Self-Replication Mechanism by Means of Self-Reconfiguration

Yann Thoma, Andres Upegui, Andres Perez-Uribe, and Eduardo Sanchez
HES-SO/HEIG-VD, Yverdon, Switzerland
email: {yann.thoma, andres.upegui, andres.perez-uribe, eduardo.sanchez}@heig-vd.ch

Abstract

Ontogenetic hardware, along with epigenetic (neural) hardware and phylogenetic (evolvable) hardware, are the key representatives of a new hardware conception paradigm known as *bio-inspired hardware*. Ontogenesis is the process that allows living beings to develop by means of mechanisms as growing, self-replication, and self-repair. During the last few years, such ontogenetic mechanisms have been presented as a solution for the design of complex electronic circuits, with the goal of coping with the increasing complexity envisioned for future nano-technology devices. This paper presents an ontogenetic mechanism that allows a system, implemented in a reconfigurable device, to self-replicate, generating an identical copy of itself, by partially self-reconfiguring the device containing it in a dynamic way.

1 Introduction

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 [3]. Artificial neural networks, evolutionary algorithms, and self-replicating systems, are some representatives of a new, different approach to artificial life. These techniques model, to different extents, natural processes such as evolution, learning, development, or reasoning. They are intended to be tolerant of imprecision, uncertainty, partial truth, and approximation.

Building bio-inspired hardware systems demand to consider a set of paradigms different than those found in traditional hardware design techniques. The PERPLEXUS project [1] (PERvasive computing framework for modeling comPLEX virtually-Unbounded Systems) aims to tackle this issue by developing a scalable hardware platform 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.

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 ubichip will thus feature bio-inspired capabilities for providing, among others, adaptation and fault-tolerance. But, how to provide such a hardwired flexible modeling framework? There exist several research fields deeply studying and proposing computational models of specific aspects of biological systems. Neurocomputing, evolutionary computation, and fault-tolerant systems are some examples of them. However, modelling living beings 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.

If one considers life on Earth, one can distinguish three levels of organization [7]: (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 bio-inspired hardware systems can be partitioned along these three axes: phylogeny, ontogeny, and epigenesis; we refer to this as the POE model [7, 8]. We consider each of the above axes within the framework of the POE model as follows:

- The phylogenetic axis of bio-inspired systems, better known as the field of evolutionary algorithms, is the simplified artificial counterpart of phylogeny in nature. If one consider the specific case of phylogenetic hardware, one finds the domain of *evolvable hardware* [12].
- The ontogenetic axis involves the *development* of a single individual from its own genetic material, essentially without environmental interactions. Self-replication and self-repair cellular systems are based on the concept of ontogeny, where a single mother cell gives rise, through multiple divisions, to a multicellular organism. Ontogenetic hardware mainly involves *hardware implementations of self-replicating and self-repairing cellular systems*
- The epigenetic axis involves *learning* through environmental interactions that take place after formation of the individual. Artificial Neural Networks

are the main representatives of this epigenetic process, where the system's synaptic weights change through interactions with the environment. Epigenetic hardware mainly involves *neural hardware architectures* [13].

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.

For clarifying terms we will line-out some analogies between the reconfigurable logic and the biologically-inspired terminology (Figure 1). We will 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.

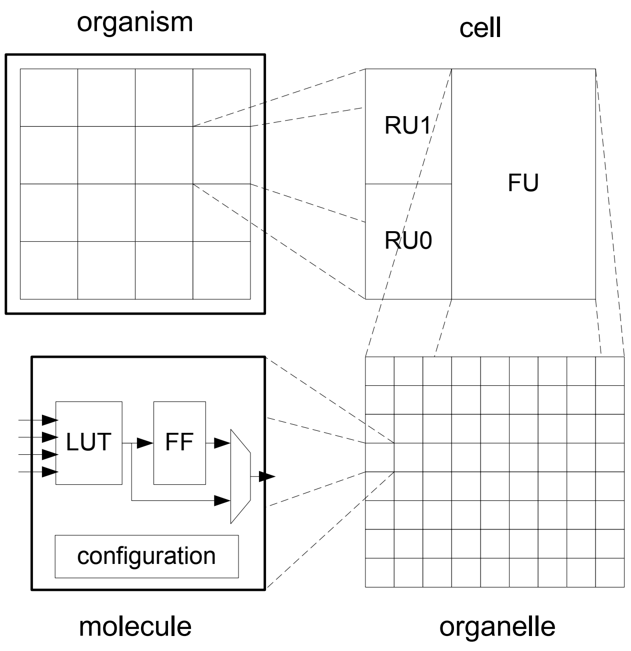

Figure 1 Hierarchical decomposition of an organism.

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. For the scope of this paper, the goal of the genome is to provide a description of the organism, and not to allow the execution of any kind of evolutionary algorithm.

In this paper we introduce THESEUS (Theseus-inspired Embedded SElf-replication Using Self-reconfiguration), and we present the hardware mechanisms required for implementing it. THESEUS focuses on the ontogenetic axis of bio-inspired hardware, more precisely on self-replicating cellular hardware systems. We present thus a hardware mechanism that allows a cell to self-replicate by generating a new cell described by the same genome. By successive cellular replications the complete organism can be created. Thanks to dynamic routing features [9] to be also included in the ubichip (which are not described in this paper), the new cell can be arbitrarily placed somewhere else in the same circuit or in a different one.

2 Ontogenetic Hardware

Ontogenetic features correspond thus to the way an organism develops from a single cell to an entire body, as well as to the capability of self-repair.

Our new platform will therefore logically allow for cellular development and self-repair. The idea behind these two concepts is to let the reconfigurable part of the chip self-organize, and 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.

We give here the description of the developmental process, derived from nature, to fully identify the requirements of the 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 [5]). 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 algorithm can be exploited by another interesting feature: 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.

Both processes of self-replication and self-repair require replication and differentiation. While differentiation can act at system level, to simply express a particular functionality depending on some factors, replication requires specific hardware mechanisms. In section 3 we describe the mechanism used by THESEUS, without considering the problem of differentiation mechanisms that would be built on top of the reconfigurable platform.

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 [2, 6, 10]. However, in POEtic, replication was possible with two limitations:

1. A configuration path had to be preconfigured 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.

While 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.

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 [11]. 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 3, and we here propose a description of self-replication at cellular level.

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. While 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.
We split the cell into three organelles, as shown on figure 2: 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

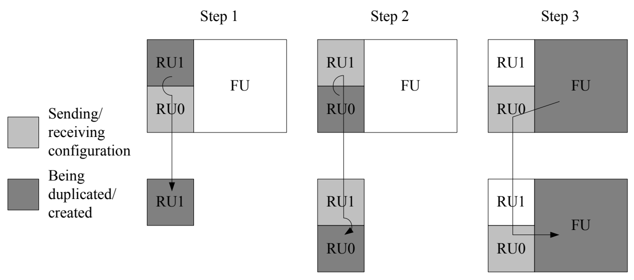

Figure 2 Self-replication process divided in three phases.

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).

2.4 Tom Thumb algorithm

Previous work on self-inspection algorithms for self-replicating systems has been done by Mange and his team with the Tom Thumb algorithm [4]. They define a cell as a set of molecules, each molecule has four memory positions, and each position stores a hexadecimal character. They define also a minimal cell that consists of 4 molecules organized as an array of 2 × 2, which implements 16 hexadecimal characters. However, only eight hexadecimal characters are needed for specifying the genome of such minimal cell, and the remaining eight are redundant data used for facilitating the self-inspection process.

These hexadecimal characters can contain two types of information: (1) the *molcode data* is the molecule code data used for configuring the artificial organism by defining the molecule's functionality, and (2) the *flag data* that defines the morphology of the cell by indicating which is the next molecule to be configured by the genome-string. The full genome is constituted by a number of molcode and flag characters (four of each for the minimal cell), arranged in a 1-d string in a manner that every odd character is a flag and every even character is a molcode.

The cell is then constructed by shifting the genome from left to right into the first molecule. When the first molecule is filled (after 4 iterations), the flag at the fourth position indicates where to continue building the cell, and the molcode stored at the third position becomes active, determining the molecule configuration. A link is created from the second position to the next molecule (i.e. the one indicated by the fourth flag), which is built in the same way. At the end, the loop must be closed by ending up in the initial molecule, and the cell is done.

For allowing the cell to self-replicate, it must be endowed with self-inspection capabilities. Because of this, the genome is shifted-in twice. The first genome will be stored in the fourth and third positions of each molecule for expressing the cell's functionality, and the second genome will continue shifting across the loop. This looping information is the one that will be copied for replicating the cell. In this manner, one of the flags creates a new link for building a complete new cell, that will contain the same configuration information that the original one. For more details on the algorithm, please refer to [4].

It must be noted that the implementation of the Tom Thumb algorithm in a reconfigurable device implies an important hardware overhead of more than twice the number of configuration bits. Keeping in mind the area constraint this overhead is not acceptable for our specifications, since the resources used for implementing the configuration registers represents an important area percentage on the whole chip, therefore we propose another self-replication mechanism.

3 Theseus mechanism

The process of self-replication requires different mechanisms on the replicator and replica side. The replicator needs to inspect itself to retrieve its genome, while the replica needs to create itself. We start with the explanation of the creation mechanism, and we continue with the self-inspection.

3.1 Organelle construction

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. Keeping in mind the nano-technology challenges, one being how to program a huge array of small reconfigurable elements, the idea behind ontogenetic hardware is to only configure a part of the circuit, 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 borrowed an idea from the Tom Thumb algorithm 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.

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 path creation requires thus the use of special flags that can take one of six possible values, described in table 1. Among these flags there is one for each direction, one to indicate the end of the building path, and a last one indicating an empty molecule that corresponds to the initial unprogrammed state.

Flag	H-flag	Signification
$\uparrow$	$\Uparrow$	develop to the North
$\rightarrow$	$\Rightarrow$	develop to the East
$\downarrow$	$\Downarrow$	develop to the South
$\leftarrow$	$\Leftarrow$	develop to the West
$\dashv$	$\Dashv$	stop development
$\circ$	$\circledcirc$	empty molecule

Table 1 Molecular flags

We insist here the difference between a flag and a h-flag. The flag is used for indicating where to continue the organelle morphogenesis. However, as the configuration is performed serially, the molecule has to detect when its flag has been received. A header is therefore used for this purpose. This header corresponds to a '1' sent before the flag, and so a h-flag corresponds to the flag data preceded by a '1'. An empty h-flag is logically coded by a chain of '0', a requirement for any unused molecule. This state has to be forced during system reset for allowing the mechanism to receive the configuration information.

An organelle's morphology is thus described by a list of molecular h-flags:

$$HF(org) = [hf(m_0), hf(m_1), \cdots, hf(m_{n-1})]$$

If we consider the examples of figure 3, the h-flags of org_0 are:

$$\begin{aligned} HF(org_0) &= [hf(m_0), hf(m_1), hf(m_2), hf(m_3)] \\ &= [\Uparrow, \Rightarrow, \Downarrow, \Dashv] \end{aligned}$$

and the h-flags of org_1 are:

$$HF(org_1) = [\Uparrow, \Uparrow, \Uparrow, \Rightarrow, \Downarrow, \Downarrow, \Downarrow, \Rightarrow, \Uparrow, \Uparrow, \Uparrow, \Rightarrow, \Downarrow, \Downarrow, \Downarrow, \Dashv]$$

We can observe that, unlike the Tom Thumb algorithm, the path doesn't need to create a loop, since the path is built as the Ariadne's thread. 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 in a serial manner, the storage of these bits correspond thus to a shift register. However, the order in which the molecular functionality configuration is sent is the inverse as that for the flags. The complete construction process is depicted in figure 4, where the configuration bits of molecule 3 are the first to be sent, the ones of molecule 0 are the last ones. As the mechanism we propose can be integrated in any kind of reconfigurable circuit, the number of configuration bits

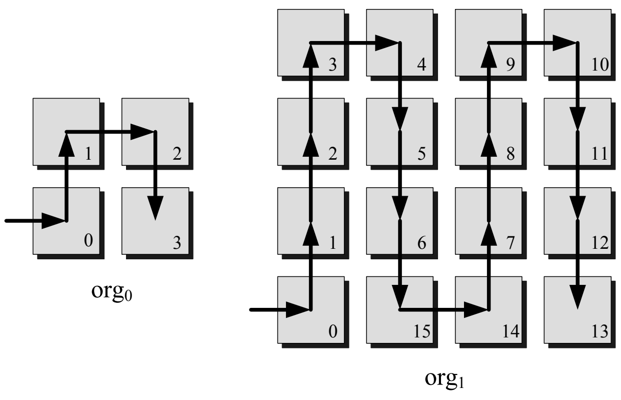

Figure 3 Possible paths for different organelle shapes

of a molecule is not relevant, and we simply represent it by $c(m_i)$ for the ith molecule. The set of configuration bits of an organelle can therefore be described 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:

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

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

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

3.2 Self-inspection

The simple construction of an organelle is not a very complicated task. However, the retrieval of its description by means of self-inspection requires more careful attention. The creation of an exact copy of the organelle requires the to be able to recover the genome in the exact order that it was sent. So the result of pulling the Ariadne's thread must be the obtention of the same genome $G(org)$, 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.

Figure 5 shows the path used to retrieve the genome for the example of the 4-molecule organelle. By shifting the entire information (the flags and the configuration bits) the retrieval can be performed in the same number of clock cycles than the organelle creation. As it can be observed in

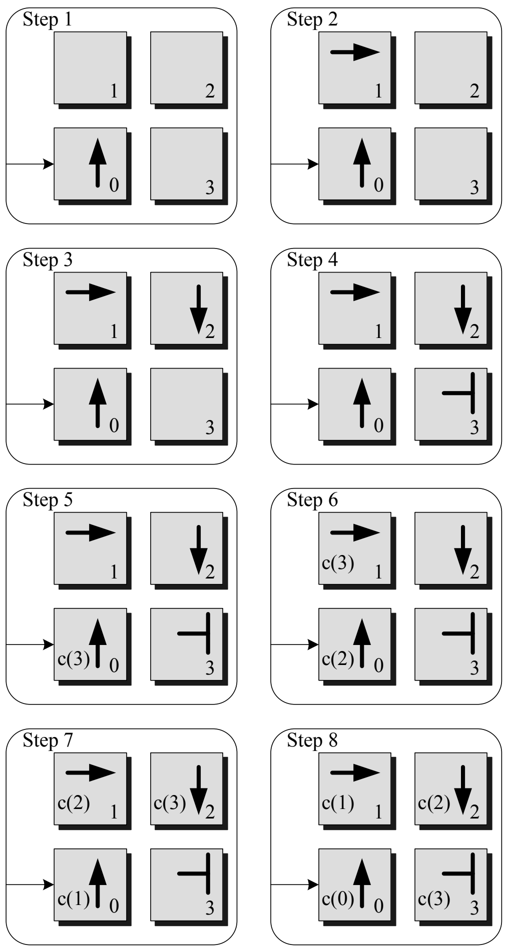

Figure 4 Creation of a 2 × 2 organelle, decomposed into 8 steps.

figure 5, the flags contain sufficient information to define the path. For instance, molecule 1, with its flag, knows where to send the configuration bits and from where the flags arrive, and molecule 0 can indicate, through a dedicated signal, that it is the source of the configuration bits for molecule 1, as well as the destination of the flags. However, while this shift register seems to be an excellent solution, it requires some special hardware, in order to keep the path unchanged for a simple reason: if the flag is shifted, then the path is destroyed.
One must thus store, in the molecule, a copy of the flag that will remain unchanged during the self-inspection process. This ensures that during self-inspection the path will not be destroyed. The storage of this extra flag implies a hardware overhead of 3 flip-flops, which is very small compared to the size of the molecule.
Figure 6 shows the flag registers, the configuration bits, and the path created by the flags during the construction of the organelle. The control system, which for the sake of simplicity is not represented here, is mainly composed of multiplexers to select the origin of the serial bits and of demultiplexers to indicate where to send these bits further.

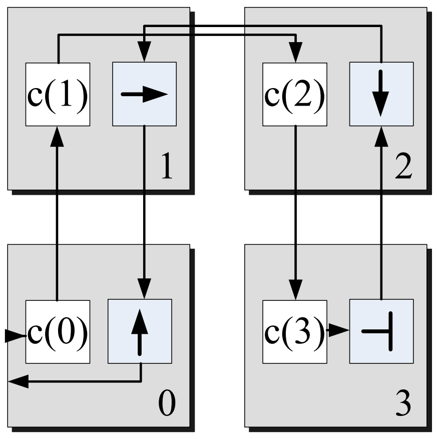

Figure 5 The path used to retrieve the genome of the organelle

The registers' shift-enable is controlled by combinatorial logic that depends of the header-bit of the shifting-flag register. When this header-bit equals '0' both flag-registers shift, while the configuration bits are disabled. When the header-bit equals '1' - i.e. when the flags registers are configured - then the configuration bits are shifted during the rest of the process.
During the self-inspection process, the fixed flags indicate exactly the directions of the path, the other registers shifting serially their contents, as presented in figure 7.
After an organelle replication, the content of the configuration bits and the flags must recover its original state. This can easily be guaranteed by reinserting the genome in the organelle. So, at every clock cycle, the retrieved bit is forwarded. In our example, the west output of molecule 0 will be directly connected to its west input.

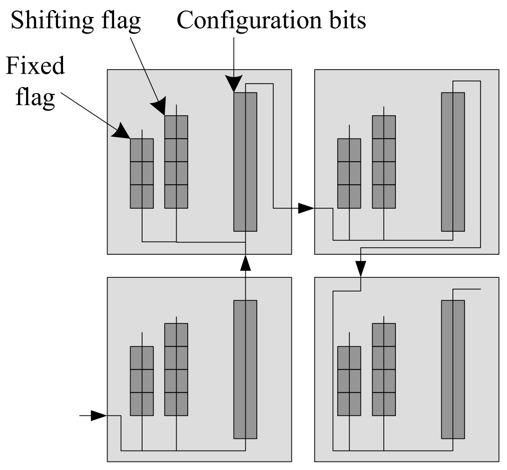

Figure 6 Register paths during the construction of an organelle.

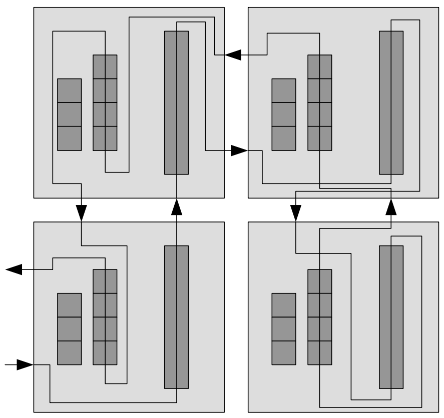

Figure 7 Register paths during the inspection of an organelle.

4 Setup and Results

4.1 Validation setup

For validating our self-replication mechanism we implemented a molecular array. Each molecule is configured by a small configuration bit-string of 80 bits which is stored in the configuration register. The functionality described by this configuration bit-string is irrelevant for the self-replication process. The number of 80 configuration bits has been arbitrarily chosen, and corresponds approximately to the number of configuration bits required for a molecule composed of a 4-input LUT, a flip-flop, 4 input multiplexers and a switchbox.
We configured the molecule array with a configuration bitstream that describes a 4-molecule organelle arranged 2 × 2. It must be noted that, unlike the Tom Thumb algorithm introduced in 2.4, this is not our minimal cell. Our mechanism allows to have a minimal organelle composed of a single molecule, and the minimal cell size will be further determined by the replicators' implementation.
The 4-molecules organelle will be thus described by a genome of the form: $[\Uparrow, \Rightarrow, \Downarrow, \Leftarrow, c(m_3), c(m_2), c(m_1), c(m_0)]$. The four first characters are the 4-bit flags that denote the cell morphology. The last four characters contain the configuration information of each of the four molecules, being each character an 80-bit configuration bitstream. The total size of the configuration string is thus $4 \times (4 + 80) = 254$ bits, among which only 16 bits are used as flags.
The system was described in VHDL at the RTL level and has been simulated with ModelSim. The self-replication process has been validated by configuring the molecule array with the aforementioned genome. The genome has correctly configured the organelle in the molecule array in 254 clock cycles. The self-inspection mechanism has allowed to correctly recover the genome, in the same number of clock cycles, for configuring a second organelle.

4.2 Synthesis results

The implementation of the full system comprising the configuration registers, along with the self-replication mechanism, required a total amount of 384 transistors and 87 flip-flops . This number has been calculated based on the schematics generated by Synplicity. It will be refined in the future, after the realization of the physical circuit.
As the size of registers is very important, regarding the transistors size, we can consider, for a molecule of 80 configuration bits, that the overhead will be around 10% of the total silicon area. This percentual overhead can be considerably reduced when including the logic cell's functionality.

5 Conclusions and Further Work

In this paper we presented THESEUS, a self-replication hardware mechanism for designing systems able to grow by means of cellular self-replication. In this way, by using a relatively small configuration bitstream (describing a single cell), it is possible to generate a complete highly complex organism composed of several cells. Cell differentiation mechanisms will be further included at a higher level.
The presented mechanism can also be useful for designing fault-tolerant circuits. The dynamic nature of our self-replication mechanism allows it to be used in hostile environments that may damage the device. The self-replication mechanism can allow a cell to repair another cell by simply rewriting its configuration if it has been (partially or completely) erased, or by re-constructing it in a different set of molecules if transistors have been physically damaged.
The silicon overhead required for implementing our self-replication mechanism remains completely acceptable. It only requires 7 registers and a small amount of logic gates, what is not a high cost compared to the amount of transistors required for building the functional part of a molecule. This typically corresponds to a 10% overhead for a molecule similar to the one of POEtic, and is reduced with the growth of the reconfigurable element.
The ubichip's *logic cell* architecture is still to be specified. However, the proposed self-replication mechanism will apply to any cellular architecture independent of the level of granularity.
There are still some open issues to develop before the silicon implementation of the mechanism:

- During self-inspection and configuration the cell cannot perform its functionality: the bitstream is being shifted across the cell's configuration register producing invalid circuits. Must cell's functionality be disabled during this self-inspection process? If yes, how to disable it?
- Cellular death makes part of the ontogenetic process. The current self-replicating system allows to give birth to a cell, but cell's destruction is not still supported. The implementation of such destruc-

tion mechanism would be interesting for increasing molecules' reusability.

- The presented mechanism assumes that there are enough empty molecules available for building the cell. It will be necessary to include a mechanism that allows to incrementally verify molecule availability, and to interrupt the cell's construction if not.
- The replicator is not a silicon embedded circuit, but a system implemented on top of the reconfigurable logic. A compact replicator architecture is still to be defined in function of the *logic cell* chosen.
- As the replicator, the differentiation mechanism is a system implemented on top of the reconfigurable logic. Several differentiation mechanisms can be explored depending on the targeted cellular functionality.

Acknowledgements

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.

6 References

[1] PERPLEXUS webpage. http://www.perplexus.org.

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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 T. Higuchi, M. Iwata, and W. Liu, editors, *Evolvable Systems: From Biology to Hardware*, volume 1259 of *LCNS*, pages 33–54, Berlin, 1997. Springer-Verlag.

[8] 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.

[9] 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.

[10] Y. Thoma, G. Tempesti, E. Sanchez, and J.-M. Moreno Arostegui. Poetic: An electronic tissue for bio-inspired cellular applications. *BioSystems*, 74(1-3):191–200, August-October 2004.

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

[12] X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. *IEEE Trans. on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 29(1):87–97, 1999.

[13] J. Zhu and P. Sutton. FPGA implementations of neural networks - a survey of a decade of progress. In P.Y.K. Cheung, G.A. Constantinides, and J.T. de Sousa, editors, *Proc. of the 13th International Conference on Field Programmable Logic and Applications (FPL'03)*, number 2778 in LNCS, pages 1062–1066, Berlin, Heidelberg, September 2003. Springer Verlag.