MOVE Processors That Self-replicate and Differentiate

Joël Rossier, Yann Thoma, Pierre-André Mudry, and Gianluca Tempesti

Ecole Polytechnique Fédérale de Lausanne (EPFL),
Cellular Architectures Research Group (CARG),
CH-1015 Lausanne, Switzerland
j.rossier@epfl.ch

Abstract. This article describes an implementation of a basic multi-processor system that exhibits replication and differentiation abilities on the POEtic tissue, a programmable hardware designed for bio-inspired applications [1, 2] . As for a living organism, whose existence starts with only one cell that first divides, our system begins with only one totipotent processor, able to implement any of the cells required by the final organism, which can also fully replicate itself, using the functionalities of the POEtic substrate. Then, analogously to the cells in a developing organism, our just replicated totipotent processors differentiate in order to execute their specific part of the complete organism functionality. In particular, we will present a working realization using MOVE processors whose instructions define the flow of data rather than the operations to be executed [3]. It starts with one basic MOVE processor that first replicates itself three times; the four resulting processors then differentiate and connect together to implement a multi-processor modulus-60 counter.

1 Introduction

Multi-cellular organization is one of the key concepts for a lot of living creatures. In fact, almost every organism, except viruses and bacteria, is based on this structure that enables an individual to develop an astounding complexity, starting from only one relatively simple cell. Moreover, being a multi-cellular organism offers more possibilities like being able to tolerate some faults, to self-repair or to exhibit self-healing capabilities.

For several reasons, such abilities could obviously be of great interest for multi-processor systems. One of the first reasons is the programmability of a group of processors having to execute collectively a specific task. Today, we can still program individually each processor of the set and give it a specific code. But the size of the electronic components is continuously shrinking and we will soon enter in the era of nano-electronics. In such a case, the processor arrays will have to be realized on an homogeneous substrate consisting in a lot of massively parallel basic nano-components. As a result, it will be very difficult, perhaps even impossible, to initialize one by one each processor of such an array.

A.J. Ijspeert et al. (Eds.): BioADIT 2006, LNCS 3853, pp. 160–175, 2006.

Consequently, the multi-processor systems of tomorrow could take advantage of a self-replication/differentiation mechanism to be more easily configured.

Then, as the scale of electronics will decrease, faults will happen in the circuits with a greater probability than today. Consequently, it could be useful to have a system that could tolerate some faults or, at least, avoid some parts of the circuit where the faults are detected.

In this paper, we propose an implementation of a system showing such capabilities consisting in one totipotent processor, i.e. a processor capable of executing all the sub-tasks required by an application, that first replicates itself. Then, the cloned processors bind themselves and differentiate in order to achieve a full multi-processor system. To illustrate this process, we will use a simple system that implements a watch counter using four processors. The system has been realized in a slightly modified version of the POEtic tissue [1, 2], a reconfigurable logic circuit structure especially designed for bio-inspired applications that will be presented in section 2.2.

In the following sections, we will first describe the background used to realize our system: we will succinctly present the Embryonics project in section 2.1. Then in section 2.2 we will recall the main characteristics of the POEtic tissue. In section 2.3, we will then expose briefly the MOVE paradigm, also known as Transport Triggered Architecture (TTA), on which our processors are based. These bases in mind, we will present the general architecture of our processor in section 3. Then, the self-replication and the differentiation/connection processes will be explained in sections 4 and 5 respectively. The following section will deal with the hardware realization of the system and its implementation on the BioWall [4]. Finally, we will conclude this article with a section discussing the future developments that our system will undergo.

2 Background

Before describing concretely our system, we will now expose the background from which we started the development of our self-replicating processors that differentiate and bind together. We will briefly present the Embryonics project and its major realization, the BioWatch. Then we will describe more thoroughly the POEtic substrate. To close this section, we will present the Transport Triggered Architecture, also known as the MOVE processor paradigm.

2.1 The Embryonics Project

The application of biological ontogenesis to the design of digital hardware has been studied for several years within the Embryonics project [5]. One of its major contribution to the field is the self-contained representation of a possible mapping between the world of multi-cellular organisms in biology and the world of digital hardware systems, based on 4 levels of complexity, ranging from the population of organisms to the molecule (Fig. 1).

Within this mapping, the Embryonics project defines an artificial organism as a parallel array of cells, where each cell is a simple processor that contains the

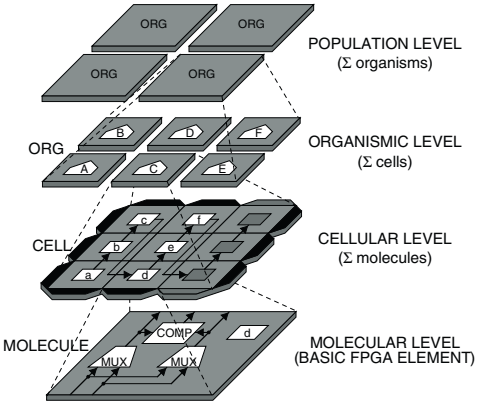

Fig. 1. The four hierarchical levels of complexity of the Embryonics project

description of the operation of every other cell in the organism in the form of a program (the genome). This program, replicated in each cell of the organism as in a living being, is read in parallel in each cell but different parts of it are executed depending on the spatial coordinates of the cell within the organism. The redundancy inherent in this approach is compensated by the added capabilities of the system, such as growth [6] and self-repair [7]. The molecules are defined as the basic elements of the programmable logic circuits; in the Embryonics project, they correspond to simple programmable multiplexers.

A configuration bitstream (the genome of the artificial organisms) is injected into the circuit, causing the molecules to self-assemble into cells. The cells themselves, after a replication phase analogous to cellular division and growth, self-organize to form the final organism.

Using this approach, the Embryonics project demonstrated two basic properties of its substrate with the implementation of the BioWatch [8], an electronic modulus-60 counter made of four cells exhibiting differentiation and fault-tolerance abilities. We have decided to use this same application to demonstrate the capabilities of our system.

2.2 The POEtic Tissue

Bio-inspiration in the design of digital hardware finds its source in essentially three biological models [9, 10]: Phylogenesis (P), the history of the evolution of the species through time, Ontogenesis (O), the development of an individual as directed by his genetic code, from its first cell to the full organism, and Epigenesis (E), the development of an individual through learning processes. All of these models, to a greater or lesser extent have been used as a source of inspiration for the development of computing machines (such as the ontogenesis in the Embryonics project or epigenesis for artificial neural networks) but before the POEtic project [1, 2], no hardware substrate had been developed that could combine the three axes of bio-inspiration into one single circuit.

Indeed, the POEtic tissue draws inspiration from these three axes and from the multi-cellular structure of complex biological organisms. This reconfigurable circuit has been designed to develop and adapt its functionality through the processes of evolution, growth and learning. The organizational architecture of a POEtic system is the same as the one of an Embryonics design: it also follows the four levels of complexity defined in figure 1, once again from the population of organisms to the molecular level.

Physically, the tissue is composed of two layers shown in the left of figure 2: a grid of molecules and a cellular routing layer. As in Embryonics, the smallest units of the POEtic programmable hardware are also called molecules and are also arranged as a two-dimensional array. The cellular routing layer is also a two-dimensional array but contains special routing units that are responsible for the inter-cellular communication. This routing layer implements a distributed routing algorithm based on identifiers allowing the creation of data paths between cells at runtime. Each molecule, as well as each routing unit, are connected to their respective four neighbours in a regular structure, also shown in the left of figure 2. Moreover, the molecules have the capability of accessing the routing units to set up connections among cells.

As shown in the right of figure 2, a molecule mainly contains a 16-bit look-up table (LUT) and a D flip-flop (DFF); its inputs are selected by multiplexers and its outputs are routed to any direction through a switchbox. Moreover, a molecule possesses different configurable operational modes that let it act of different manners. The content of the LUT and of the DFF, as well as the selection of the multiplexers for the inputs and the outputs of a molecule and the mode in which the molecule has to work, are defined by 76 bits of configuration.

In the first four operational modes, that are quite standard in the reconfigurable hardware area, a molecule can be configured as a simple 16-bit LUT, as two 8-bit LUT, as a 8-bit LUT plus a 8-bit shift register, or as a 16-bit

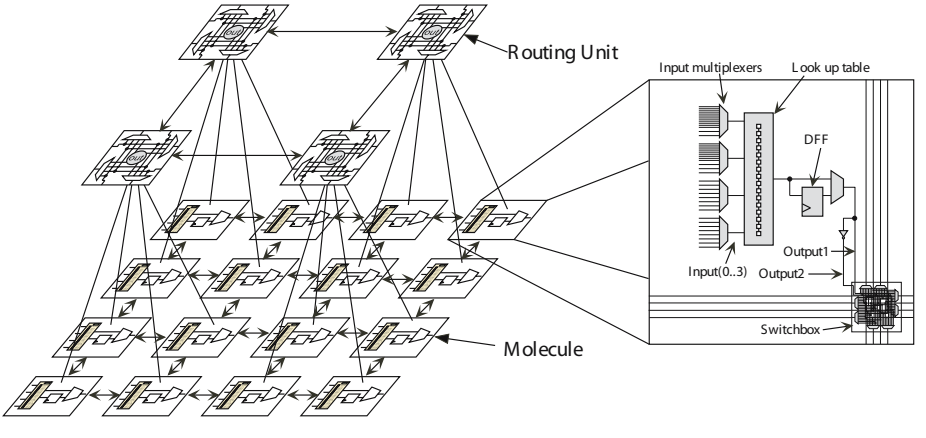

Fig. 2. Left: POEtic two-layer physical structure with the molecules and their routing units. Right: Basic structure of a POEtic molecule.

shift-register. Then there are four additional operational modes that are specific to the POEtic tissue: the first two are the Output and Input modes in which the molecule is connected to its routing unit and contains the 16-bit long routing identifier of the molecule itself, respectively of the molecule from where the information has to arrive. The third special mode is the Trigger mode, in which the task of the molecule is to supply a trigger signal needed by the routing algorithm for synchronization purposes. The last mode is the Configure mode, in which a molecule has the capability of partially reconfiguring its neighbours, i.e. the molecule can modify a fixed subset of the configuration bits of its neighbours (68 bits out of 76).

Inter-molecular communication, i.e. short-range communication between the programmable logic elements in the POEtic circuit, is implemented by a switch box (identical in all molecules) that prevents the possibility of short circuits in the network by using multiplexers and directional lines. There are two of these lines from and to each cardinal direction.

Inter-cellular routing, i.e. long-range communication between the processors implemented using the programmable logic, is implemented using a distributed routing algorithm inspired by Moreno [11], that automatically connects the cells inputs and outputs. A non-connected input (target) or output (source) can initiate the creation of a path by broadcasting its identifier, in case of an output, or the identifier of its source, in case of an input. The path linking them is then created using a parallel implementation of the breadth-first search algorithm, similar to Lee's algorithm [12] that configures multiplexers in the routing units. When all the paths have been created, the organism can start operation, and executes its task, until a new routing is launched.

Note that in the standard POEtic design, in the IO modes, the molecules only have one control signal that forces or not a connection to be established. In addition to this, to implement self-replication we had to slightly modify the standard POEtic design in order to improve the IO molecules with another control signal that makes the molecule to accept or not a connection. As a result, our version of the POEtic IO molecules has two control signals: one to force a molecule to establish a connection, i.e. `ForceConnect`, the other to accept the connections, i.e. `AcceptConnect`.

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

2.3 MOVE Processors

We will now present the basic processor structure that has been used for the realisation of our system: the MOVE architecture, also known as the Transport-Triggered Architecture (TTA) [3, 13, 14]. This paradigm was originally developed

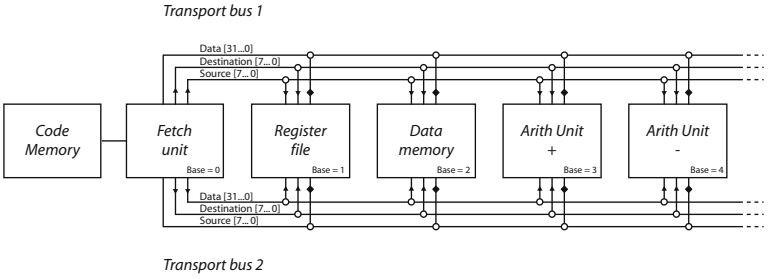

Fig. 3. Internal structure of a TTA processor

for the design of application-specific dataflow processors (processors where the instructions define the flow of data, rather than the operations to be executed).

In many respects, the overall structure of a TTA-based system is fairly conventional: data and instructions can be fetched to the processor from the main memory using standard mechanisms (caches, memory management units, etc.) and are decoded as in conventional processors. The basic differences lay in the architecture of the processor itself, and hence in the instruction set.

Rather than being structured, as is usual, around a more or less serial pipeline, a MOVE processor (Fig. 3) relies on a set of Functional Units (FUs) connected together by one or more transport busses. All the computation is carried out by the functional units (examples of such units can be adders, multipliers, register files, etc.) and the role of the instructions is simply to move data from and to the FUs in the order required to implement the desired operations. Since all the functional units are uniformly accessed through input and output registers, instruction decoding is reduced to its simplest expression, as only one instruction is needed: `move`.

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

3 Processor Architecture

After the presentation of the background used for our realization, we will now describe it more precisely. As mentioned, our test system is composed of four processors, the cells, that form a 4-digit modulus-60 counter, the organism, counting seconds and minutes. Each of the processors must then handle one digit. Consequently, two of them count from 0 to 9 while the two others count from 0 to 5. In their final configuration, they are logically organized so as to

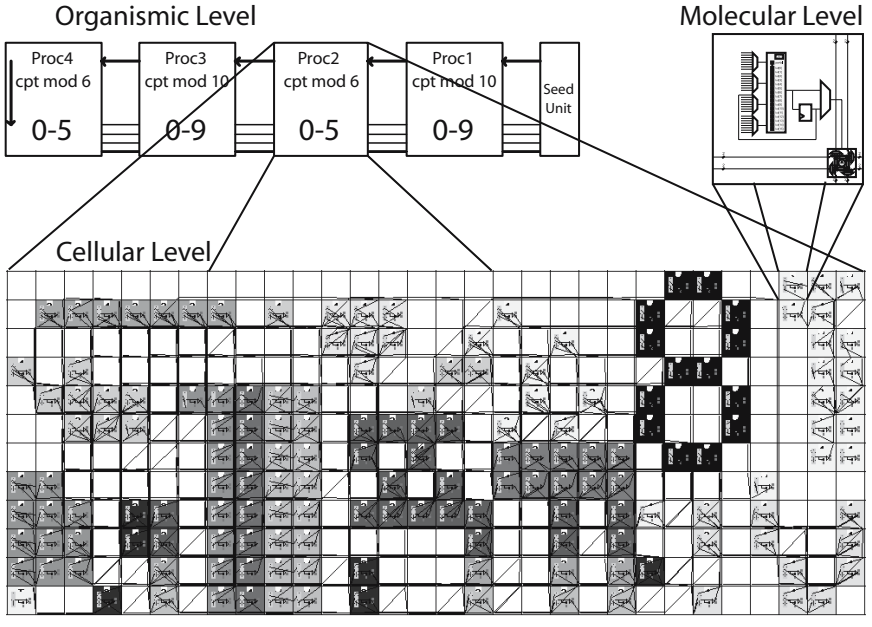

Fig. 4. The three hierarchical levels of our system: organism/counter final configuration, cell/processor mapping on POEtic, molecule/POEtic element

form a chain that is represented in the organismic level of figure 4 (note also the presence of the Seed Unit, whose function will be explained in section 5).

The normal operation of the system is the following: the processor that handles the rightmost digit, i.e. the units of seconds in the clock parallel, permanently counts from 0 to 9. When this processor arrives at 9, it generates a signal (`EnableCount`) telling the next processor, which handles the tens of seconds, to increment its own digit. When the tens of seconds processor arrives at 5, it generates in its turn a signal enabling the next processor on the chain, i.e. the units of minutes, to count. And so on until the tens of minutes.

As exposed in the precedent section, we realized our processors using the MOVE paradigm. Its actual implementation in POEtic molecules is shown in the cellular level of figure 4, while its logical architecture can be seen in figure 5. It resulted in a TTA processor possessing the following Functional Units (FU):

- **FU Cmp** used to compare two values. The result is directly given to the Execution Stack (see below for a short explanation).
- **FU Inc** used to increment one value.
- **FU Position** used to get the position of the processor inside the chain.
- **FU EnableIn** used to get the value of the `EnableCount` signal coming from the precedent processor on the chain.
- **FU EnableOut** used to set the value of the signal enabling the counting of the next processor on the chain.

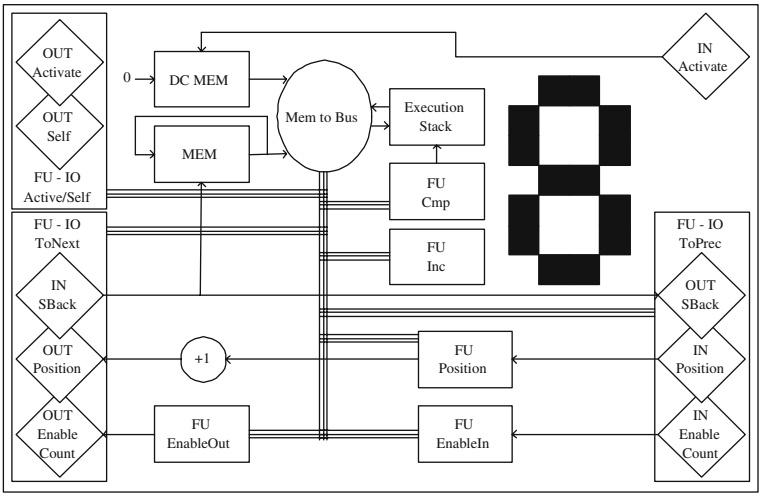

Fig. 5. Detailled architecture of the processor

- **FU IO Active/Self** used to connect one not yet differentiated processor, or used by the processor to connect itself in order to enable the whole system when the differentiation/connection process is finished (see section 5 for more details).
- **FU IO ToPrec** used to set up and configure the connections to the precedent processor on the chain. It is used to receive the processor position inside the chain and the `EnableCount` signal. It is also used to transmit the `Sback` signal whose purpose is explained in section 5.
- **FU IO ToNext** used to set up and configure the connections to the next processor on the chain.

The three FU IOs permit the processor to control the behaviour of the Inputs and Outputs: the processor can access and set up the `ForceConnect` (to force a molecule to establish a connection) and the `AcceptConnect` (to allow a molecule to accept the connections) control signals by setting the appropriate values in the FU IO registers.

Our MOVE processor, as is usual, contains a data bus spanning all the FUs and two memory busses: one for the source addresses and the other for the destination addresses of each `move` instruction. The processor has two memories: one memory (MEM) for the normal operation of the processor (i.e. the counting and the generation of signals) and another memory (DC MEM) that contains the code for the differentiation and connection mechanisms.

Then, as the processor has been realized on the POEtic substrate, which provides a specific molecule mode to implement shift memories (see section 2.2), we decided that, instead of an addressable memory that could support jumps in the code, we would use cyclic memories, where each instruction is read successively, and executed or not, depending on a special unit called Execution Stack.

To summarize the behaviour of the Execution Stack, we can say that, when facing an *"if condition then (x1; x2; ...) else (y1; y2; ...) end"* instruction, if the condition is valid, the stack will permit the execution of the X instructions and then block the Y instructions. Otherwise, it will permit the Y execution and block the X one. A more detailed explanation of this unit can be found in [15, 16].

Finally, for demonstration purposes, we added a special unit that is used to display the digit handled by each processor.

4 Self-replication

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

The self-replication process that we have implemented is based on the self-inspection concept [17], where, in order to replicate itself, a system has to generate its description by examining its own structure. This description is then used to create an identical copy of the original system [18].

More precisely, such a self-replication process in our reconfigurable circuit should proceed as follows: first, the cell that wants to replicate itself has to emit the configuration bits of every one of its molecules. Then, in some way, these bits are routed to their destination, i.e. the place where the copy will be constructed. These configuration bits are then injected into molecules that are not yet configured. These molecules receive their new configuration and become copies of the initial molecules. When all the configuration bits of each molecule of the initial system have been emitted, routed and injected in their new place, the cell has replicated itself.

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

For that purpose, we had to separate our self-replicating processor in two parts: a functional part (FP) that contains the object we want to replicate, i.e. the MOVE processor itself, together with its corresponding replication path, and a self-replication part (SRP) that is of course in charge of the self-replication (Fig. 6).

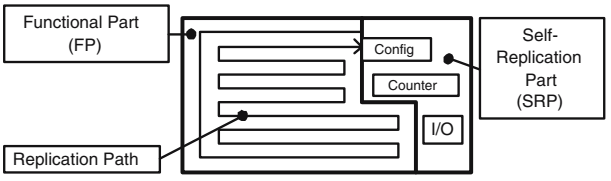

Fig. 6. Mandatory parts for a POEtic self-replication

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

We will now detail the self-replication process that uses the self-configuration ability of the POEtic molecules as well as their distributed routing. At the beginning, as shown in figure 6, the system contains the following elements:

- **Functional Part FP** the processor that has to be replicated (Fig. 5).
- **Emitting SRP** that contains an Output molecule and is used to connect to one or more receiving molecules.
- **One or more Receiving SRP** that contain an Input molecule and are used to receive the connection from the Emitting SRP.
- **Replication Paths** that are already configured. The first path span all the molecules of the FP. The others draw the same trajectory as the first path and are placed next to the Receiving SRP.

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

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

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

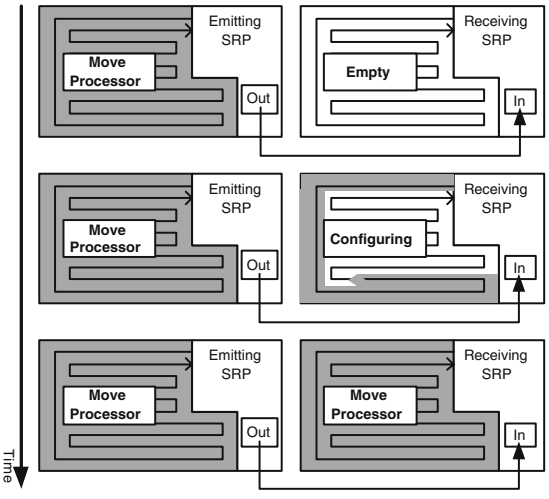

Fig. 7. Three steps of the self-replication process

of transmission of the configuration bits, the Emitting SRP and its replication path emulate a shift register buckling on itself, so that the FP finds again its initial state.

On the other side, the Receiving SRP gets the configuration bit from its Input molecule and injects it in its own replication path. This process repeats itself during a number of clock cycles determined by the SRPs and that is equal to the total number of configuration bits that have to be expressed, i.e. 68 bits that are configurable per molecule times the number of molecules to be replicated.

When the configuration is finished, the system contains two (or more) replicated FP that can start their normal functionality.

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

5 Differentiation and Connections

In living organisms, when the first cell has divided, resulting in many totipotent identical cells, these latter have to specialize to handle a specific task that depends on their neighbouring cells and on the place they have inside the entire organism. As a result, the cells differentiate and connect themselves together to form the working organism.

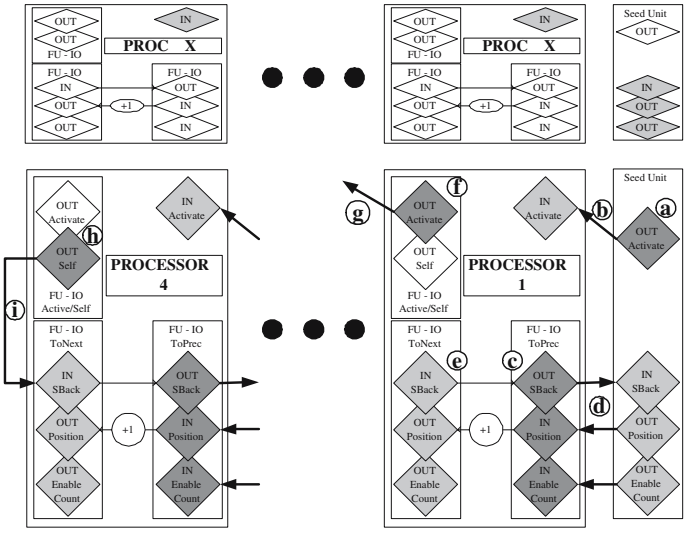

Fig. 8. Processor differentiation and connections. Top: quiescent state of the processors just after the self-replication. Bottom: connection process and differentiated processors.

Similarly, after the self-replication phase, the POEtic substrate contains four identical quiescent totipotent processors that still need to differentiate and connect in order to achieve the whole system functionality. This situation is shown at the top of figure 8 (note that only the IO elements of the processors are represented and that on the top of the figure the labels are not detailled).

In fact, the processor are waiting for an activation signal that will launch the differentiation/connections process. In order to generate this first activation signal, we implemented a special unit: the Seed Unit (SU). It possesses a counter that makes it wait for the end of the processor self-replication. At that time, the SU activates the `ForceConnect` control signal that forces the connection of its Output **OUT Activate**, *(a)* in figure 8. This Output will then initiate a distributed routing process to connect the nearest **IN Activate** molecule that is configured in order to accept the connections *(b)*. Note that all the quiescent processors have their **IN Activate** molecule waiting for a connection, i.e. with their `AcceptConnect` control activated.

As a result, the nearest replicated processor accepts the connection, becomes linked to the SU and receives an activation signal through the newly established connection. This activation signal makes the processor to activate its differentiation and connection memory (DC MEM in figure 5) and start the shifting and the execution of its instructions.

The first instruction makes the `ForceConnect` control of the FU_ IO_ ToPrec be activated *(c)*: the IO molecules of this FU will immediately try to initiate new connections. The only available corresponding molecules that have their `AcceptConnect` control activated are the ones of the SU, consequently these molecules become linked *(d)*. As a result, the processor gets from these new

inputs its position inside the chain (zero in this case) as well as its `EnableCount` signal.

Then the DC MEM makes the processor activate the `AcceptConnect` controls of its FU_ IO_ ToNext *(e)*; this is done in order for the next processor on the chain to be able to connect back.

Finally, as the processor now knows it is not the last one of the chain, having already received its position from the FU_ IO_ ToPrec, it will activate the `ForceConnect` control of its **OUT Activate** molecule *(f)*. This molecule will then establish a connection to the next available **In Activate** molecule *(g)*, and launch the differentiation-connection process of the second processor.

The second processor will then execute this process again, connect its FU_ IO_ ToPrec to the first processor (whose FU_ IO_ ToNext now accept the connections). The same process happens for the third processor on the chain.

For the last processor, the points *(f)* and *(g)* are not executed: as the processor knows it is the last one of the chain, it does not need to connect to another processor but must inform the whole chain that the differentiation-connection process is finished and that the system now has to begin its normal multi-processor activity. Consequently, instead of its **OUT Activate** molecule, the fourth processor will activate the `ForceConnect` control of its **OUT Self** molecule *(h)*. This latter will then connect to the **IN SBack** molecule of its FU_ IO_ ToNext *(i)*.

This last connection provides an activation signal that is transmitted through the whole processor chain using the **In SBack** and the **OUT SBack** IOs. This signal activates the processors MEM memories (figure 5), whose instructions contain the code required to execute each of the functionalities needed by the application. The spatial position of the processor inside the chain, defined through the differentiation process, is used to select the appropriate functionality.

6 Hardware Implementation

To have access to a sufficient number of molecules and to be able to integrate our modifications to the design, we decided to emulate the POEtic substrate on the BioWall [4], a two-dimensional electronic wall designed for bio-inspired applications and composed of an array of reconfigurable circuits.

We made two major modifications to the standard POEtic specifications: the first one is the improvement in the control signals of the IO molecules explained in section 2.2. The second one is the following: unlike the standard POEtic connection schema shown in the left of figure 2 where each routing unit is simultaneously connected to four molecules, we realized our POEtic implementation with one routing unit per molecule, permitting a denser connection pattern.

The realization of one of our processor, with its self-replicating part, needs 30x12 POEtic molecules to be implemented. Using the BioWall for the implementation, we have 25x80 POEtic molecules available, which is sufficient to demonstrate the self-replication, the differentiation/connection process and finally the normal operation of our multi-processor system.

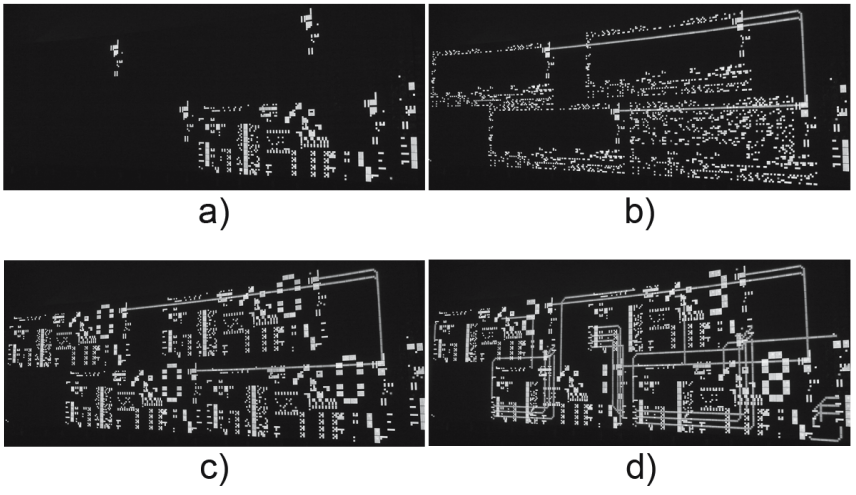

Fig. 9. a) Initialization state. b) Self-replication phase. c) The four totipotent processors before the differentiation/connection phase. d) Operational system.

Moreover, the display capabilities of the BioWall allows us to visually check and demonstrate the correct behaviour of the entire system: some pictures of it are shown in figure 9 and a video of the whole process can be found at `http://carg2.epfl.ch/Staff/JR/Videos/PoeMoveSR.avi`.

7 Conclusion and Future Developments

We have realized a multi-processor system that exhibits self-replication, differentiation and distributed connection abilities. Moreover, we have implemented the whole system in hardware on the BioWall, demonstrating the feasibility of the concepts. Nevertheless, a number of things can certainly be improved.

With its distributed connection ability, our system can bind together processors that have no fixed predetermined place on the substrate. If the system had a cellular fault-detector, it could detect and disable faulty processors. As a result the differentiation/connections process would automatically avoid the faulty processor and connect to the next correctly working one.

From another point of view, our system can not tolerate individual errors. As a result, one of the improvements that could be added to the POEtic molecules, as in the Embryonics project [7], is a molecular fault-tolerance capability.

Moreover, as already mentioned in section 4, the POEtic substrate has only partial self-configuration abilities (the configuration bits that define the replication path can not be changed by the system). As a result, the replication paths must be pre-configured in order to cross all the molecules that have to be replicated. One major planned improvement consists of changing the POEtic specification by allowing the system to set or reset each configuration bit and consequently enabling a complete self-replication.

Another improvement could be to differentiate the memory: in our system, each processor possesses the same memory and executes the instructions or not, depending on its position in the processor chain. To limit this redundancy we could modify the memories and the differentiation process in order to copy only the instructions needed by a specific processor in its specific memory.

Then, our system replicates and differentiates only once at the beginning. We are currently working on a way to make these processes occur permanently during the life of the organism, allowing in that manner growth, adaptation and re-configuration in case of failures.

Despite all the things that we plan to integrate to future designs, we can already say that in its current state, our realization is a real improvement compared to the existing ones for several reasons. Firstly, contrary to the Embryonics project, where the genome had to be injected in parallel in each cell, in our design we only need to provide the genome one time to the circuit.

In the Embryonics project again, the circuit had been designed expressly for the realization of a watch counter. With the use of the MOVE paradigm, our design is much more versatile and can be modified very quickly to adapt to any logical task, just by adding some Functional Units.

Moreover, as mentioned in the precedent section, compared to the standard POEtic design, we made some improvements on the IO molecules and on the routing layer in our hardware implementation.

Finally we can say that, even if some consequent work remains to be done, our design is one good step ahead in the realization of a really efficient self-replicating electronic system.

References

1. Tyrrell A., Sanchez E., Floreano D., Tempesti G., Mange D., Moreno J.-M., Rosenberg J., Villa A., POEtic Tissue: An Integrated Architecture for Bio-Inspired Hardware, Proceedings of the 5th International Conference on Evolvable Systems: From Biology to Hardware (ICES'2003), pp. 129-140, 2003
2. Thoma Y., Tempesti G., Sanchez E., Moreno J.-M., POEtic: an electronic tissue for bio-inspired cellular applications, BioSystems 76, pp. 191-200, 2004
3. Tabak D., Lipovski G.J., MOVE architecture in digital controllers, IEEE Transactions on Computers C-29, pp. 180-190, 1980
4. Tempesti G., Mange D., Stauffer A., Teuscher C., The BioWall: An Electronic Tissue for Prototyping Bio-Inspired Systems. Proceedings of the 2002 NASA/DOD Conference on Evolvable Hardware, pp. 221-230, 2002
5. Mange D., Sipper M., Stauffer A., Tempesti G., Towards Robust Integrated Circuits: The Embryonics Approach, Proceedings of the IEEE 88(4): pp. 516-541, 2000
6. Mange D., Stauffer A., Petraglio E., Tempesti G., Embryonic Machines that Divide and Differentiate, Proc. 1st Int. Workshop on Biologically Inspired Approaches to Advanced Information Technology (BioADIT04), pp. 328-343, 2004
7. Tempesti G., Mange D., Stauffer A., A robust multiplexer-based FPGA inspired by biological systems, Journal of Systems Architecture 43(10): pp. 719-733, 1997

8. Stauffer A., Mange D., Tempesti G., Teuscher C., A Self-Repairing and Self-Healing Electronic Watch: The BioWatch, Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware (ICES'2001), pp. 112-127, 2001
9. Sanchez E., Mange D., Sipper M., Tomassini M., Perez-Uribe A., Stauffer A., Phylogeny, Ontogeny, and Epigenesis: Three Sources of Biological Inspiration for Softening Hardware, Proceedings of the 1st International Conference on Evolvable Systems: From Biology to Hardware (ICES96), pp. 34-54, 1997
10. Sipper M., Sanchez E., Mange D., Tomassini M., Perez-Uribe A., A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems, IEEE Transaction on Evolutionary Computation 1(1): pp. 83-97, 1997
11. Moreno J.-M., Sanchez E.,Cabestany J., An in-system routing strategy for evolvable hardware programmable platforms, Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society, 2001
12. Lee C.Y., An Algorithm for Path Connections and Its Applications, IRE Transactions on Electronic Computers EC-10(3): pp. 346-365, 1961
13. Corporaal H., Microprocessor Architectures from VLIW to TTA, John Wiley & Sons, 1998
14. Corporaal H., Mulder H., MOVE: A framework for high-performance processor design, Proceedings of the International Conference on Supercomputing, pp. 692-701, 1991
15. Restrepo H.F., Tempesti G., Mange D., Implementation of a Self-replicating Universal Turing Machine, In Alan Turing: Life and Legacy of a Great Thinker, pp. 241-269, 2004
16. Restrepo H.F., Implementation of a Self-repairing Universal Turing Machine, Swiss Federal Institute of Technology (EPFL), PhD thesis 2457, 2001
17. Ibàñez J., Anabitarte D., Azpeitia I., Barrera O., Barrutieta A., Blanco H., Echarte F., Self-inspection based reproduction in cellular automata, Proceedings of the 3rd European Conference on Artificial Life (ECAL95), pp. 564-576, 1995
18. Laing R., Automaton models of reproduction by self-inspection, Journal of Theoretical Biology 66, pp. 437-456, 1977