



**HAL**  
open science

## FPGA Implementation of Numerical P Systems

Zeyi Shang, Sergey Verlan, Gexiang Zhang, Haina Rong

► **To cite this version:**

Zeyi Shang, Sergey Verlan, Gexiang Zhang, Haina Rong. FPGA Implementation of Numerical P Systems. International Journal of Unconventional Computing, 2021, 16 (2-3), pp.279-302. hal-04032275

**HAL Id: hal-04032275**

**<https://hal.science/hal-04032275v1>**

Submitted on 25 Apr 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

# FPGA Implementation of Numerical P Systems

ZEYI SHANG<sup>1,2</sup>, SERGEY VERLAN<sup>2</sup>, GEXIANG ZHANG<sup>1,3\*</sup>

<sup>1</sup> School of Electrical Engineering, Southwest Jiaotong University, Chengdu  
611756, China

<sup>2</sup> Univ Paris Est Creteil, LACL, F-94010 Creteil, France

<sup>3</sup> College of Information Science and Technology, Chengdu University of  
Technology, Chengdu 610059, China

Numerical P Systems (NPS) is a variant of P systems using real-valued quantities. It was shown that it can successfully be applied for different real-world problems, in particular in the area of robotic control. In this paper we introduce an extension of NPS, called Generalized Numerical P Systems (GNPS) and we describe an efficient implementation of GNPS using FPGA hardware. This allows to build fast controller chips based on (G)NPS and interacting directly with the environment. Two test cases are presented describing the implementation results of Sobel image edge detection algorithm.

*Key words:* Membrane Computing, numerical P system, enzymatic numerical P system, generalized numerical P system, Field Programmable Gate Array (FPGA), hardware implementation.

## 1 INTRODUCTION

Membrane computing (MC) is a natural computation paradigm abstracted from structural and functional features of living cells [29, 24]. Computing models in MC are called *P Systems*. There is an intense theoretic research in the area of MC focusing primarily on computability aspects, in particular on Turing-computability, see [15] for an overview. There is a huge number of

---

\* ✉ Corresponding author: Gexiang Zhang, E-mail: zhgxtdylan@126.com

27 different variants of P systems, we cite some important variants below: cell-  
28 like P systems [29, 16], tissue/population-like P systems [19, 1, 8], spiking  
29 neural P systems [13, 10, 14] and P systems with active membranes [30, 12,  
30 36].

31 Because of the biological background of P systems, they are used as mod-  
32 eling frameworks for biological and ecological systems, see [5, 18, 20, 23]  
33 for more details. As for engineering application of P systems, it is a relatively  
34 new direction in its early stage comparing to biological/ecological modeling  
35 of P systems. Complex market interactions are modeled by population dy-  
36 namic P systems in [34]. Several variants of spiking neural P systems are  
37 used in power system fault diagnosis [37, 32, 27, 45, 33]. Other applications  
38 can be found in [46, 43, 44].

39 Numerical P systems introduced in [31] is a variant of P systems very  
40 different from the standard model. Instead of objects and rewriting rules it  
41 features real-valued variables which are updated in discrete time steps using  
42 a set of equations. This particularity, long time unexplored, is very interesting  
43 for applications in the area of control theory, as usually a control is described  
44 using a set of differential equations, which in many cases can be translated to  
45 the NPS equation set. In order to be applicable to practical case studies NPS  
46 model was extended to Enzymatic Numerical P systems (ENPS) [25] that al-  
47 lowed much more complex behaviors in the corresponding equations. This  
48 in turn lead to the concept of membrane controllers [2] that are ENPS sys-  
49 tems designed to act as controllers and running in some environment. As  
50 a test bed the control of differential wheeled robots [2] (motion, obstacle  
51 avoidance, wall following, robot following) was developed. In [39] a kine-  
52 matic controller and a proportional-integral-derivative controller are designed  
53 for wheeled mobile robots. Another interesting application is found in [38]  
54 where an environment classifier and a novel multi-behaviors control approach  
55 are proposed to enhance the reactive navigation performance of autonomous  
56 mobile robots.

57 Membrane controllers mentioned above require several ingredients. The  
58 evolution of a robot in a real or simulated environment requires a program  
59 that reads/transmits the values of robot' sensors (usually distance and speed),  
60 runs the simulation of the controller (given in ENPS form) for one or several  
61 steps and then updates/transmits the values of actuators (usually robot wheel  
62 motors). Before running the simulation this program should assign initial  
63 values for the membrane controller and after the simulation it should retrieve  
64 corresponding output values from it. The controller itself is simulated using  
65 a custom simulator [2, 7] or by Matlab code [39, 38]. In some cases [7, 38]

66 the experiments were carried out in real robot environments. To speed-up  
67 the simulation in the case of complex controllers [40] the use of graphical  
68 processing units (GPU) hardware architectures was proposed [11].

69 In this article we extend NPS to a new model that we call Generalized Nu-  
70 merical P systems (GNPS). The main idea behind this extension is to provide  
71 a theoretical background allowing us to build custom parallel hardware archi-  
72 tectures using Field-Programmable Gate Array (FPGA) technology. We stud-  
73 ied what operations can be efficiently performed in hardware and restricted  
74 the GNPS architecture to be a series of rules that imply that the dynamics of  
75 the system is described using equations written in Presburger arithmetic. This  
76 allows in turn a very efficient translation to a hardware description language  
77 (HDL) used for FPGA circuit design, allowing to run the model at the clock  
78 speed ( $10^8$  steps/s) and using a low number of resources. To assist this trans-  
79 lation we developed a compiler that translates GNPS to Verilog HDL. This  
80 allows to simplify the design process and to rapidly develop real hardware  
81 prototypes. Moreover, it turns out that there is a tight link between GNPS,  
82 sequential circuits [35], Mealy/Moore automata [21, 22] and synchronous  
83 programming languages like Esterel and Lustre.

84 We considered several test cases and in this paper we present two of them,  
85 describing implementation results of Sobel image edge detection algorithm  
86 on Diligent Basis 3 FPGA board, based on Xilinx Artix 7 architecture.

87 This paper is organized as follows: Section 2 introduces the definition of  
88 NPS and ENPS, analyzing their relations with systems of difference equa-  
89 tions, together with the definition of a normal form of (E)NPS. Section 3 ex-  
90 tends the original definition of NPS to GNPS. Section 4 discusses the FPGA  
91 implementation of two case studies in detail. Conclusions are drawn in Sec-  
92 tion 5.

## 2 DEFINITIONS

### 93 2.1 Numerical P Systems

94 Numerical P systems were introduced in [31] as a model for the study of  
95 economical processes. They have a tree-like structure and each compartment  
96 contains a set of real-valued variables as well as evolution rules, called pro-  
97 grams. They are formally defined as follows.

98 A numerical P system is the construct

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \dots, (Var_m, Pr_m, Var_m(0)))$$

99 where  $m > 0$  is the degree of the system,  $H$  is a set of labels,  $\mu$  is a mem-  
 100 brane structure,  $Var_i$ ,  $Pr_i$  and  $Var_i(0)$  are the set of variables, programs and  
 101 initial values from compartment  $i$ ,  $1 \leq i \leq m$ . By convention, we will label  
 102 variables with two indices such that  $Var_j = \{x_{1j}, \dots, x_{k_jj}\}$ .  
 103 A program (rule)  $P_{li} \in PR_i$  has the following form

$$P_{li} : F_{li}(x_{1i}, \dots, x_{k_ii}) \rightarrow c_{l1}|v_1 + \dots + c_{ln_i}|v_{n_i}$$

104 where variables  $v_1, \dots, v_{n_i}$  belong to membrane  $i$ , or to the neighboring ones  
 105 (the parent or the children of  $i$ ).

106 The first part of the rule (function  $F$ ) is called the production function,  
 107 while the second part (at the right-hand-side of the arrow) is called the repar-  
 108 tition protocol.

109 A rule is applied as follows [31]. First the value of the production function  
 110 is computed, based on current values of the variables. Second, each vari-  
 111 able  $v_s$ ,  $1 \leq s \leq n_i$  from the repartition protocol part receives the fraction  
 112  $\frac{c_{ls}}{\sum_{t=1}^{n_i} c_{lt}}$  of the computed production function value. If several rules update  
 113 the same variable, then the corresponding amounts are added. Finally, the  
 114 value of a variable at the beginning of each new step is reset to 0 if this vari-  
 115 able was used in a computation of some production function.

116 It is not very difficult to observe that a computation in a NPS corresponds  
 117 to a discrete time series, where the value of a variable at some time step is  
 118 a function of the values of several variables at the previous time step. More  
 119 precisely the evolution of the system can be described by the following equa-  
 120 tions:

$$x_{ji}(t) = \sum_{P_{lk} \text{ has } x_{ji} \text{ in rhs as } v_s} F_{lk}(x_{1k}(t), \dots, x_{r_kk}(t)) \frac{c_{ls}}{\sum_{r=1}^{n_k} c_{lr}} + \bar{x}_{ji}(t) \quad (1)$$

121 where  $\bar{x}_{ji}(t) = \begin{cases} x_{ji}(t) & \text{if } x_{ji} \text{ does not appear in any production function } F_{lk}, \\ 0 & \text{otherwise.} \end{cases}$

**Example 2.1.** Consider the following NPS, also depicted in Figure 1, with  
 two membranes nested as follows:  $[1[2]2]_1$ . Let  $Var_1 = \{a, b, f\}$ ,  $Var_2 =$   
 $\{x, y\}$ ,  $Var_1(0) = (0, 1, 3)$ ,  $Var_2(0) = (0, 1)$ . The rules of the system are  
 defined as follows:

$$Pr_{11} : 4(a + b) \rightarrow 1|a + 1|f + 2|x.$$

$$Pr_{12} : 3(x + y) \rightarrow 1|b + 1|x + 1|y.$$

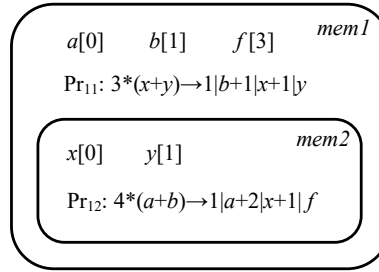


FIGURE 1  
 Numerical P system from Example 2.1. The nested (membrane) structure is represented by a Venn diagram; the variables and the rules are placed in corresponding locations; the initial value of variables follow them in square brackets.

122 It is not difficult to observe that the corresponding system can be rewritten  
 123 as following time series with initial conditions  $a(0) = 0, b(0) = 1, f(0) = 3,$   
 124  $x(0) = 0, y(0) = 1.$

$$\begin{cases} a(t+1) = a(t) + b(t) \\ b(t+1) = x(t) + y(t) \\ f(t+1) = f(t) + a(t) + b(t) \\ x(t+1) = x(t) + y(t) + 2(a(t) + b(t)) \\ y(t+1) = x(t) + y(t) \end{cases} \quad (2)$$

125 This can be also written in a matrix form as follows

$$\begin{bmatrix} a \\ b \\ f \\ x \\ y \end{bmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{bmatrix} a \\ b \\ f \\ x \\ y \end{bmatrix} \quad (3)$$

In many cases systems of recurrences can be solved analytically using standard methods. The analytical solution for system defined by Equation (2) is given below:

$$\begin{aligned} a(t) &= 2 \times 3^{t-2}, & b(t) &= 4 \times 3^{t-2}, & f(t) &= 3^{t-1} + 3, \\ x(t) &= 8 \times 3^{t-2}, & y(t) &= 2 \times 3^{t-2} + 1, & t &> 1 \end{aligned}$$

126 We would like to notice that by definition it is allowed to have several  
 127 rules in a membrane. In this case, at each step, one of them is chosen and  
 128 applied non-deterministically. However, for practical considerations, most of  
 129 the systems considered in the literature have only one rule per membrane.

130 *Enzymatic Numerical P systems*

131 Enzymatic numerical P systems (ENPS), introduced in [25] are an extension  
 132 of NPS that adds a new type of rules (called enzymatic):

$$P_{li} : F_{li}(x_{1i}, \dots, x_{k_i i})(e \rightarrow) c_{l1} | v_1 + \dots + c_{ln_i} | v_{n_i}$$

The application of this rule is conditioned to the verification of the minimality condition between the values of  $e$  and  $x_{1i}, \dots, x_{k_i i}$ . Unfortunately, there is no unique definition for this condition – several papers use different ones. Here is the list of most used conditions:

$$e > \min(c(x_{1i}), \dots, c(x_{k_i i})), \text{ in [25]}$$

$$e > \min(x_{1i}, \dots, x_{k_i i}), \text{ in [25]}$$

$$e > \min(|x_{1i}|, \dots, |x_{k_i i}|), \text{ in [26].}$$

133 The first definition above uses the function  $c(x)$ , which is the concentration  
 134 of  $x$  in the biological sense. For example, for a production function  $2x +$   
 135  $y$  we obtain  $c(x) = x/2$ ,  $c(y) = y$ , see [25] for more details. As in the  
 136 case of NPS, if there are several applicable rules, one of them is used non-  
 137 deterministically. One can observe that ENPS also yield time series that use  
 138 a conditional operator.

**Example 2.2.** Consider the following ENPS, also depicted in Figure 2, with three membranes nested as follows:  $[_1[_2[_3]_3]_2]_1$ . Let  $Var_1 = \{a, b\}$ ,  $Var_2 = \{E\}$ ,  $Var_3 = \{c\}$ ,  $Var_1(0) = (0, 3)$ ,  $Var_2(0) = (2n)$ ,  $n > 0$ ,  $Var_3(0) = 0$ . The set of rules is defined as follows:

$$Pr_1 : 2b + 1(E \rightarrow) 1 | a,$$

$$Pr_2 : E - 1 \rightarrow 1 | E,$$

$$Pr_3 : 2(c + 1)(E \rightarrow) 1 | c + 1 | b.$$

It is not difficult to observe that the corresponding system can be rewritten

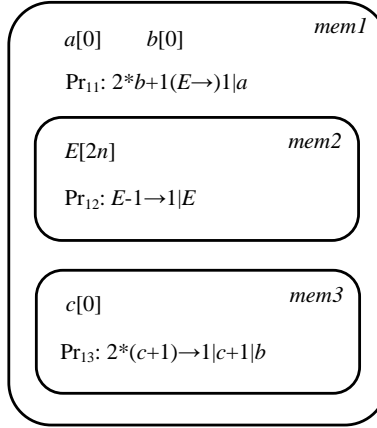


FIGURE 2  
Enzymatic numerical P system from Example 2.2.

as following time series.

$$\begin{aligned}
 a(t+1) &= \text{if } E(t) > b(t) \text{ then } 2b(t) + 1 \text{ else } a(t) \\
 b(t+1) &= c(t) + 1 \\
 c(t+1) &= c(t) + 1 \\
 E(t+1) &= E(t) - 1 \\
 a(0) &= 0, \quad b(0) = 0, \quad c(0) = 0, \quad E(0) = 2n
 \end{aligned}$$

139 Clearly, we obtain that  $a(t) = \sum_{i=0}^t 2i + 1 = t^2$  (for  $t \leq n$ ).

140 **2.2 Binary normal forms for (E)NPS**

141 Traditionally, NPS evolve in  $min_1$  mode (from each membrane a single pro-  
 142 gram is selected and applied), see [9]. In [17] so called all-parallel mode is  
 143 introduced for (E)NPS, when all applicable rules from a membrane are ap-  
 144 plied at the same step (we remark that contrary to traditional P systems there  
 145 is no competition for symbols/variable values). The advantage of such a mode  
 146 is that the computation becomes deterministic. Another advantage is that the  
 147 system allows a very nice binary normal form.

**Definition 2.1.** A (E)NPS is said to be in the binary normal form if all rules



are of form

$$P : F(x_1, \dots, x_n) \rightarrow c|x_k + L|\lambda, \text{ or}$$

$$P : F(x_1, \dots, x_n)(E \rightarrow)c|x_k + L|\lambda,$$

148 for some  $n, k > 0$ ;  $c, L \geq 0$  and where  $\lambda$  is a special dummy variable.

149 It is relatively easy to see that any (E)NPS working in all-parallel mode can  
150 be transformed to an equivalent one in the binary normal form. To achieve  
151 this, any rule/program

$$F(x_{1j}, \dots, x_{kj}) \rightarrow c_1|v_1 + \dots + c_l|v_l$$

152 can be replaced by  $l$  rules that assign the same proportion of  $F$  to each vari-  
153 able  $x_l$

$$F(x_{1j}, \dots, x_{kj}) \rightarrow c_s|v_s + (K - c_s)|\lambda, 1 \leq s \leq l, K = \sum_{p=1}^l c_p.$$

154 We remark that when  $l = 1$ , then  $L = K - c_s = 0$ , so there is no variable  
155  $\lambda$  introduced in the rule.

156 A similar transformation can be done with enzymatic programs.

157 Moreover, if we relax the condition that variables of the production func-  
158 tion should be from the same membrane, then it is possible to obtain a stronger  
159 result. This allows to combine all rules related to a single variable into one  
160 rule by choosing appropriate coefficients. In this case several programs of  
161 type

$$F_n(x_{1j}, \dots, x_{kj}) \rightarrow c_n|x + z_n|\lambda$$

162 can be combined as

$$\sum_{p=1}^n \frac{c_p}{c_p + z_p} F_p(x_{1j}, \dots, x_{kj}) \rightarrow 1|x. \quad (4)$$

163 (E)NPS having only rules of the above type is said to be in *unary normal*  
164 *form*. It is easy to observe that Equation (4) corresponds to the following time  
165 series

$$x(t+1) = \sum_{p=1}^n \frac{c_p}{c_p + z_p} F_p(x_{1j}(t), \dots, x_{kj}(t)) + \bar{x}(t). \quad (5)$$

166 which is exactly the Equation (1).

### 3 GENERALIZED NUMERICAL P SYSTEMS

167 In this section we introduce a generalization of (E)NPS, called Generalized  
168 Numerical P System (GNPS), which has some interesting properties helpful  
169 for the hardware implementation of the model. As a starting point we take the  
170 notion of the membrane controller [2], hence from the beginning we assume  
171 to be interested not in the final result of the computation, but in the dynamics  
172 of the model. This naturally leads to the inclusion of the concept of dedicated  
173 input and output variables. The functioning of the system supposes that input  
174 variables are read-only and can be updated by an external entity at each step.  
175 The output variables are write-only and an external entity may use their values  
176 at each step. Such a definition allows to effectively build controllers based on  
177 GNPS, without using any additional tools or mechanisms to pass the values  
178 and start/stop the computation.

179 From the structural point of view we use a structural abstraction interme-  
180 diate between a tree-based structure and a flattened system (more precisely  
181 a hypergraph structure), being the equivalent of the network of cells [9] in  
182 NPS. This allows to have the notion of the locality (useful for hardware im-  
183 plementation as it can trigger the use of neighbor cells), but does not impose  
184 the strong restriction of a tree structure — some examples of membrane con-  
185 trollers spend an enormous amount of time for the data propagation because  
186 of the imposed tree structure. Concretely, this allows production functions  
187 to contain variables defined in a different membrane and also the repartition  
188 protocol may involve variables from any combination of membranes.

189 The main difference of GNPS with respect to previous models is a new  
190 type of rules that generalize all previous ones. This comes from the obser-  
191 vation that rules of ENPS are rather limited (and also have a poorly defined  
192 semantics). The ENPS simulator PeP [6] already proposed to use some sim-  
193 ple arithmetic predicates to control the applicability of the rules. With GNPS  
194 we propose to go further and to use conditional rules of form (we separate by  
195 a semicolon local variables from the other ones):

$$P(x_1, \dots, x_k; E_1, \dots, E_m); F(x_1, \dots, x_k) \rightarrow c_1|v_1, \dots, c_n|v_n, \quad (6)$$

196 where  $P$  is predicate in Presburger arithmetic (we recall that this is the first-  
197 order theory of the natural numbers with addition, i.e. one can use compar-  
198 isons, Boolean operations, additions, subtractions and constant multiplica-  
199 tions in expressions). Moreover, in the basic variant of the definition we will  
200 restrict production functions  $F$  to be Presburger as well. However, in order to  
201 accommodate real-case scenarios we will allow the usage of a finite algebraic

signature (a set of functions that can be used in addition to the operations in Presburger arithmetic) for both production functions and predicates.

**Example 3.1.** We can consider the following predicate for a rule

$$P(x, y, z; E, F) = E > x \wedge (F > y * 2 + 3 * z).$$

If we consider an algebraic signature containing the ordinary multiplication operation ( $\sigma = \{\times\}$ ), then it would be possible to write the following predicate

$$P(x, y, z; E, F) = E > x \wedge ((F > y * 2 + 3 * z) \vee (E + F > x \times y + z)).$$

Finally, in order to obtain a deterministic evolution of the system, we assume that GNPS works in all-parallel mode, i.e. all applicable rules are applied at each step. This allows to greatly simplify the design of the hardware implementations.

Formally, we define a GNPS as the following tuple

$$\Pi = (m, I, O, (Var_1, Var_1(0)), \dots, (Var_m, Var_m(0)), Pr, \sigma),$$

where  $m$ ,  $Var_i$  and  $Var_i(0)$  have the same meaning as in NPS (the number of cells/membranes, the vectors of internal variables and their initial values). The rules are no more specific to some membrane, so they are all collected in the set  $Pr$ . Each rule is of form (6). In the case of an always true predicate, it can be omitted. Used variables in each rule induce a dependency hypergraph. When this hypergraph is a tree, we may use a Venn diagram notation and place rules in corresponding cells/membranes. The input (resp. output) variables are given by the set  $I$  (resp.  $O$ ). The algebraic signature  $\sigma$  contains the list of additional functions used (with respect to the addition/subtraction and constant multiplication). If  $\sigma = \emptyset$  then it may be omitted from the definition.

We recall that the evolution of the system is performed in all-parallel mode, i.e. all applicable rules are applied in parallel at each step.

#### 4 FPGA IMPLEMENTATION

According to the discussion in Section 2.2 and the fact that Presburger arithmetic is recursive, any GNPS system can be rewritten as the following time series (where  $X(t)$ ,  $Y(t)$  and  $Q(t)$  are the vectors of input, output and internal variables, respectively, at time  $t$ ):

$$Q(t + 1) = F(Q(t), X(t)) \tag{7}$$

$$Y(t + 1) = G(Q(t), X(t)) \tag{8}$$

225 These equations are the generalization (using real numbers instead of Boolean  
226 values) of equations used in switching algebra [35] for the definition of the  
227 concept of Mealy automaton [21], which together with Moore automaton [22]  
228 form the basis of modern synchronous circuit design. Since from the imple-  
229 mentation point of view real numbers should be encoded using a fixed bit size,  
230 it appears that GNPS are very similar to vector Moore/Mealy machines. This  
231 in turn allows a straight implementation using hardware FPGA technology.

232 For implementation efficiency we considered following restrictions for  
233 GNPS:

- 234 • Real values are replaced by their approximation using a fixed-point bi-  
235 nary representation.
- 236 • The production functions are linear.
- 237 • The predicates are Presburger-definable.

238 The above restrictions allow to relatively easy obtain the Mealy/Moore  
239 machine in form of Equation (7). The corresponding functions  $F$  and  $G$  are  
240 linear enriched with conditional statements as it is shown in Example 4.1.

241 *Remark 4.1.* In the basic case we consider an empty signature  $\sigma$  as this al-  
242 lows a straight translation to Verilog. For more complex computations, corre-  
243 sponding functions should be implemented additionally as Verilog modules.  
244 This can induce a delay, as in many cases it is not possible to compute corre-  
245 sponding functions in one clock step.

246 *Remark 4.2.* In the case of fixed-point encoding, it is possible to easily im-  
247 plement the multiplication operation working in one time step. This can be  
248 done either directly (by using multiplication code dependent on the width of  
249 the encoding), or using a special component of FPGA called *DSP slice* that  
250 allows to perform multiplication operations in one step (up to 48-bit width).

251 *Remark 4.3.* Contrary to multiplication, it is not easy to implement the divi-  
252 sion operation in one time step. However, the division by a constant  $c$  can be  
253 seen as the multiplication by  $c^{-1}$ .

254 **Example 4.1.** Consider the system depicted on Fig. 3. It has two input, two  
255 output and two internal variables. The system computes the average value of  
256 its inputs and also indicates if this value changed by more than 0.1% on the  
257 previous step. We recall that all rules are executed in parallel.

The set of equations corresponding to this system is the following ( $a(0) =$

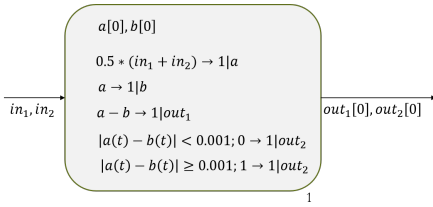


FIGURE 3  
P system from Example 4.1.

$$b(0) = out_1(0) = out_2(0) = 0.$$

$$a(t+1) = \frac{in_1(t) + in_2(t)}{2}$$

$$b(t+1) = a(t)$$

$$out_1(t+1) = a(t) - b(t)$$

$$out_2(t+1) = \text{if } |a(t) - b(t)| < 0.001 \text{ then } 0 \text{ else } 1$$

258 It can be directly transformed to Verilog as follows (we assume a fixed point  
259 encoding of real numbers over 32 bits and using 12 bits for the fractional part).  
260 In the below listing the fixed-point (constant) multiplication is performed by  
261 the function `_mult` (recall that 2048 is 0.5 in the chosen fixed-point encod-  
262 ing).

```
263 module A #(parameter WIDTH = 32, parameter BPPOS = 12)
264 (output [WIDTH:0] out1, output [WIDTH:0] out2,
265 input [WIDTH:0] in1, input [WIDTH:0] in2,
266 input clk
267 );
268 reg [WIDTH:0] a = 0;
269 reg [WIDTH:0] b = 0;
270 reg [WIDTH:0] c = 0;
271 reg [WIDTH:0] d = 0;
272
273 always @(posedge clk) begin
274   a <= _mult(in1+in2, 2048);
275   b <= a;
276   out1 <= a-b;
277   out2 <= a-b < 4 && a-b > -4 ? 0 : 4096;
278 end
279 endmodule
280
281
```

282 The synthesis of this circuit using Vivado tools uses 87 logic cells.

283 It can be seen that the translation is rather straightforward. A compiler  
284 `FPNtoVerilog` was developed in order to assist in this translation. As in-  
285 put it takes the GNPS model in form of Equations (7) and (5) and produces

286 as output behavioral Verilog code implementing the corresponding Mealy/  
287 Moore automaton.

288 The compiler performs the following steps:

- 289 1. Parse the input file.
- 290 2. Identify input and output symbols.
- 291 3. Flatten the obtained system.
- 292 4. Perform constant propagation.
- 293 5. Convert all constants to fixed-point real number representation.
- 294 6. Write Verilog output.

295 These steps are performed using standard compiling techniques. The last  
296 step is straightforward as a sequential Boolean (switching) function/circuit  
297 can be directly translated to Verilog. As a result a file containing the syn-  
298 thesizable (in FPGA) Verilog module whose code simulates each step of the  
299 GNPS at each clock tick is generated. Two case studies are designed as target  
300 models and their FPGA implementation process is detailed to elucidate how  
301 a GNPS can be implemented in a FPGA. Since the algorithm we implement  
302 is making use of the square root function, we consider that the signature of  
303 the system is  $\sigma = \{\sqrt{\cdot}\}$ .

304 Our target development board is Digilent BASYS 3 equipped with a Xilinx  
305 Artix-7 XC7A35T-1CPG236C FPGA as core component. The FPGA devel-  
306 oping environment is Xilinx Vivado 2019.1 and Verilog is used as HDL. A  
307 Dell *Latitude* outfitted with a Intel Core i7-7820HQ and 16 GB RAM is the  
308 host computer.

#### 309 **4.1 Case study 1**

310 The GNPS model (called GNPS1 for simplicity) of case study 1 is illustrated  
311 in Figure 4. It corresponds to Sobel image edge detection algorithm. GNPS1  
312 only has a skin membrane, without any inner membranes. A program is ap-  
313 plicable if its conditional rule can be met. Variable  $e$  is assigned a big enough  
314 value so that the 4 programs can be executed at the same time. We employ  
315 fixed-point number format to represent real values. Specifically, every vari-  
316 able is assigned a 20-bit register in which the first bit designates the sign  
317 bit, the following 8-bit denoting integer part and the rest of 11-bit presenting  
318 fraction part of a real number. We also use the signature  $\sigma = \{x^2, \sqrt{\cdot}\}$ .

<i>Skin</i>			
$a_1[6.7]$	$a_{21}[5.03]$	$a_{31}[4.31]$	
$a_{41}[2.28]$	$a_{51}[1.92]$	$a_{61}[0.85]$	$e[7]$
$a_2[2.43]$	$a_{22}[1.71]$	$a_3[0.94]$	$\theta[0.2]$
$a_4[-3.07]$	$a_{24}[5.46]$	$b_1[0]$	$b_2[0]$
$b_3[0]$	$b_{31}[0]$	$b_4[0]$	$b_{41}[0]$
$\left\{ \begin{array}{l} P_{11} = e > \min(a_1, a_{21}, a_{31}, a_{41}, a_{51}, a_{61}) \\ Pr_{11} : a_1 + 2a_{21} + a_{31} - a_{41} - 2a_{51} - a_{61} \rightarrow 1 \mid b_1 \end{array} \right.$			
$\left\{ \begin{array}{l} P_{21} = e > \min(a_2, a_{22}) \\ Pr_{21} : \sqrt{a_2^2 + a_{22}^2} \rightarrow 1 \mid b_2 \end{array} \right.$			
$\left\{ \begin{array}{l} P_{31} = e > a_3 \\ Pr_{31} : 2 * (a_3 - \theta) \rightarrow 1 \mid b_3 + 1 \mid b_{31} \end{array} \right.$			
$\left\{ \begin{array}{l} P_{41} = e > \min(a_4, a_{24}) \\ Pr_{41} : a_4 + 2a_{24} \rightarrow 1 \mid b_{41} + 1 \mid b_4 \end{array} \right.$			

FIGURE 4

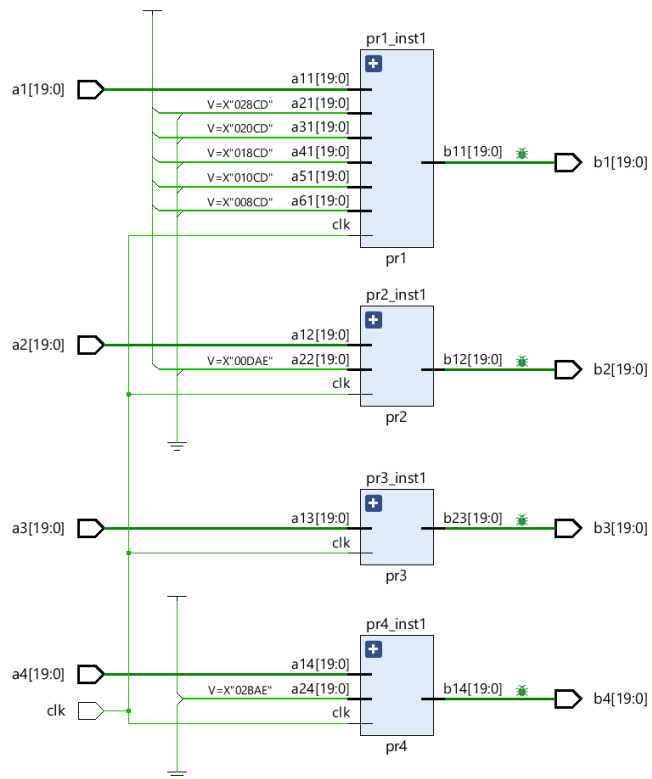
GNPS model for case study 1. It implements the core computations of Sobel image edge detection algorithm. GNPS1 has a skin membrane containing 4 programs and 19 real-value variables. The predicate for each program is taken to a separate line before it. All the programs execute in parallel if their conditional rules are satisfied. Input variables are  $a_1, a_2, a_3, a_4$  (highlighted in red). Output variables are  $b_1, b_2, b_3, b_4$  (highlighted in blue). Others are intermediate variables.

319 After inputting GNPS1 to our `FPNtoVerilog` compiler, the output is  
320 a behavioral model that specifies the behavior of GNPS1. We also provided  
321 the implementations of the functions from the signature  $\sigma$  (square root and the  
322 square function). Next, this model is translated to register transfer level (RTL)  
323 using Vivado tools. The upper-most level of GNPS1 schematic generated by  
324 Vivado is depicted in Figure 5. Before proceeding to the next processing  
325 step, RTL model should be evaluated to verify that it dose possess required  
326 functions/behaviors. A test file named *testbench* is designed to perform such  
327 verification work. In the *testbench*, a model should be instantiated at first.  
328 Then input data/impulse to its input ports of the model and analyze outcomes  
329 represented in the form of timing waveforms. It is notable that outcomes  
330 are simulated by software (Vivado for our case, this simulation process is  
331 called *Behavioral Simulation* in Vivado.) instead of FPGA. In this research,  
332 two case study models are constructed as sequential circuits, namely clock is  
333 involved as a metronome to synchronize operations. Rising edge of clock is  
334 the trigger signal, i.e., operations can only be carried out after a rising edge  
335 and variables hold their values until next rising edge. The period of clock is  
336 set to 10 ns in the test bench.

337 Behavioral simulation omits any gate delays and data path delays, which  
338 means that results are output instantaneously, at the same time of trigger edge  
339 for sequential circuits [3] and the change instant of signals for combinational  
340 circuits. GNPS1's behavioral simulation timing waveform is shown in Figure  
341 6, from which we can see that  $b_1$ ,  $b_3$ ,  $b_4$  get their outcomes after the first ris-  
342 ing edge of clock. While the result of  $b_2$  arises after the second rising edge,  
343 for the sum of two operands should be computed in the first period of clock  
344 and during that time the initial value for variable to be radicated is zero. Here  
345 we resort to corresponding Vivado IP core to compute the square root [42].  
346 Apparently, behavioral simulation suggests that four programs of GNPS1 ex-  
347 ecute simultaneously and output variables emit results 20 ns later. As stated  
348 above, behavioral simulation neglects any delays so the waveform cannot re-  
349 flect the real timing situation. Post implementation timing simulation which  
350 can only be conducted after *implementation* operation can provide more reli-  
351 able timing waveform.

352 As the platform for design & validation of prototype circuits [4], the essen-  
353 tial task of FPGA implementation is to obtain circuits. *Synthesis* procedure  
354 bridges the gap between RTL models and circuits. Models can be synthesized  
355 after behavioral simulation if it behaves as expected behaviors. There are two  
356 important tasks should be done after synthesizing target model: setting con-  
357 straints and debug cores. Constraints include timing constraints and physical





**FIGURE 5**  
 Block diagram of GNPS1 RTL model. After the design of a model, the corresponding schematic can be drawn automatically by Vivado. The schematic characterizes the same functions/behaviors as RTL model representing by HDL.

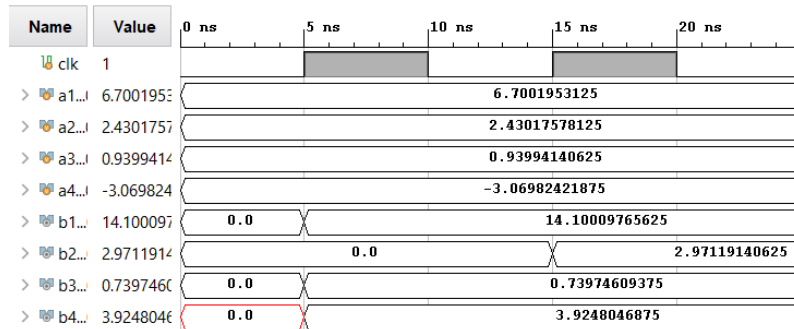


FIGURE 6 Behavioral simulation of GNPS1. The first rising edge of clock is at 5 ns.  $b_2$  gets its result at 15 ns and other three variables get their results at 5 ns since all the delays are neglected. This is not the reality but a simplification for assessing the behaviors of target model.

358 constraints. In timing constraints, the period of the clock and input/output  
 359 delays are set, while physical constraints specify I/O configurations, mapping  
 360 ports of model to pins of FPGA. The clock period is set 10 ns in the two case  
 361 studies. To save pins, only  $b_3$  and  $b_4$  are set as output ports for GNPS1. All  
 362 the constraints are written in constraint file (.xdc) in the format of industry  
 363 standard Synopsys Design Constraints (SDC) [41]. The variables to be debug-  
 364 gged in hardware debug procedure can be marked and set after synthesis.

365 The subsequent procedure of *Synthesis* is *Implementation*, which performs  
 366 plan & route of the synthesized circuits and other vital operations such as  
 367 power and hardware resource consumption analysis, real timing analysis. As  
 368 mentioned above, behavior simulation ignores any delays so the timing situ-  
 369 ation cannot be evaluated from its waveform. Nevertheless, gate delays and  
 370 data path delays of a model are taken into account after implementation so  
 371 the timing of a design can be revealed by post implementation timing simula-  
 372 tion, as shown in Figure 7. According to design timing summary provide by  
 373 Vivado, the worst negative slack (WNS) is 3.97 ns, worst hold slack (WHS)  
 374 0.058 ns and worst pulse width slack (WPWS) 3.75 ns.

375 For the sake of comparing the computing speed of FPGA hardened GNPS  
 376 and software simulation of GNPS, speedup is defined as the ratio of elapsed  
 377 time of two methods. A software called *PeP* which dose not have a GUI dedi-

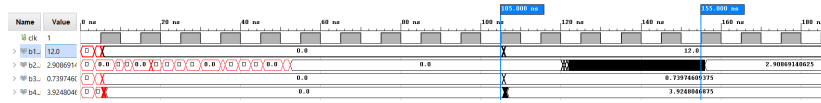


FIGURE 7

Post implementation timing simulation of GNPS1. Port  $b_1$ ,  $b_3$  and  $b_4$  obtain their steady output value after the eleventh rising edge of clock, indicating it costs 110 ns to get results. For  $b_1$ , its steady output value emerges after sixteenth rising edge, costing 160 ns to compute outcome.

```
INFO:Simulation finished succesfully after 1 steps and 0.005651 seconds; End state below:
num_ps = {
  Skin:
    var = { a1: 0.00, a21: 0.00, a31: 0.00, a41: 0.00, a51: 0.00, a61: 0.00, a2: 0.00, a22: 0.00, a3: 0.00, a4:
0.00, a24: 0.00, b1: 14.10, b2: 2.97, b3: 0.74, b31: 0.74, b4: 3.92, b41: 3.92, theta: 0.00, }
  E = { e: 7.00, }
}
```

FIGURE 8

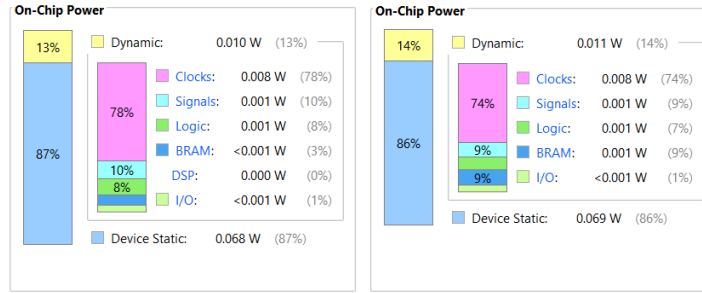
Software simulation of GNPS1. It is assumed that GNPS1 evolves one step to stop. Here we can see that there is no one-to-one correspondence between a clock cycle and a GNPS step. For complex arithmetic computations, one step of GNPS requires more than one clock cycle.

378 cates to emulate (E)NPS [6]. For the sake of simulating other types P systems,  
379 one can resort to P-Lingua [28]. GNPS1 is transformed to its ENPS counter-  
380 part and emulated by *PeP*, which outputs results and elapsed time to com-  
381 pute the results, as shown in Figure 8. Then the speedup of FPGA hardened  
382 GNPS1 is calculated in Equation 9. The maximum error of output variables  
383 is computed in Equation 10.

$$\frac{5.651 \times 10^6}{160} = 3.5319 \times 10^4 \quad (9)$$

$$\left| \frac{3.92 - 3.92480469}{3.92} \right| \times 100\% \approx 1.225765 \times 10^{-3} \quad (10)$$

384 The estimated power consumption is reported after implementation, shown  
385 in Figure 9 (a). Because the function of GNPS1 and GNPS2 is not complicate,  
386 the dynamic power merely shares 13% to 14% of total power and clock power  
387 makes up more than 70% of dynamic power.



(a) Power consumption of GNPS1 is 0.078 w. (b) Power consumption of GNPS2 is 0.08 w.

FIGURE 9

Total power consumption is the sum of device static power and dynamic power. Power consumption of the two cases are nearly the same, although GNPS1 works in all parallel and GNPS2 works in sequential.

388 After implementation of a model, its *Place & Route* planning is also per-  
 389 formed. Vivado provides powerful view check feature so the place and route  
 390 can be zoomed in to see each path clearly. Hardware resource cost for GNPS1  
 391 is listed in Table 1.

392 If design objectives are satisfied after implementation, one can proceed  
 393 to *generate bitstream* which contains design specifications and *program de-*  
 394 *vice* that download bitstream to FPGA to carry out physical *plan & route*.  
 395 The real computing results of FPGA cannot be observed straightforwardly,  
 396 but requires a particular procedure called *hardware debug*. The variables to  
 397 be debugged should be marked in Verilog codes or marked in the net list.  
 398 We mark  $b_3$  and  $b_4$  in Verilog codes as debug signals. After programing de-  
 399 vice, the integrated logic analyzer window open automatically. Debug signals  
 400 should be added into the window manually, then run debug to exhibit values  
 401 computed by FPGA, shown in Figure 10.

## 402 4.2 Case study 2

403 In practical applications such as image processing and robot path planning,  
 404 computation process comprises several sequential procedures. In each pro-  
 405 cedure, multiple functions may be performed in all parallel mode, like the  
 406 way GPNS1 works. In this subsection, we modify GNPS1 from all parallel

Resource	Used	Available	Utilization %
Slice	680	8150	8.34
LUT	1309	20800	6.29
LUTRAM	159	9600	1.66
FF	2126	41600	5.11
BRAM	1.5	50	3.00
IO	41	106	38.68
BUFG	2	32	6.25

TABLE 1  
Hardware resources utilization of GNPS1

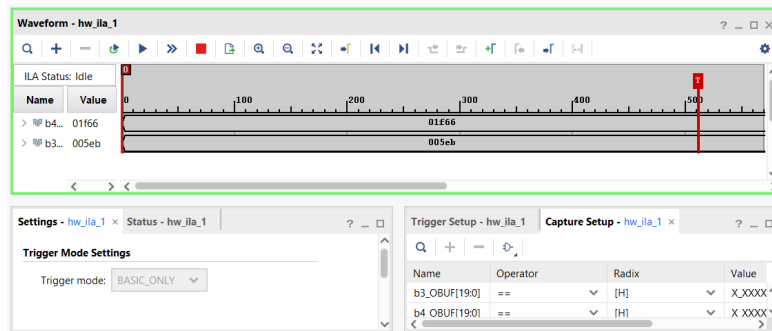


FIGURE 10  
Hardware debug of GNPS1. Input variables cannot be debugged so there is no clock signal. Values are represented in hexadecimal, 01f66 is 8038 in decimal.  $8038 \div 2^{11} = 3.9248046875$ , which is the value of  $b_4$ . 005eb is 1515 in decimal,  $1515 \div 2^{11} = 0.73974609375$ , which is the value of  $b_3$ .

407 to sequential mode, resulting GNPS2, depicted in Figure 11. Conditional rule  
 408 of membrane 1 is met at beginning, so program  $Pr_{11}$  and  $Pr_{21}$  take place  
 409 simultaneously. Other conditional rules are not met for the initial value of  
 410 conditional variables are zeros. After  $Pr_{21}$  modifying the value of  $e_2$  from 0  
 411 to 3, programs in membrane 2 are triggered to execute. So does membrane  
 412 3 and 4. It is worth to note that  $Pr_{21}$  and  $Pr_{22}$  consume  $e_1$  and  $e_2$  so each  
 413 program can only execute once. In short, a train-type ignition method is used  
 414 to control the execution sequence of programs.

415 Input GNPS2 to `FPNtoVerilog` obtaining the RTL model. The block  
 416 diagram of GNPS2 RTL model is illustrated in Figure 12. Edge detection  
 417 is used as the trigger signal to impel the next membrane to execute. By this  
 418 way, the train-type ignition is realized. According to Figure 7 reflecting actual  
 419 timing condition of each program, program  $Pr_{11}$ ,  $Pr_{13}$ ,  $Pr_{14}$  cost 11 clock  
 420 cycles to compute while  $Pr_{12}$  needs 16 clock cycles. The testbench designed  
 421 for GNPS2 conforms to these delays and the behavioral simulation is shown  
 422 in Figure 13, in which variable *cout1* is the edge detection signal that can  
 423 detect the rising edge of *cont*. Be ware that  $b_i$  ( $i = 1, 2, 3, 4$ ) gets value  
 424 instantly for the neglect of delays in behavioral simulation.

425 Synthesize GNPS2 and open the synthesized model, complete constraints  
 426 design and debug core set as that of GNPS1, then implement GNPS2. Run  
 427 post implementation timing simulation to check the timing situation, as shown  
 428 in Figure 14. *PeP* simulation of GNPS2 shown in Figure 15, so the speedup  
 429 of FPGA implementation is computed in Equation 11. Power consumption  
 430 of GPNS2 is given in Figure 9 (b).

$$\frac{9.306 \times 10^6}{480} \approx 1.9388 \times 10^4 \quad (11)$$

431 Hardware consumption for GNPS2 is summarized in Table 2. It can be  
 432 seen that GNPS2 utilizes a little bit more resources than GNPS1 for its more  
 433 complicate logic. At last, perform hardware debug to verify that FPGA hard-  
 434 ened GNPS2 obtained correct outcomes, shown in Figure 16.

435 Finally, the last tests were performed using an autonomous execution of  
 436 the system without output and using distributed read-only memory data stor-  
 437 age for the input. Under this setup the speed of 100Mhz (corresponding to  
 438 the system clock) was achieved. This means that a GNPS model can be sim-  
 439 ulated at a speed of  $10^8$  steps per second. We would like to remark that in  
 440 real-use cases the reaction speed will be dependent on the input/output del-  
 441 ay. It is pointed out that the input/output circuits are not system-specific and  
 442 can be reused for different simulations. However, at the present state they

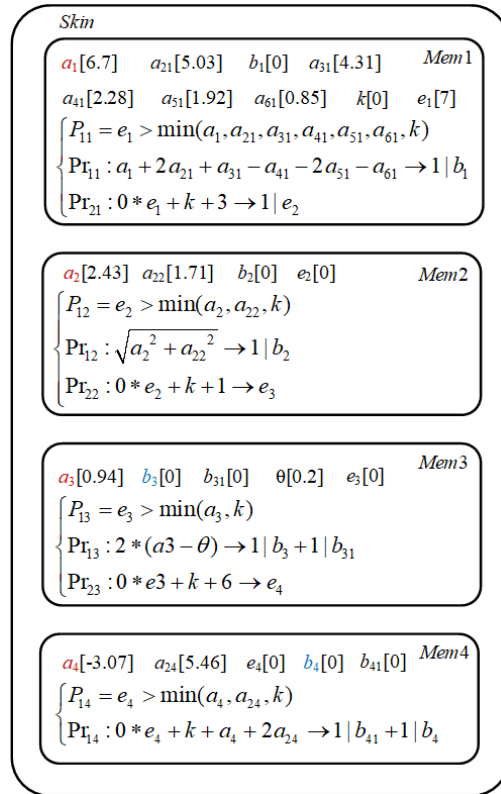


FIGURE 11

GNPS model for case study 2 is numbered as GNPS2. The equations inside are the core computations of Sobel image edge detection algorithm. GNPS2 has 5 membranes and evolves 4 steps to reach halt condition. Programs in each membrane compute concurrently while each membrane execute serially.

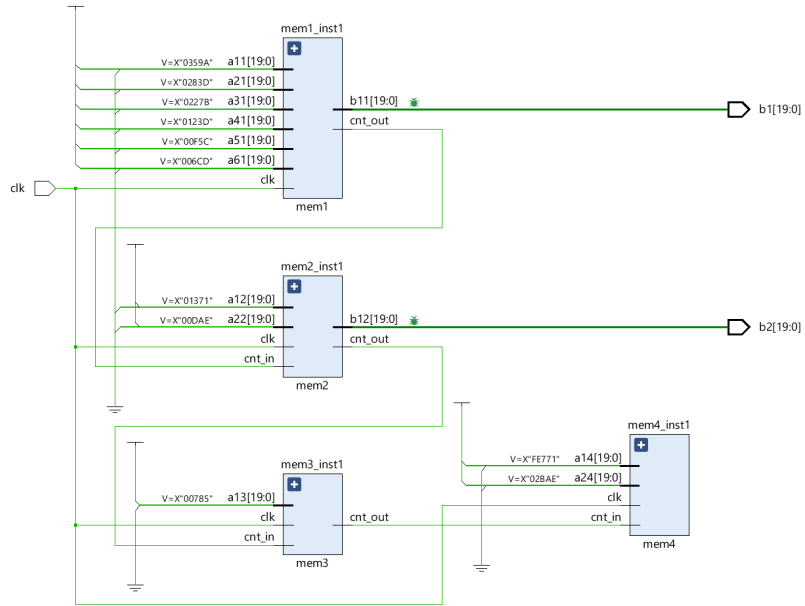


FIGURE 12  
Block diagram of GNPS2 RTL model. Each membrane is modeled in Verilog basic functional unit, *module*. The bug icons indicate variables to be debugged in *Hardware Debug* procedure.

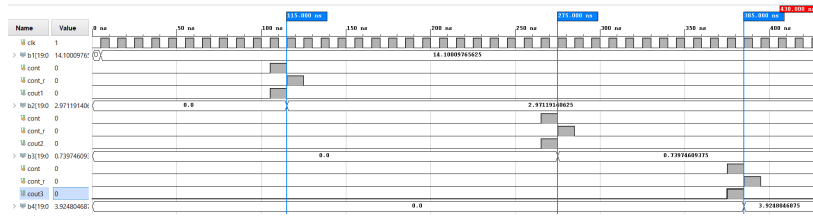
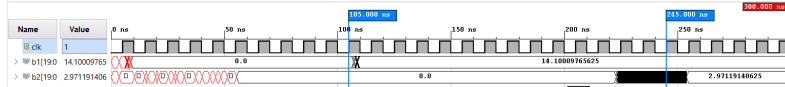
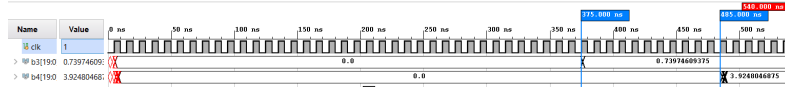


FIGURE 13  
Behavioral simulation of GNPS2. At a rising edge of clock, if  $cout_i = 1$ , execute programs in the next membrane. Three edge detection signal appears at 105 ns, 265 ns and 375 ns respectively and lasts 10 ns.





(a) Variable  $b_1$  gets its value in the 11th cycle, while  $b_2$  obtains its value in the 25th cycle.



(b) Variable  $b_3$  gets its value in the 38th cycle, while  $b_4$  obtains its value in the 49th cycle.

FIGURE 14

Post implementation timing simulation of GNPS2. The real timing of  $b_2$  and  $b_3$  is a little different than expected.

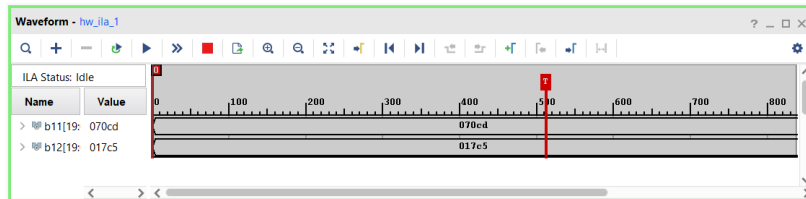
```
INFO:Simulation finished succesfully after 4 steps and 0.009306 seconds; End state below:
num_ps = {
  Skin:
    var = {}
    E = {}
  Mem1:
    var = { a1: 0.00, a21: 0.00, a31: 0.00, a41: 0.00, a51: 0.00, a61: 0.00, k: 0.00, b1: 14.10, }
    E = { e1: 0.00, }
  Mem2:
    var = { a2: 0.00, a22: 0.00, b2: 2.97, }
    E = { e2: 0.00, }
  Mem3:
    var = { a3: 0.00, b3: 0.74, b31: 0.74, }
    E = { e3: 0.00, }
  Mem4:
    var = { a4: 0.00, a24: 0.00, b4: 3.92, b41: 3.92, }
    E = { e4: 0.00, }
}
```

FIGURE 15

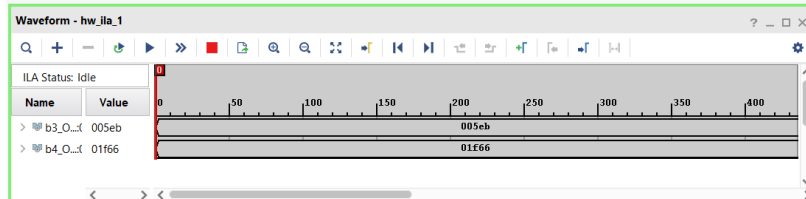
Software simulation of GNPS2. It is assumed that GNPS1 evolves four steps to stop. CPU of the host computer costs 0.009306 s to get results.

Resource	Used	Available	Utilization %
Slice	744	8150	9.13
LUT	1327	20800	6.38
LUTRAM	156	9600	1.63
FF	2150	41600	5.17
BRAM	1.50	50	3.00
IO	41	106	38.68
BUFG	2	32	6.25

TABLE 2  
Hardware resources utilization of GNPS2



(a) Hardware debug of  $b_1$  (connect to  $b_{11}$ ) and  $b_2$  (connect to  $b_{12}$ ).



(b) Hardware debug of  $b_3$  and  $b_4$ .

FIGURE 16  
Hardware debug of GNPS2. Values are represented in hexadecimal, 070cd is 28877 in decimal.  $28877 \div 2^{11} = 14.10009765625$ , which is the value of  $b_1$ . 17c5 is 6085 in decimal,  $6085 \div 2^{11} = 2.97119140625$ , which is the value of  $b_2$ .

443 need to be integrated manually in the final hardware design. At this moment,  
444 the development of `FPNtoVerilog` continues in order to integrate the au-  
445 tomatic generation of input/output modules. This will allow a generation of  
446 a hardware circuit directly from the GNPS specification, without any user  
447 intervention.

## 5 CONCLUSIONS

448 In this work we provide an efficient custom FPGA-based hardware architec-  
449 ture design for the implementation of generalized numerical P systems. This  
450 allows a high-speed simulation of the corresponding system as well as a di-  
451 rect on-chip handling of input and output data. This architecture has a solid  
452 theoretical basis based on GNPS systems and consumes a low number of  
453 resources. However, at the moment a manual intervention during the design  
454 process is still necessary and we concentrate our future effort on the complete  
455 authorization of the design. In perspective, this will allow to create custom  
456 chips performing a control function and handling their input/output in an au-  
457 tomated manner, without having any hardware programming knowledge.

458 Another important point is to test the obtained chips by using them to  
459 directly perform the robot control. Input/output from serial ports, leds and  
460 switches on the BASYS 3 board were tested to build data transmission chan-  
461 nels. We have made several preliminary investigations showing the possi-  
462 bility to acquire sensor data and to send control signals to the Pioneer 3  
463 DX wheeled robot from the GNPS controller. A further development would  
464 need to design the corresponding interface and accompanying functions in the  
465 compiler. We have also shown that the underlying model is extremely close  
466 to well known circuit design abstractions based on Mealy/Moore automata.  
467 This can give a new research direction investigating the links between GNPS  
468 and synchronous programming languages.

469 The method for FPGA implementation of GNPS is proposed and veri-  
470 fied. Results show that it is feasible to implement GNPS on FPGA to exploit  
471 their parallelism to speedup computations. Comparing to software simula-  
472 tion of GNPS, the speedup achieved is an order of  $10^4$ . Obviously, the hard-  
473 ware architecture of GNPS is a parallel one. However, the fact that programs  
474 and membranes worked as processing units to handle tasks inspires us that a  
475 FPGA hardened GNPS seems to be a heterogeneous multicore processor for  
476 different programs and membranes working in parallel have different func-  
477 tions. As a consequence, this heterogeneous architecture defined by P systems  
478 differing from current architectures can be applied in computation intensive

479 fields such as image/video processing, robot path planning, big data, etc.

## ACKNOWLEDGMENTS

480 This work is supported by the National Natural Science Foundation of China  
481 (61972324, 61672437, 61702428), by Beijing Advanced Innovation Center  
482 for Intelligent Robots and Systems (2019IRS14), Artificial Intelligence Key  
483 Laboratory of Sichuan Province (2019RYJ06) and the Sichuan Science and  
484 Technology Program (2018GZDZX0043, 2018GZ0185, 2018GZ0086).

## REFERENCES

- 485 [1] Francesco Bernardini and Marian Gheorghe. (2004). Population P systems. *Journal of*  
486 *Universal Computer Science*, 10(5):509–539.
- 487 [2] Cătălin Buiu, Cristian Ioan Vasile, and Octavian Arsene. (2012). Development of mem-  
488 brane controllers for mobile robots. *Information Sciences*, 187:33–51.
- 489 [3] Joseph Cavanagh. (2016). *Sequential Logic and Verilog HDL Fundamentals*. CRC Press.
- 490 [4] Pong P. Chu. (2008). *FPGA Prototyping by Verilog Examples*. John Wiley & Sons, Inc.
- 491 [5] Gabriel Ciobanu, Mario J. Pérez-Jiménez, and Gheorghe Puaun, editors. (2006). *Applica-*  
492 *tions of Membrane Computing*. Natural Computing Series. Springer.
- 493 [6] Andrei George Florea and Cătălin Buiu. GitHub—PeP: (Enzymatic) Numerical P System  
494 simulator. <https://github.com/andrei91ro/pep>.
- 495 [7] Andrei George Florea and Cătălin Buiu. (June 2017). Modelling multi-robot interactions  
496 using a generic controller based on numerical p systems and ros. In *2017 9th International*  
497 *Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6.
- 498 [8] Rudolf Freund, Gheorghe Păun, and Mario J. Pérez-Jiménez. (2005). Tissue P systems  
499 with channel states. *Theoretical Computer Science*, 330(1):101–116.
- 500 [9] Rudolf Freund and Sergey Verlan. (2007). A formal framework for static (tissue) P  
501 systems. In *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki,*  
502 *Greece, June 25-28, 2007*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–  
503 284. Springer.
- 504 [10] Marc García-Arnau, David Pérez, Alfonso Rodríguez-Patón, and Petr Sosík. (2009). Spik-  
505 ing Neural P Systems: Stronger Normal Forms. *International Journal of Unconventional*  
506 *Computing*, 5(5):411–425.
- 507 [11] Manuel García-Quismondo, Luis F. Macías-Ramos, and Mario J. Pérez-Jiménez. (2013).  
508 *Implementing Enzymatic Numerical P Systems for AI Applications by Means of Graphic*  
509 *Processing Units*, pages 137–159. Springer, Berlin, Heidelberg.
- 510 [12] Zsolt Gazdag and Gábor Kolonits. (2019). A new method to simulate restricted variants  
511 of polarizationless P systems with active membranes. *Journal of Membrane Computing*,  
512 1(4):251–261.
- 513 [13] Mihai Ionescu, Andrei Păun, Gheorghe Păun, and Mario J. Pérez-Jiménez. (2006). Com-  
514 puting with spiking neural P systems: Traces and small universal systems. In *DNA Com-*  
515 *puting, 12th International Meeting on DNA Computing, DNA12, Seoul, Korea, June 5-9,*  
516 *2006*, volume 4287 of *Lecture Notes in Computer Science*, pages 1–16. Springer.

- 517 [14] Yun Jiang, Yansen Su, and Fen Luo. (2019). An improved universal spiking neural P  
518 system with generalized use of rules. *Journal of Membrane Computing*, 1(4):270–278.
- 519 [15] Shankara Narayanan Krishna. (2011). An overview of membrane computing. In *Dis-  
520 tributed Computing and Internet Technology - 7th International Conference, ICDCIT 2011,  
521 Bhubaneshwar, India, February 9-12, 2011*, volume 6536 of *Lecture Notes in Computer  
522 Science*, pages 1–14. Springer.
- 523 [16] Shankara Narayanan Krishna and Raghavan Rama. (2001). P systems with replicated  
524 rewriting. *Journal of Automata, Languages and Combinatorics*, 6(3):345–350.
- 525 [17] Alberto Leporati, Antonio E. Porreca, Claudio Zandron, and Giancarlo Mauri. (2013).  
526 Improved Universality Results for Parallel Enzymatic Numerical P Systems. *International  
527 Journal of Unconventional Computing*, 9(5-6):385–404.
- 528 [18] Vincenzo Manca, Luca Bianco, and Federico Fontana. (2005). Evolution and oscillation in  
529 p systems: Applications to biological phenomena. In *Membrane Computing*, pages 63–84,  
530 Berlin, Heidelberg. Springer.
- 531 [19] Carlos Martín-Vide, Gheorghe Păun, Juan Pazos, and Alfonso Rodríguez-Patón. (2003).  
532 Tissue P systems. *Theoretical Computer Science*, 296(2):295–326.
- 533 [20] Miguel A. Martínez-del-Amor, Luis F. Macías-Ramos, Luis Valencia-Cabrera, and Mario J.  
534 Pérez-Jiménez. (2016). Parallel simulation of population dynamics P systems: updates  
535 and roadmap. *Natural Computing*, 15(4):565–573.
- 536 [21] G. H. Mealy. (Sept 1955). A method for synthesizing sequential circuits. *The Bell System  
537 Technical Journal*, 34(5):1045–1079.
- 538 [22] Edward F. Moore. (1956). Gedanken-experiments on sequential machines. *Automata  
539 studies*, pages 129–153.
- 540 [23] Anthony Nash and Sara Kalvala. (2019). A P system model of swarming and aggregation  
541 in a Myxobacterial colony. *Journal of Membrane Computing*, 1(2):103–111.
- 542 [24] Linqiang Pan, Gheorghe Păun, and Gexiang Zhang. (2019). Foreword: Starting jmc.  
543 *Journal of Membrane Computing*, 1(4):1–2.
- 544 [25] Ana Pavel, Octavian Arsene, and Cătălin Buiu. (2010). Enzymatic numerical P sys-  
545 tems - a new class of membrane computing systems. In *Fifth International Conference  
546 on Bio-Inspired Computing: Theories and Applications, BIC-TA 2010, Changsha, China,  
547 September 23-26, 2010*, pages 1331–1336. IEEE.
- 548 [26] Ana Brândusa Pavel, Cristian Ioan Vasile, and Ioan Dumitrache. (2012). Robot localization  
549 implemented with enzymatic numerical P systems. In *Biomimetic and Biohybrid Systems -  
550 First International Conference, Living Machines 2012, Barcelona, Spain, July 9-12, 2012*,  
551 volume 7375 of *Lecture Notes in Computer Science*, pages 204–215. Springer.
- 552 [27] Hong Peng, Jun Wang, Jun Ming, Peng Shi, Mario J. Pérez-Jiménez, Wenping Yu, and  
553 Chengyu Tao. (2018). Fault Diagnosis of Power Systems Using Intuitionistic Fuzzy  
554 Spiking Neural P Systems. *IEEE Transactions on Smart Grid*, 9(5):4777–4784.
- 555 [28] Ignacio Pérez-Hurtado, David Orellana-Martín, Gexiang Zhang, and Mario J. Pérez-Jiménez.  
556 (2019). P-lingua in two steps: flexibility and efficiency. *Journal of Membrane Computing*,  
557 1(2):93–102.
- 558 [29] Gheorghe Păun. (2000). Computing with membranes. *Journal of Computer and System  
559 Sciences*, 61(1):108–143. First circulated as TUCS Research Report No 208 (November  
560 1998).
- 561 [30] Gheorghe Păun. (2001). P systems with active membranes: Attacking np-complete  
562 problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90.

- 563 [31] Gheorghe Păun and Radu A. Păun. (2006). Membrane Computing and Economics:  
564 Numerical P Systems. *Fundamenta Informaticae*, 73(1-2):213–227.
- 565 [32] Haina Rong, Mianjun Ge, Gexiang Zhang, and Ming Zhu. (2018). A novel approach  
566 for detecting fault lines in a small current grounding system using fuzzy reasoning spik-  
567 ing neural P systems. *International Journal of Computers, Communications & Control*,  
568 13(4):521–536.
- 569 [33] Haina Rong, Kang Yi, Gexiang Zhang, Jianping Dong, Prithwineel Paul, and Zhiwei  
570 Huang. (2019). Automatic Implementation of Fuzzy Reasoning Spiking Neural P Systems  
571 for Diagnosing Faults in Complex Power Systems. *Complexity*, 2019:Article ID 2635714,  
572 16 pages.
- 573 [34] Eduardo Sánchez-Karhunen and Luis Valencia-Cabrera. (2019). Modelling complex  
574 market interactions using pdp systems. *Journal of Membrane Computing*, 1(1):40–51.
- 575 [35] Claude E. Shannon. (Jan 1949). The synthesis of two-terminal switching circuits. *The*  
576 *Bell System Technical Journal*, 28(1):59–98.
- 577 [36] Manca Vincenzo. (2019). From biopolymer duplication to membrane duplication and  
578 beyond. *Journal of Membrane Computing*, 1(4):292–303.
- 579 [37] Tao Wang, Gexiang Zhang, Junbo Zhao, Zhenyou He, Jun Wang, and Pérez-Jiménez. (May  
580 2015). Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural  
581 p systems. *IEEE Transactions on Power Systems*, 30(3):1182–1194.
- 582 [38] Xueyuan Wang, Gexiang Zhang, Xiantai Gou, Prithwineel Paul, Ferrante Neri, Haina  
583 Rong, Qiang Yang, and Hua Zhang. (2020). Multi-behaviors coordination controller  
584 design with enzymatic numerical P systems for robots. *Integrated Computer-Aided Engi-  
585 neering*, 27:in press.
- 586 [39] Xueyuan Wang, Gexiang Zhang, Ferrante Neri, Tao Jiang, Junbo Zhao, Marian Gheorghe,  
587 Florentin Ipate, and Raluca Lefticaru. (2016). Design and implementation of membrane  
588 controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integrated*  
589 *Computer-Aided Engineering*, 23(1):15–30.
- 590 [40] Xueyuan Wang, Gexiang Zhang, Haina Rong, Prithwineel Paul, and Hua Zhang. (2018).  
591 Multi-behaviors coordination controller design with enzymatic numerical P systems for au-  
592 tonomous mobile robots in unknown environments. In Michael J. Dinneen and Radu Nico-  
593 lescu, editors, *Proceedings of the Asian Branch of International Conference on Membrane*  
594 *Computing (ACMC2018)*, volume 530, pages 257–287. Centre for Discrete Mathematics  
595 and Theoretical Computer Science, Auckland, New Zealand.
- 596 [41] Xilinx. Vivado design suite user guide: Using constraints. [https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2019\\_2/ug903-vivado-using-constraints.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug903-vivado-using-constraints.pdf).
- 599 [42] Xilinx. Xilinx cordic v6.0 logicore ip product guide. [https://www.xilinx.com/support/documentation/ip\\_documentation/cordic/v6\\_0/pg105-cordic.pdf](https://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf).
- 602 [43] Gexiang Zhang, Jixiang Cheng, Marian Gheorghe, and Qi Meng. (2013). A hybrid ap-  
603 proach based on differential evolution and tissue membrane systems for solving constrained  
604 manufacturing parameter optimization problems. *Applied Soft Computing*, 13(3):1528–  
605 1542.
- 606 [44] Gexiang Zhang, Marian Gheorghe, Linqiang Pan, and Mario J. Pérez-Jiménez. (2014).  
607 Evolutionary membrane computing: A comprehensive survey and new results. *Information*  
608 *Sciences*, 279:528–551.
- 609 [45] Gexiang Zhang, Mario J. Pérez-Jiménez, and Marian Gheorghe. (2017). *Real-life appli-  
610 cations with membrane computing*. Springer.

- 611 [46] Gexiang Zhang, Haina Rong, Ferrante Neri, and Mario J. Pérez-Jiménez. (2014). An Opti-  
612 mization Spiking Neural P System for Approximately Solving Combinatorial Optimization  
613 Problems. *International Journal of Neural Systems*, 24(5):Article No. 1440006, 16 pages.