovery process. During the recovery mode, the configuration bits along with the morphogenetic construction flags are shifted in the same order that they are used in the organelle construction process. During this recovering process, the output serial data extracted are directly and automatically reinjected in the shift register. Therefore, at the end of the process, the configuration of the scanned organelle remains identical as before the recovering.

For a more detailed description of the THESEUS mech-

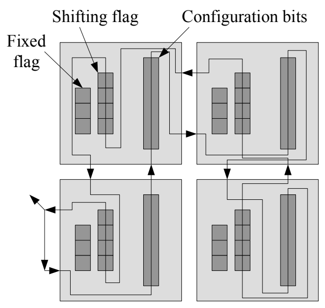

Figure 8. Shift register during the configuration recovery

anism, including cell's construction, self-inspection, replication, and hardware implementation, please refer to [17].

4.2.3 Dynamic Routing

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 network. Therefore the *ubichip* has to propose hardware mechanisms to handle this type of dynamic routing.

Considering the typical high silicon overhead due to routing matrices, specially high for dynamic routing, we chose a solution requiring the less amount of logic as possible, while being flexible enough to deal with the changing topology of the network. One of the simplest physical realization consists in the wormhole routing concept [8]. However the hardware overhead of this kind of algorithms is not suitable for the granularity of the reconfigurable array. Therefore, we will implement a dynamic routing algorithm, by improving the routing implemented in the POEtic chip [15]. The risk of congestion will be reduced by means of three features: (1) the new algorithm will better exploit the existing paths, (2) an 8-neighborhood will allow a dramatic reduction of congestion risk compared to the amount of logic required, and (3) path destruction is allowed in order to remove the unused connections. And finally, while in POEtic the circuit execution was frozen during a routing process, in the *ubichip* the creation of a new path will let the system run without interruption.

The basic idea of the algorithm is to construct paths between sources and targets by dynamically configuring multiplexers, and by letting the data follow the same path for each pair of source and target. A phase of path creation executes a breadth-first search distributed algorithm, looking for the shortest path. Sources and targets can decide to connect to their corresponding unit at any time by launching a routing process.

Special routing units will be implemented in hardware. They will be composed of the multiplexers needed to route the data, the corresponding registers required to store the multiplexers configuration, and a finite state machine to handle the routing processes. The routing units will be connected to the logic units in order to allow the implemented cells (neurons for instance) to manage the creation of new connections.

A routing process, dealing with the construction of a new path, is decomposed in 5 main phases: (1) When a source or a target wants to initiate a connection, it activates a global signal. If different logic units start such a process at the same time, priority is given to the most bottom-left unit. (2) After a master is identified, it sends serially its ID. (3)At the end of the address broadcasting, all sources and targets have compared the data with their own address and know if they are involved in the current process. (4)A breadth-first search algorithm then searches for the shortest path. (5) When the target is found, a backward signal allows for the configuration of the multiplexers present on the path, and the process ends up with a new path, allowing the newly connected logic unit to share information.

4.2.4 Scalability Issues

One of the most salient features of complex systems is the dense interaction scheme established between their constituent components. 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).

If one considers the neural application described in subsection 4.2.1, it 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 what can be attained with any foreseeable packaging technology. One of the major hardware issues to be faced by the Perplexus project is related to the scalability of the basic building blocks (*ubichips*) 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 analyse and adapt the principles involved in the Address Event Representation (AER) scheme, initially proposed in [14] and developed later in [2]. 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

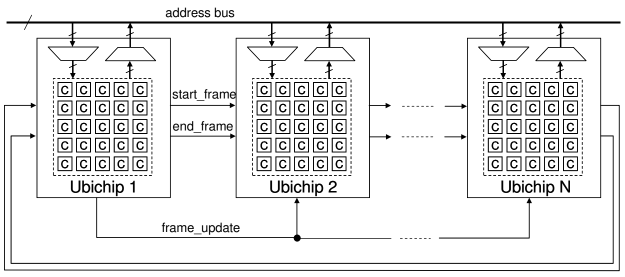

Figure 9. AER implementation.

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.

The AER communication scheme can be easily adapted to the computational needs of a distributed system such as that constituted by the Perplexus platform, where a *ubidule* can contain more than one *ubichip*. A global bus containing the address of the source components that generate events at a given time is shared between all the *ubichips*. Every *ubichip* 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 *ubichips* present in the system. In this way, every *ubichip* 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 *ubichip* is granted to access the bus. When the last *ubichip* 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 *ubichips* may update their outputs from the inputs received in the current simulation frame. Figure 9 represents the system organisation for the implementation of this communication scheme. The boxes labeled as *C* in the figure are the components that correspond to the PEs introduced in the subsection 4.2.1, which are implemented in the different *ubichips*.

It is worth noting that, because it basically consists on a multiplexed broadcasting of information, this communication scheme is valid either for a single-*ubichip* per *ubidule* scenario or for a multi-*ubichip* per *ubidule* scenario. Furthermore, this communication scheme may provide enough bandwidth for the communication needs of the applications considered in Perplexus, even in the single-*ubichip* 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 *ubichip* 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 [4]. In the case of a multi-*ubichip* 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 neural network considered within the framework of the project.

5 Summary

We introduced the Perplexus hardware platform, which goal is to provide the computing requirements for simulating our large-scale complex systems for permitting the study of emergent complex behaviors in a virtually unbounded wireless network of computing modules.

In this paper we have introduced the *ubidule*, a modular and customizable pervasive device whose core is the *ubichip*, a reconfigurable electronic chip capable of implementing bio-inspired hardware systems featuring growth, learning, and evolution. We have also presented the architectural and reconfigurability mechanisms that will allow an efficient implementation of such systems. These mechanisms are dynamic routing, distributed self-reconfiguration, and a neural-friendly logic cell architecture, keeping in mind the scalability issues that rise in the implementation of such type of complex systems.

At present, we have an operating *ubidule* prototype able to deal with USB sensors and actuators, and able to communicate via Wi-Fi and Bluetooth. We have also a full VHDL description of the *ubichip* and a software tool for editing circuit configurations. Both have been validated by implementing self-replicating neural systems featuring dynamic connectivity patterns. A first test version of the chip is currently being fabricated in UMC L180 technology.

Three applications will directly benefit from the advantages offered by the Perplexus platform: culture dissemination modeling, neurobiological modeling, and cooperative collective robotics. We will perform comparisons between classical software simulations and simulations running on a network of *ubidules*.

The Perplexus platform will thus provide unprecedented modeling capabilities thanks to the pervasive nature of the *ubidule*, its bio-inspired capabilities, its strong interaction with the environment, and its dynamical topology.

Acknowledgments

This project is funded by the Future and Emerging Technologies programme IST-STREP of the European Community, under grant IST-034632 (PERPLEXUS). 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.

References

[1] W. Barker, D. M. Halliday, Y. Thoma, E. Sanchez, G. Tempesti, and A. M. Tyrrell. Fault tolerance using dynamic reconfiguration on the poetic tissue. *IEEE Transactions in Evolutionary Computation*, 11(5):666–684, 2007.

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

[3] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424(4–5):175–308, 2006.

[4] 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.

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

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

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

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

[9] Phidgets webpage. http://www.phidgets.com.

[10] 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.

[11] 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.

[12] J. Rossier, Y. Thoma, P.-A. Mudry, and G. Tempesti. MOVE processors that self-replicate and differentiate. In A. 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.

[13] 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 Trans. on Evolutionary Computation*, 1(1):83–97, 1997.

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

[15] 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.

[16] 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.

[17] Y. Thoma, A. Upegui, A. Perez-Uribe, and E. Sanchez. Self-replication mechanism by means of self-reconfiguration. In *Workshop Procedings of the International Conference on Architecture of Computing Systems 2007 (ARCS'07)*, pages 105–112. VDE Verlag, Berlin, 2007.

[18] Toradex webpage. http://www.toradex.com.

[19] 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.

[20] A. Upegui, Y. Thoma, E. Sanchez, A. Perez-Uribe, J. Moreno, and J. Madrenas. The Perplexus bio-inspired reconfigurable circuit. In *Proc. of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems*, pages 600–605, 2007.

Answers to the editor’s comments:

Comment:

“Some typesetting errors as noticed by the reviewers must be removed:
- in the page 3, just below the figure 1, must be "purpose" instead of "pourpose".
- in the first column of the page 7, "THeseus-inspired" instead of "Theseus-inspired" and "obtaintion" instead of "obtention".”

Answer:

Done… but we didn’t corrected “obtention” because “obtaintion” does not exists.

Comment

Some minimal details - figures or tables , namely –regarding some current experimental results should be added but this just optionally and just related to the stage of the project.

Answer

We didn’t included it because we consider that it remains outside the scope of the paper. The presentation of our current partial results would imply the introduction of the applications, what would need an important amount of space, and would imply thus to reduce the description of the hardware platform, that is the goal of the paper.

Comment

Section 4.2.2 deals with the self-replication and self-repair. I agree with the reviewers opinion that “ a more detailed description, including the circuit implementation and the algorithms description, and an example would be much more contributive for this type of paper”

Answer

We included an example and some implementation details about the self-replication mechanism.