Building a basic membrane computer

Alejandro Millan, Julian Viejo, Juan Quiros,
Manuel J. Bellido, David Guerrero, and Enrique Ostua

Grupo ID2 – Universidad de Sevilla (Spain)
Email: `amillan@us.es, julian@dte.us.es, jquiros@dte.us.es, bellido@dte.us.es, guerre@dte.us.es, ostua@dte.us.es`
`www.dte.us.es/id2`

Summary. In this work, we present the building of two well-known membrane computers (squares generator and divisor test). Although they are very basic machines they present problems common to every P system (competition, parallel execution of rules, membrane dissolution, etc.) that have to be solved in order to get real emulations for them. The presented designs mimic the systems operation in a realistic way, by achieving both maximum parallelism and non-determinism, and demonstrating for the first time that a membrane computer can actually be built *in silico*. Our architectures fully emulate the membranes behaviour yielding to a performance of one transition per clock cycle, supposing a real physical realization of the mentioned machines.

Key words: membrane computing, P system, digital circuit design, parallel computing, reconfigurable hardware, FPGA.

1 Introduction

Membrane computing was introduced in 2000 by Gheorghe Păun [12]. This topic is based on living cells and the first models are defined as an hierarchical structure of compartments (membranes), which contains objects (chemicals), which evolve according to applicability rules (chemical reactions). However, membrane computing has a very important problem: implementation. This kind of computing is extremely powerful but is a machine-oriented solution so current efforts in this way have been focused on improve the simulation of such system on both software and hardware platforms.

Several works exist on membrane computing from the hardware point of view. All of them have been focused on FPGA designing, trying to mimic the internal structure of P systems, in one way or another, into an electronic device.

Firstly, in [13], authors present a development in which the membranes of the system evolve in a parallel way although, internally, rules in each membrane are executed sequentially. Also, the general functioning of the system is deterministic: rules are applied by following a pre-established order.

Secondly, in [1], after some works on this topic [5, 6], authors achieve an architecture based on grouping rules into macro-rules and generating a new power set of macro-rules (each macro-rule is conformed by one or more of the original rules). In this way, they calculate this power set by including all the possible combinations of the original rules and, following a non-deterministic process, the architecture chooses one macro-rule and apply it in a maximal way. This process continues sequentially until none macro-rule can be applied. So, they achieve a certain degree of parallelism between the execution of the original rules.

Thirdly, in [10, 11], after a considerable amount of works [7, 8, 9], authors finally present two architectures that simulate P systems: the first one is focused on rules and the second one is focused on membranes. This second architecture mimics very well the structure of P systems but it suffers from important limitations: it allows systems in which competing rules (rules that consume the same objects) are prioritized what yields to deterministic systems only. Also, object selection is done sequentially so parallelism is set aside in this way.

Fourthly, in [15, 14], a very important advance on the topic is presented. These works cover the emulation of P systems on reconfigurable hardware, observing both parallelism (in terms of competing rules and object selection) and non-determinism. Authors present a development capable of automatically generate the hardware equivalent to a given P system that can compute each of its transitions in constant time (5 clock cycles). The problem with this approach is that the architecture supports min_1 transition mode only (rules can be applied one time maximum [3]).

In conclusion, although several authors have performed relevant efforts on the topic of electronic realization of P systems, until now, nobody has achieved an structure that mimics the fully parallel and non-deterministic behaviour of such machines.

Exactly in this aspect, our current work tries to contribute something relevant. We have adopted a very different point of view: membrane computing is a machine-oriented computational model so we think it is impossible to design a machine (or architecture) that solves all the problems. In the same way that we need to develop new algorithms to solve new problems in an algorithm-oriented computational model, we think it is necessary to design a new specific machine for each P system when needed. At least, while being important to observe the inherent features of this kind of systems: i.e. non-determinism and maximal parallelism. So, following this idea, we have started our work by trying to build the very first membrane computers we all know (Fig. 1 and Fig. 2: the ones presented in the foundational paper [12]).

The rest of the paper is organized as follows: in Sect. 2, we explain in detail the developed architectures and the designs employed to exactly emulate maximum parallelism and non-determinism of the chosen P systems, in Sect. 3, we present the operation results obtained by the built emulators, and finally in Sec. 4, we finish with the main conclusion of this work.

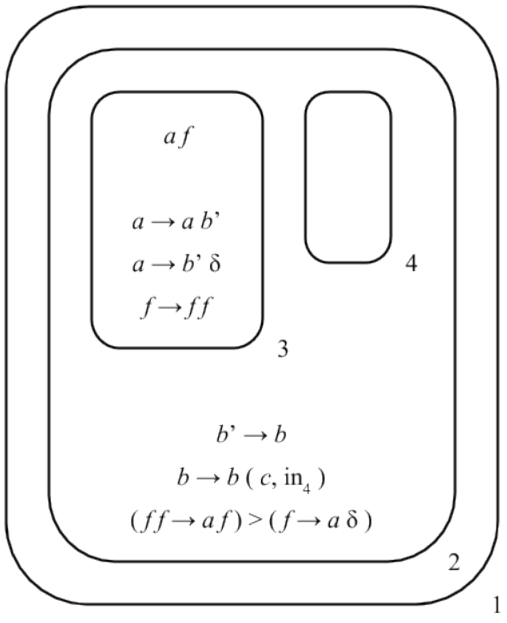

Fig. 1. Computer 1: n^2 generator ($n >= 1$).

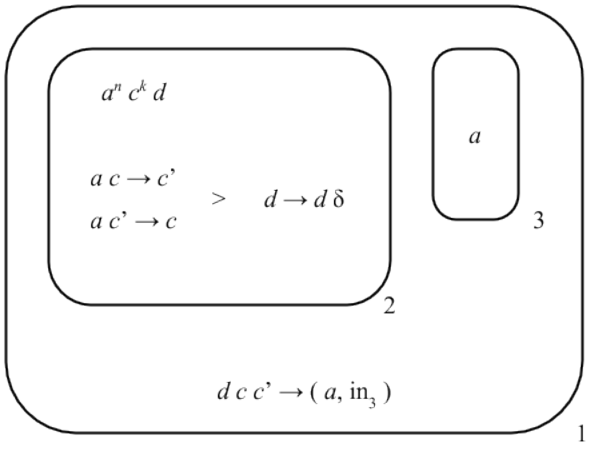

Fig. 2. Computer 2: Divisor test (Does k divide n?).

2 Design

With the exposed idea, we have developed architectures for the both mentioned P systems. However, there are common aspects that are interesting to be exposed firstly.

The main problem in the implementation of P systems is the application of rules, moreover when they are competing for common objects. So, the basic structure we have employed is shown in Fig. 3. It does not represent a rule corresponding to any of the chosen systems but illustrates how we have oriented the design of rules. In this figure, a very simple rule is shown: the REG block stores the amount of available f objects while the Logic block (associated to rule r_1) calculates how many times the rule has to be applied in order to consume all f objects. Finally, there is a feedback operation in which the object amount is adjusted according to rule applications. This construct allows the execution of the rule multiple times in a single clock cycle. From this basic case, we are going to show how we have addressed with competition in the chosen P systems.

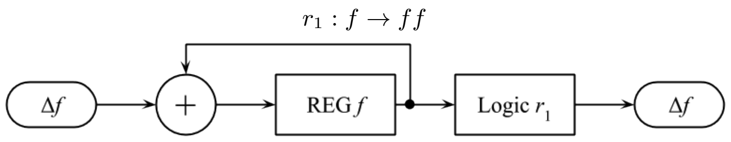

Fig. 3. Basic structure emulating object and rule.

Competition in Computer 1 (Fig. 1) comes from the two first rules in membrane 3. They consume a same object a. In this case, object a maintains its amount until the second rule is triggered (then membrane 3 is dissolved). So, in this system, we have included a random number generator (based on an maximum-length Galois linear-feedback shift register, LFSR [2, 4]) that ensures all possible executions are performed in a non-deterministic way. The LFSR determines the cycle in which the second rule is applied. Note that, being a maximum-length LFSR, it allows the system to go through all the possible executions without repeating anyone. The hardware structure corresponding to these rules is shown in Fig. 4.

Competition in Computer 2 (Fig. 2) comes from the two first rules in membrane 2. They consume a same object a and an specific object (c or c' in each case). In this membrane, it is necessary to distribute object a randomly between the two rules, also taking into account the amount of objects c and c' available in the system. The hardware employed to mimic this behaviour is presented in Fig. 5 and it is based on the following algorithm:

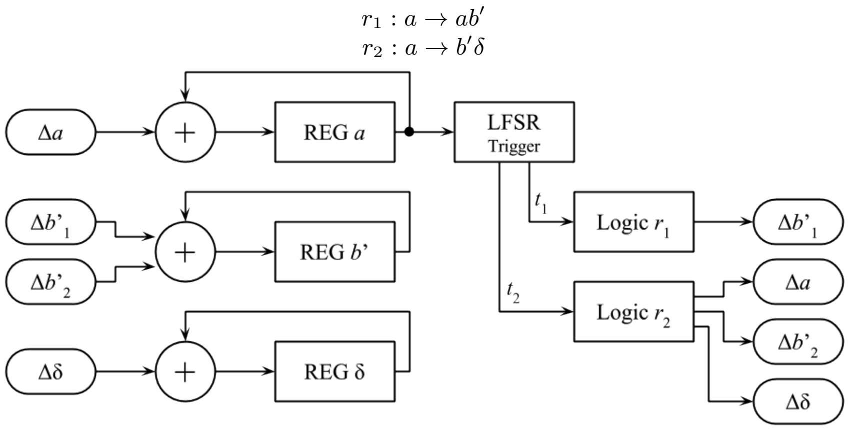

Fig. 4. Structure resolving competition case in Computer 1.

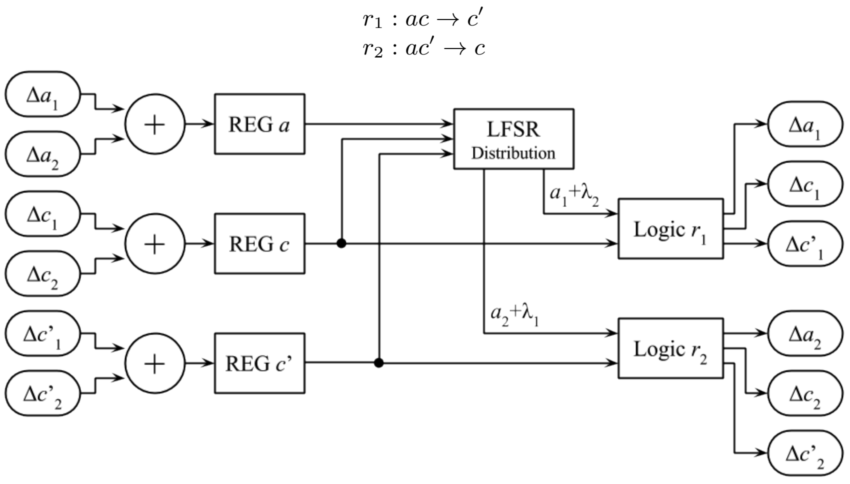

Fig. 5. Structure resolving competition case in Computer 2.

1. Randomly let:
   $a = a_1 + a_2$
2. Let:
   $\lambda_1 = max\{0; a_1 - c\}$, $\lambda_2 = max\{0; a_2 - c'\}$
   $\tau_1 = min\{c; a_1 + \lambda_2\}$, $\tau_2 = min\{c'; a_2 + \lambda_1\}$
3. Apply $r_1 \times \tau_1$ times and $r_2 \times \tau_2$ times.

It is important to denote that the algorithm describes only the idea under the design but the hardware structure calculates τ_1 and τ_2 in a combinational way (i.e. in a single clock cycle).

The rest of rules on both machines have been designed following the structure described in Fig. 3. They do not imply competition so they can be constructed in a direct way.

Finally, the machines interfaces are shown in Fig. 6 and 7. In the case of Computer 1, the Reset signal launch a new system computation (performing one transition per Clock cycle). The Master Reset and Seed inputs are employed at first only in order to initialize the LFSR by the user (the seed is obtained from the time past since the circuit was powered on; by using a 50 MHz counter). Then, the machine works continuously generating, in a non-deterministic way, all the squares existing in its computing range. Each time a square is produced, its value is presented through the Answer output bus and the Data Valid signal is activated (yielding to a new Reset and a new execution).

In the case of Computer 2, the Clock, Reset, Master Reset, and Seed signals work in a similar way to Computer 1. Also, Number n and Number k pass to the system the test inputs. Once the numbers have been tested, Data Valid is activated and the Answer signal indicates True or False as a response.

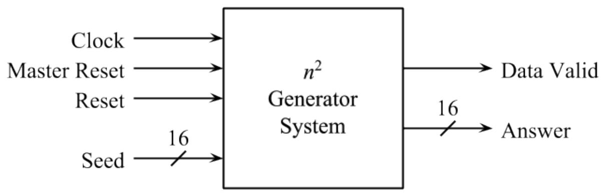

Fig. 6. Computer 1 interface.

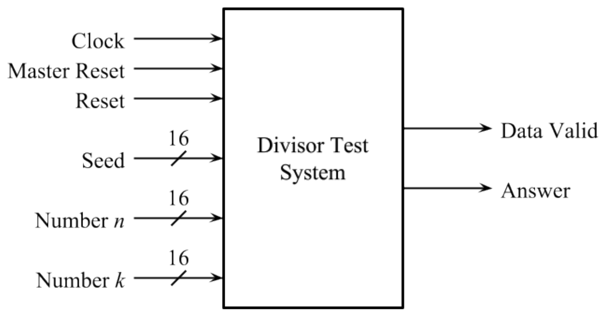

Fig. 7. Computer 2 interface.

3 Results

Both machines have been tested in both software and hardware ways. On the one hand, the software testing has been performed with Xilinx ISE v11.4. In Fig. 8 we present an operation detail of Computer 2 having $k = 21$ and $n = 63$. The machine starts having 63 a objects and 21 c objects inside membrane 2. In the first transition, rule r_1 is applied 21 times, giving us a new configuration without c objects and 42 a objects and 21 c' objects. In the second transition, rule r_2 is applied 21 times, giving us a new configuration without c' objects and 21 a objects and 21 c objects. In the third transition, rule r_1 is applied 21 times (again), giving us a new configuration without a and c objects and 21 c' objects. In the fourth transition neither rules r_1 nor r_2 can be applied, so membrane 2 is dissolved (rule r_3), giving us a new configuration with 21 c' objects and 1 d object inside membrane 3. Finally, rule r_4 can not be applied because the lack of c objects inside the membrane, so system is halted. At this moment, DV (data valid) signal activates and the Answer output indicates True (k divides n). As we have mentioned previously, each transition is processed in a single clock cycle.

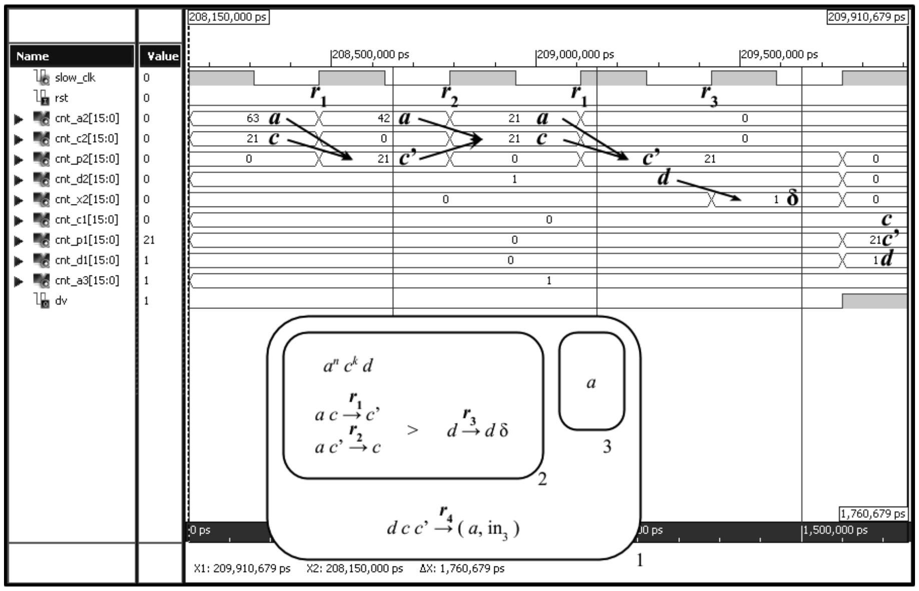

Fig. 8. Simulation detail of Computer 2 testing if $k = 21$ divides $n = 63$.

On the other hand, the machines have been tested on hardware by programming them into both Digilent Basys 2 training boards equipped with Spartan 3E-1200 FPGAs.

Operation of Computer 1 board (Fig. 9) is started by pressing the Master Reset button what feeds a random seed to the LFSR (also lightning on the Master

Reset indicator). Then, the machine starts computing executions of the P system, yielding the calculated answers to the 4-digit 7-segment display. These executions are carried out in a non-deterministic order but they cover all the possible combinations and generate all squares existing between 1^2 and 63^2. Once all squares are shown, the machine halts and the Finish indicator is lighted on. For demonstration purposes only, the board clock frequency has been reduced to a human-visible one (ca. 25 Hz) and can be observed through the Clock indicator.

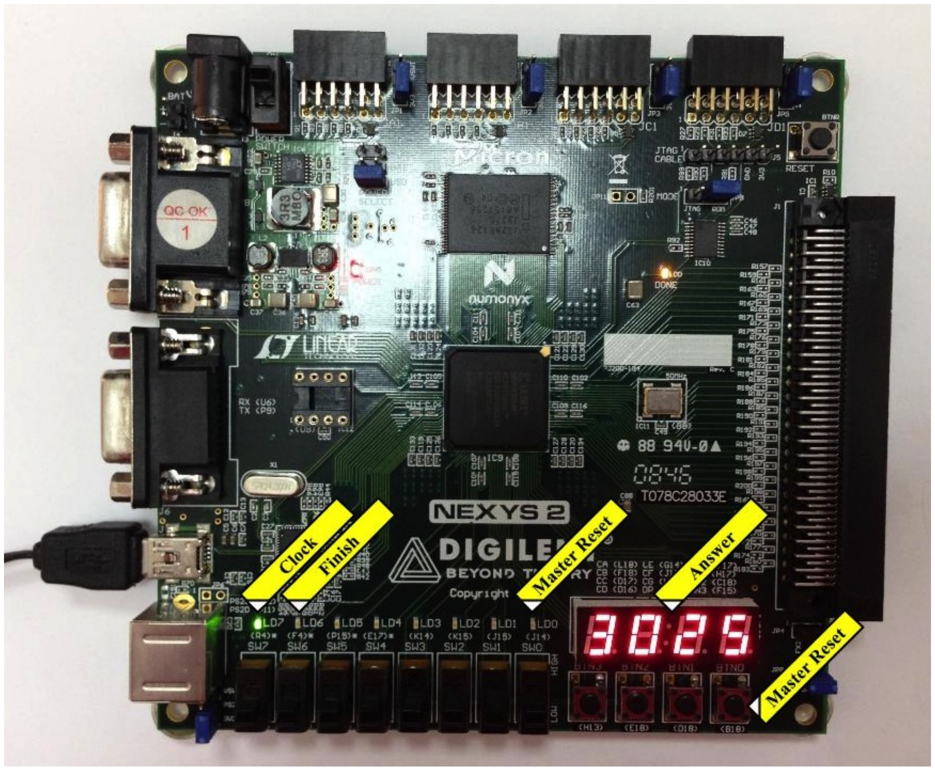

Fig. 9. Computer 1 training board emulator: Clock indicator (LD7), Finish indicator (LD6), Master Reset indicator (LD0), Answer (7-segment display), and Master Reset button (BTN0).

In a similar way, operation of Computer 2 board (Fig. 10) is also started by pressing the Master Reset button what feeds a random seed to the LFSRs present in the circuit. Then, the machine starts computing executions of the P system (each time a execution finishes the Data Valid indicator is lighted on). The executions are fed with random inputs and the machine shows them continuously on the 7-segment display (Numbers n and k). In order to facilitate humans to understand results, the board pauses execution when the division test is successful during enough time to read the display. Also for demonstration purposes only, the board clock frequency has been reduced to a human-visible one (ca. 25 Hz) and can be observed through the Clock indicator.

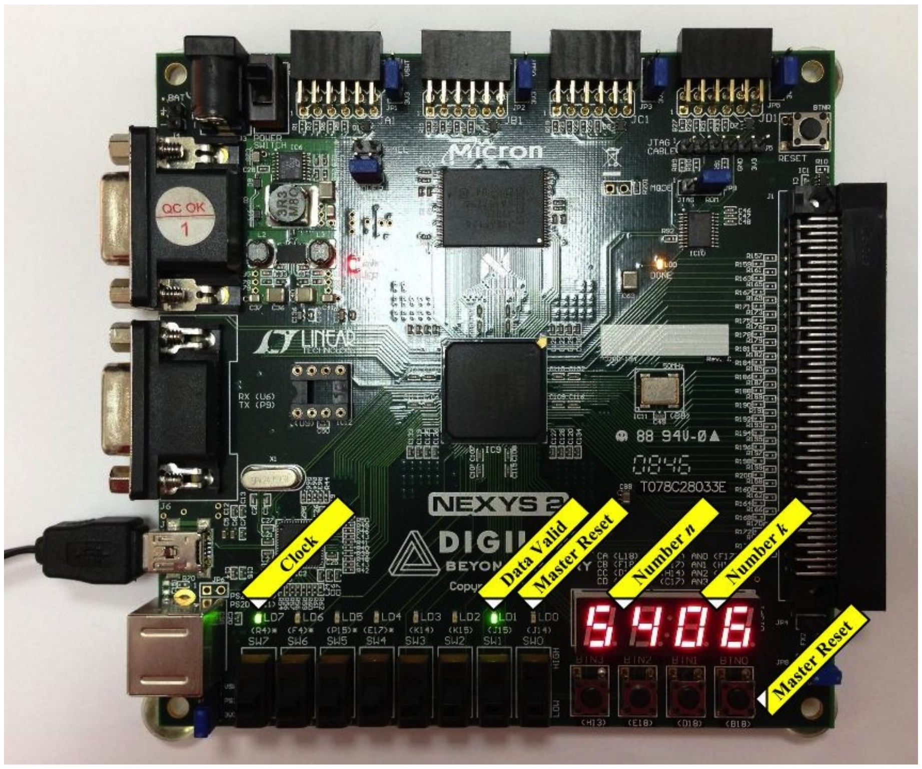

Fig. 10. Computer 2 training board emulator: Clock indicator (LD07), Data Valid indicator (LD1), Master Reset indicator (LD0), Number n (first two digits of 7-segment display), Number k (last two digits of 7-segment display), and Master Reset button (BTN0).

The hardware details are presented in Table 1 and the device utilization is shown for both cases in Table 2 and Table 3 respectively. As we can observe, both computers consume a small portion of the available resources showing themselves as a very efficient design also in terms of hardware needs. Also, it is noticeable that register width has an important impact on resource utilization: 64-bit in the case of Computer 1 what yields to a maximum amount of ca. 10^{19} objects of each type (register values are of signed type). These register widths has been chosen according to the 4-digit displays available on the training boards.

	Computer 1	Computer 2
Target Device	xc3s1200e-4fg320	
Clock Frequency	50 MHz	
Performance	50 Mtransition/s	
Register Width	64-bit	16-bit

Table 1. Hardware details.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	625	17344	3%
Number of 4 input LUTs	2434	17344	14%
Number of occupied Slices	1345	8672	15%
- Number of Slices containing only related logic	1345	1345	100%
- Number of Slices containing unrelated logic	0	1345	0%
Total Number of 4 input LUTs	2479	17344	14%
- Number used as logic	2434		
- Number used as route-thru	45		
Number of bonded IOBs	18	250	7%
- IOB Flip Flops	2		
Number of BUFGMUXs	2	24	8%
Number of MULT18X18SIOs	10	28	35%
Average Fanout of Non-Clock Nets	2.27		

Table 2. Device utilization for Computer 1.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	269	17344	1%
Number of 4 input LUTs	842	17344	4%
Number of occupied Slices	503	8672	5%
- Number of Slices containing only related logic	503	503	100%
- Number of Slices containing unrelated logic	0	503	0%
Total Number of 4 input LUTs	958	17344	5%
- Number used as logic	842		
- Number used as route-thru	116		
Number of bonded IOBs	19	250	7%
- IOB Flip Flops	1		
Number of BUFGMUXs	2	24	8%
Number of MULT18X18SIOs	4	28	14%
Average Fanout of Non-Clock Nets	3.25		

Table 3. Device utilization for Computer 2.

4 Conclusion

In this work, we have shown that it is possible to fully emulate a P system without loosing its intrinsic features of maximal parallelism and non-determinism. With that aim, machines presented in the foundational paper of the discipline (square generator and divisor test) have been built. Our designs mimic the internal structure of the P systems allowing the resulting hardware to perform as the theoretical system should: processing one transition per clock cycle. The systems evolve in a non-deterministic way and rules are applied in a maximal parallel derivation mode; what, to the best of our knowledge, supposes the first real emulation of a P system *in-silico*.

References

1. Alonso, S., Fernandez, L., Arroyo, F., Gil, J.: A Circuit Implementing Massive Parallelism in Transition P Systems. International Journal "Information Technologies and Knowledge", vol. 2, pp. 35–42 (2008).
2. Bonde, V., Kale, A.: Design and Implementation of a Random Number Generator on FPGA. International Journal of Science and Research, vol. 4, no. 5, pp. 203–208 (2015).
3. Freund, R., Ibarra, O., Paun, A., Sosik, P., Yen H.: Catalytic P Systems. In "The Oxford Handbook of Membrane Computing", pp. 83–117, Oxford University Press (2009). ISBN:9780199556670
4. George, M., Alfke, P.: Linear Feedback Shift Registers in Virtex Devices. Xilinx Application Note, XAPP210 v1.3 (2007).
5. Martinez, V., Fernandez, L., Arroyo, F., Garcia, I.: A HW circuit for the application of Active Rules in a Transition P System Region. Proc. 4th International Conference Information Research and Applications, Varna (Bulgary), pp. 80–87 (2006). ISBN-10: 954-16-0036-0.
6. Martinez, V., Fernandez, L., Arroyo, F., Guiterrez, A.: HW Implementation of a Bounded Algorithm for Application of Rules in a Transition P-System. Proc. 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara (Romania), pp. 32-38 (2006).
7. Nguyen, V., Kearney, D., Gioiosa, G.: Balancing performance, flexibility and scalability in a parallel computing platform for membrane computing applications. Lecture Notes in Computer Science, vol. 4860, pp. 385–413 (2007). DOI:10.1007/978-3-540-77312-2_24
8. Nguyen, V., Kearney, D., Gioiosa, G.: An implementation of membrane computing using reconfigurable hardware. Computing and Informatics, vol. 27, no. 3+, pp. 551569 (2008).
9. Nguyen, V., Kearney, D., Gioiosa, G.: An Algorithm for Non-deterministic Object Distribution in P Systems and Its Implementation in Hardware. Lecture Notes in Computer Science, vol. 5391, pp. 325–354 (2009). DOI:10.1007/978-3-540-95885-7_24
10. Nguyen, V., Kearney, D., Gioiosa, G.: An extensible, maintainable and elegant approach to hardware source code generation in Reconfig-P. The Journal of Logic and Algebraic Programming, vol. 79, no. 6, pp. 383-396 (2010). DOI:10.1016/j.jlap.2010.03.013

11. Nguyen, V., Kearney, D., Gioiosa, G.: A Region-Oriented Hardware Implementation for Membrane Computing Applications. Lecture Notes in Computer Science, vol. 5957, pp. 385–409 (2010). DOI:10.1007/978-3-642-11467-0_27
12. Paun, G.: Computing with Membranes. Journal of Computer and System Sciences, vol. 61, pp. 108–143 (2000). DOI:10.1006jcss.1999.1693
13. Petreska, B., Teuscher, C.: A Reconfigurable Hardware Membrane System. Lecture Notes in Computer Science, vol. 2933, pp. 269–285 (2004). DOI:10.1007/978-3-540-24619-0_20
14. Quiros, J., Millan, A., Viejo, J.: *Implementacion sobre hardware reconfigurable de una arquitectura no determinista, paralela y distribuida de alto rendimiento, basada en modelos de computacion con membranas* (Implementation on reconfigurable hardware of a non-deterministic, parallel, and distributed high performance architecture based on membrane computing models). PhD Thesis (2016).
15. Verlan, S., Quiros, J.: Fast Hardware Implementations of P Systems. Lecture Notes in Computer Science, vol. 7762, pp. 404–423 (2013). DOI:10.1007/978-3-642-36751-9_27