Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 244-252

First Steps Towards a CPU Made of Spiking Neural P Systems

Miguel A. Gutiérrez-Naranjo, Alberto Leporati

Miguel A. Gutiérrez-Naranjo
University of Sevilla, Department of Computer Science and Artificial Intelligence
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mail: magutier@us.es

Alberto Leporati
Università degli Studi di Milano – Bicocca, Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336/14, 20126 Milano, Italy
E-mail: alberto.leporati@unimib.it

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: We consider spiking neural P systems as devices which can be used to perform some basic arithmetic operations, namely addition, subtraction, comparison and multiplication by a fixed factor. The input to these systems are natural numbers expressed in binary form, encoded as appropriate sequences of spikes. A single system accepts as inputs numbers of any size. The present work may be considered as a first step towards the design of a CPU based on the working of spiking neural P systems.
Keywords: Spiking neural P systems, Arithmetic operations, Membrane Computing

1 Introduction

Spiking neural P systems (SN P systems, for short) have been introduced in [3] as a new class of distributed and parallel computing devices. They were inspired by *membrane systems* (also known as *P systems*) [12, 13, 7] and are based on the neurophysiological behavior of neurons sending electrical impulses to other neurons. In SN P systems the processing elements are called *neurons* and are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consist of a number of copies of a single object type, namely the *spike*. Each neuron may also contain rules which allow to remove a given number of spikes from it, or to send spikes (possibly with a delay) to other neurons. The application of every rule is determined by checking the contents of the neuron against a regular set associated with the rule.

Formally, an SN P system of degree $m \geq 1$, as defined in [4], is a construct of the form $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$, where $O = \{a\}$ is the singleton alphabet (a is called *spike*); $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, with $1 \leq i \leq m$, where: $n_i \geq 0$ is the *initial number of spikes* contained in σ_i; R_i is a finite set of *rules* of the following two forms:

(1) $E/a^c \rightarrow a;d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$ are integer numbers. If $E = a^c$, then it is usually written in the simplified form $a^c \rightarrow a;d$; similarly, if a rule $E/a^c \rightarrow a;d$ has $d = 0$, then we can simply write it as $E/a^c \rightarrow a$. Hence, if a rule $E/a^c \rightarrow a;d$ has $E = a^c$ and $d = 0$, then we can write $a^c \rightarrow a$;

(2) $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a;d$ of type (1) from R_i, we have $a^s \notin L(E)$ (where $L(E)$ denotes the regular language defined by E);

$syn \subseteq \{1,2,\ldots,m\} \times \{1,2,\ldots,m\}$, with $(i,i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons; $in, out \in \{1,2,\ldots,m\}$ indicate the *input* and *output* neurons of Π.

The rules of type (1) are called *firing* (also *spiking*) *rules*, and are applied as follows. If the neuron σ_i contains $k \geq c$ spikes, and $a^k \in L(E)$, then the rule $E/a^c \rightarrow a;d \in R_i$ can be applied. The execution of this rule removes c spikes from σ_i (thus leaving $k-c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i,j) \in syn$. If $d = 0$, then the spike is immediately emitted, otherwise it is emitted after d computation steps of the system. (Observe that, as usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.) If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed*, so that it cannot receive new spikes (if a neuron has a

synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost), and cannot fire new rules. In the step $t+d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t+d+1$) and select rules to be fired.

Rules of type (2) are called *forgetting* rules, and are applied as follows: if the neuron σ_i contains *exactly* s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i. In what follows we will use an *extended version* of forgetting rules, written in the form $E/a^s \to \lambda;d$. The application of these rules is analogous to that of firing rules. With respect to their basic version, extended forgetting rules are controlled by a regular expression, and may compete against firing rules for their application. It is possible to prove that the use of extended forgetting rules does not modify the computational power of SN P systems.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. In case two or more rules can be applied in a neuron at a given computation step, only one of them is nondeterministically chosen. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

The *initial configuration* of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron, with all neurons being open. During the computation, a configuration is described by both the number of spikes present in each neuron and by the number of steps to wait until it becomes open (this number is zero if the neuron is already open). A *computation* in a system as above starts in the initial configuration. A positive integer number is given as input to a specified *input neuron*. Usually, the number is specified as the time elapsed between the arrival of two spikes. However, as discussed in [4], other possibilities exist: for example, we can consider the number of spikes initially contained in the input neuron, or the number of spikes read in a given interval of time. All these possibilities are equivalent from the point of view of computational power. To pass from a configuration to another one, for each neuron a rule is chosen among the set of applicable rules, and is executed. Generally, a computation may not halt. However, in any case the output of the system is usually considered to be the time elapsed between the arrival of two spikes in a designated *output cell*. Defined in this way, SN P systems compute (partial) functions of the kind $f : \mathbb{N} \to \mathbb{N}$; they can also indirectly compute functions of the kind $f : \mathbb{N}^k \to \mathbb{N}$ by using a bijection from $\mathbb{N}^k$ to $\mathbb{N}$. It is not difficult to show that SN P systems can simulate register machines [4], and hence are universal.

If we do not specify an input neuron (hence no input is taken from the environment) then we use SN P systems in the *generative* mode; we start from the initial configuration, and we look at the output produced by the system. Note that generative SN P systems are inherently nondeterministic, otherwise they would always reproduce the same sequence of computation steps, and hence the same output. Dually, we can neglect the output neuron and use SN P systems in the *accepting* mode; for $k \geq 1$, the natural numbers $n_1, n_2, \ldots, n_k$ are read in input and, if the computation halts, then the numbers are accepted. Also in these cases, SN P systems are universal computation devices [3, 4].

In this paper we consider SN P systems in a different way. We will use them to build the components of a restricted Arithmetic Logic Unit in which one or several natural numbers are provided in binary form, some arithmetic operation is performed and the result is sent out also in binary form. The arithmetic operations we will consider are addition, subtraction and multiplication among natural numbers. Each number will be provided to the system as a sequence of spikes: at each time step, zero or one spikes will be supplied to an input neuron, depending upon whether the corresponding bit of the number is `0` or `1`. Also the output neuron will emit the computed number to the environment in binary form, encoded as a spike train.

The paper is organised as follows. In section 2 we present an SN P system which can be used to add two natural numbers expressed in binary form, of any length (that is, composed of any number of bits). In section 3 we present an analogous SN P system, that computes the difference (subtraction) among two natural numbers. Section 4 contains the description of a very simple system that can be used to compare two natural numbers. Section 5 first extends the system presented in section 2 to perform the addition of any given set of natural numbers, and then describes a spiking neural P system that performs the multiplication of any natural number, given as input, by a fixed factor embedded into the system. Finally, section 6 concludes the paper and suggests some possible directions for future research.

2 Addition

In this section we describe a simple SN P system that performs the addition of two natural numbers. We call such a system the *SN P system for 2-addition*. It is composed of three neurons (see Figure 1): two *input* neurons and an *addition* neuron, which is also the output neuron. Both input neurons have a synapse to the addition neuron. Each input neuron receives one of the numbers to be added as a sequence of spikes, that encodes the number in

Time step	$Input_1$	$Input_2$	Add	Output
$t=0$	0	0	0	0
$t=1$	**0**	**1**	0	0
$t=2$	**0**	**0**	1	0
$t=3$	**1**	**1**	0	**1**
$t=4$	**1**	**0**	2	**0**
$t=5$	**1**	**1**	2	**0**
$t=6$	0	0	3	**0**
$t=7$	0	0	1	**1**
$t=8$	0	0	0	**1**

Table 1: Number of spikes in each neuron of Π_{Add}, and number of spikes sent to the environment, at each time step during the computation of the addition $11100_2 + 10101_2 = 110001_2$

binary form. As explained above, no spike in the sequence at a given time instant means 0 in the corresponding position of the binary expression, whereas one spike means 1. Note that the numbers provided as input to the system may be arbitrarily long. The input neurons have only one rule, $a \to a$, which is used to forward the spikes to the addition neuron as soon as they arrive. The addition neuron has three rules: $a \to a$, $a^2/a \to \lambda$ and $a^3/a^2 \to a$, which are used to compute the result.

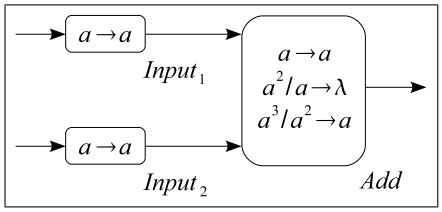

Figure 1: An SN P system that performs the addition among two natural numbers

Theorem 1. *The SN P system for 2-addition outputs the addition in binary form of two non-negative integers, provided to the neurons σ_{Input_1} and σ_{Input_2} in binary form.*

Proof. At the beginning of the computation, the system does not contain any spike. During the computation, neuron σ_{Add} may contain 0, 1, 2 or 3 spikes. We can thus divide the behavior of σ_{Add} in three cases:

• If there are no spikes, no rules are activated and in the next step 0 spikes are sent to the environment. This encodes the operation $0+0=0$.

• If there is 1 spike, then the rule $a \to a$ is triggered. The spike is consumed and one spike is sent out. This encodes $0+1=1+0=1$.

• If there are 2 spikes, then the rule $a^2/a \to \lambda$ is triggered. No spike is sent out and one spike (the carry) remains in the neuron for the next step.

• If there are 3 spikes, then the rule $a^3/a^2 \to a$ is applied. One spike is sent to the environment, two of them are consumed and one remains for the next step.

From this behavior, it is easily seen that the output is computed correctly. At the third computation step, the first of the spikes in the spike train that encodes the output in binary form is emitted by σ_{Add}. □

As an example, let us consider the addition $28+21=49$, that in binary form can be written as $11100_2 + 10101_2 = 110001_2$. Table 1 reports the number of spikes contained in each neuron of Π_{Add}, as well as the number of spikes sent to the environment, at each time step during the computation. The input and the output sequences are written in bold.

3 Subtraction

The *Subtraction SN P system*, illustrated in Figure 2, consists of ten neurons. The first input number, the *minuend*, is provided to neuron σ_{Input_1} in binary form, encoded as a spike train as described above. Similarly, the

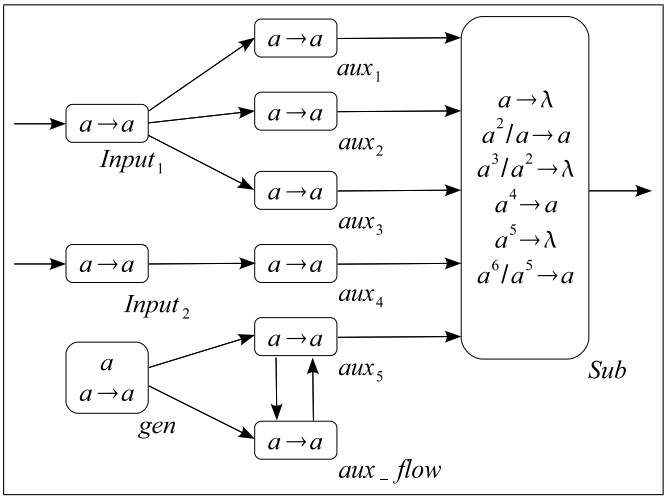

Figure 2: An SN P system that performs the subtraction among two natural numbers

Time step	$Input_1$	$Input_2$	aux_1	aux_2	aux_3	aux_4	aux_5	Sub	Output
$t=0$	0	0	0	0	0	0	0	0	0
$t=1$	**0**	**1**	0	0	0	0	1	0	0
$t=2$	**0**	**1**	0	0	0	1	1	1	0
$t=3$	**1**	**0**	0	0	0	1	1	2	0
$t=4$	**1**	**0**	1	1	1	0	1	3	**1**
$t=5$	**0**	**1**	1	1	1	0	1	5	**0**
$t=6$	**1**	**1**	0	0	0	1	1	4	**0**
$t=7$	**1**	0	1	1	1	1	1	2	**1**
$t=8$	0	0	1	1	1	0	1	6	**1**
$t=9$	0	0	0	0	0	0	1	5	**1**
$t=10$	0	0	0	0	0	0	1	1	**0**

Table 2: Number of spikes in each neuron of Π_{Sub}, and number of spikes sent to the environment, at each time step during the computation of the subtraction $1101100_2 - 110011_2 = 111001_2$

second input number (the *subtrahend*) is supplied in binary form to neuron σ_{Input_2}. The set of neurons σ_{aux_1}, σ_{aux_2} and σ_{aux_3} act as a multiplier of the minuend: they multiply by 3 the number of spikes provided by neuron σ_{Input_1}. The system contains also a subsystem composed of neurons σ_{gen}, σ_{aux_flow} and σ_{aux_5}, whose target is to provide a constant flow of spikes to σ_{Sub}. All the neurons mentioned up to now have only one rule: $a \to a$. The neurons σ_{aux_i}, for $1 \leq i \leq 5$, are connected with neuron σ_{Sub}; this is both the output neuron and the neuron in which the result of the subtraction is computed, by means of six rules: $a \to \lambda$, $a^2/a \to a$, $a^3/a^2 \to \lambda$, $a^4 \to a$, $a^5 \to \lambda$ and $a^6/a^5 \to a$. At the beginning of the computation all neurons are empty except σ_{gen}, which contains one spike.

Theorem 2. *The subtraction SN P system outputs the subtraction, in binary form, of two non-negative integer numbers, provided in binary form to neurons σ_{Input_1} (the minuend) and σ_{Input_2} (the subtrahend).*

The result can be easily checked by direct inspection of all possible cases. A detailed proof of this theorem — not given here, due to the lack of space — can be found in [2].

As an example let us calculate $108 - 51 = 57$, that in binary form can be written as $1101100_2 - 110011_2 = 111001_2$. Table 2 reports the number of spikes that occur in each neuron of Π_{Sub}, at each time step during the computation. Note that at each step only one rule is active in the subtraction neuron, and thus the computation is deterministic. The first time step in which the output starts to be emitted by the system is $t = 4$.

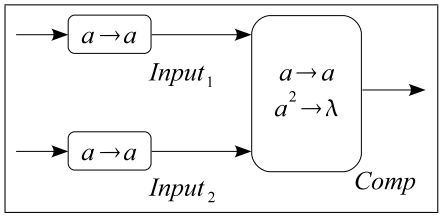

Figure 3: An SN P system that compares two natural numbers of any length, expressed in binary form

4 Checking Equality

Checking the equality of two numbers is a different task with respect to computing addition or subtraction. When comparing two numbers the output should be a binary mark, which indicates whether they are equal or not. Since an SN P system produces a spike train, we will encode the output as follows: starting from an appropriate instant of time, at each computation step the system will emit a spike if and only if the two corresponding input bits (that were inserted into the system some time steps before) are different. So doing, the system will emit no spike to the environment if the input numbers are equal, and at least one spike if they are different. Stated otherwise, if we compare two n-bit numbers then the output will also be an n-bit number: if such an output number is 0, then the input numbers are equal, otherwise they are different.

Bearing in mind these marks for equality and inequality, the design of the SN P system is trivial. It consists of three neurons: two input neurons, having $a \to a$ as the single rule, linked to a third neuron, the *checking neuron*. This checking neuron is also the output neuron, and it has only two rules: $a^2 \to \lambda$ and $a \to a$. The system is illustrated in Figure 3.

5 Multiplication

In this section we present a first approach to the problem of computing the multiplication of two binary numbers by means of SN P systems. The main difference between multiplication and the addition or subtraction operations presented in the previous sections is that in addition and subtraction the n-th digit in the binary representation of the inputs is used exactly once, to compute the n-th digit of the output, and then it can be discarded. On the contrary, in the usual algorithm for multiplication the different digits of the inputs are reused several times; hence the design of a device that executes this algorithm needs some kind of memory. Other algorithms for multiplication, such as Booth's algorithm (see, for example, [1]) also need some kind of memory, to store the intermediate results.

We propose a family of SN P systems for performing the multiplication of two non-negative integer numbers. In these systems only one number, the multiplicand, is provided as input; the other number, the multiplier, is instead encoded in the structure of the system. The family thus contains one SN P system for each possible multiplier.

In the design of our systems, we exploit the following basic fact concerning multiplication by one binary digit: any number remains the same if multiplied by 1, whereas it produces a 0 if multiplied by zero. Bearing this fact in mind, an SN P system associated to a fixed multiplier only needs to add different copies of the multiplicand, by feeding such copies to an addition device with the appropriate delay. Before presenting this design, we extend the 2-addition SN P system from section 2 to an n-addition SN P system.

5.1 Adding n numbers

In this section we present a family $\{\Pi_{Add}(n)\}_{n\in\mathbb{N}}$ of SN P systems which allows to add numbers expressed in binary form. Precisely, for any integer $n \geq 2$ the system $\Pi_{Add}(n)$ computes the sum of n natural numbers. In what follows we will call $\Pi_{Add}(n)$ the SN P system for n-addition. For $n = 2$ we will obtain the SN P system for 2-addition that we have described in section 2.

The system $\Pi_{Add}(n)$ consists of $n+1$ neurons: n *input* neurons and one *addition neuron*, which is also the output neuron. Each input neuron has only one rule, $a \to a$, and is linked to the addition neuron. This latter neuron computes the result of the computation by means of n rules r_i, $i \in \{1,\ldots,n\}$, which are defined as follows: $r_i \equiv a^i/a^{k+1} \to a$ if i is odd and $i = 2k+1$, whereas $r_i \equiv a^i/a^k \to \lambda$ if i is even and $i = 2k$.

As an example, Figure 4 shows $\Pi_{Add}(5)$, the SN P system for 5-addition.

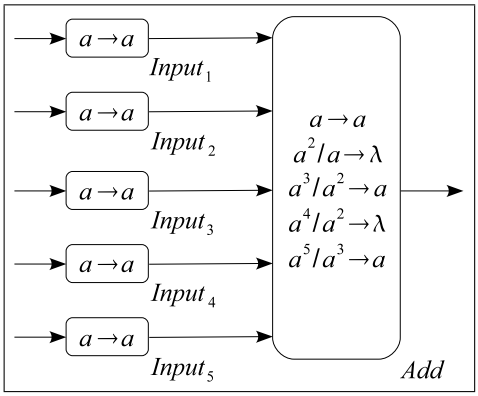

Figure 4: An SN P system that performs the addition among five natural numbers

Theorem 3. *The SN P system for n-addition outputs the addition in binary form of n non-negative integer numbers, provided to the neurons $\sigma_{Input_1}, \ldots, \sigma_{Input_n}$ in binary form.*

Proof. Let $A_1, \ldots, A_n$ be the n numbers to be added, and let $a_i^p a_i^{p-1} \ldots a_i^0$ be the binary expression of A_i, $1 \leq i \leq n$, padded with zeros on the left to obtain $(p+1)$-digit numbers (where $p+1$ is the maximum number of digits among the binary representations of $A_1, \ldots, A_n$). Hence we can write $A_i = \sum_{k=0}^{p} a_i^k 2^k$ for all $i \in \{1, 2, \ldots, n\}$.

For each $i \in \{1, \ldots, n\}$, let A_i' be the number with binary expression $a_i^p \ldots a_i^1$, i.e., $A_i' = \sum_{k=1}^{p} a_i^k 2^{k-1}$. Moreover, let $U = \sum_{i=1}^{n} a_i^0$ and let $k \in \mathbb{N}$ and $\alpha \in \{0, 1\}$ such that $U = 2k + \alpha$ ($\alpha = 1$ if U is odd and $\alpha = 0$ if U is even). The addition of $A_1, \ldots, A_n$ can be written as:

$$\sum_{i=1}^{n} A_i = \sum_{i=1}^{n} \sum_{k=0}^{p} a_i^k 2^k = \left(\sum_{i=1}^{n} \sum_{k=1}^{p} a_i^k 2^k \right) + \sum_{i=1}^{n} a_i^0 = 2 \left(\sum_{i=1}^{n} A_i' + k \right) + \alpha$$

According to this formula, if $b_r \ldots b_0$ is the binary expression of $\sum_{i=1}^{n} A_i$, then $b_0 = \alpha$ and $b_r \ldots b_1$ is the binary expression of $\sum_{i=1}^{n} A_i' + k$.

Let us assume now that at the time instant t there are i spikes in neuron σ_{Add}. These spikes can come from the input neurons, or they may have remained from the previous computation step. Let us compute b_t, the t-th digit of the output, dividing the problem in the following cases.

• Let us assume that i is odd and $i = 2k + 1$. Then, according to the previous formula, $b_t = 1$ and k units should be added to the computation of the next digit. This operation is performed by the rule $a^i/a^{k+1} \to a$. By applying this rule, one spike is sent to the environment ($b_t = 1$) and $k+1$ spikes are consumed, so that $i - (k+1) = 2k + 1 - (k+1) = k$ spikes remain.

• Let us assume that i is even and $i = 2k$. Then, according to the previous formula, $b_t = 0$ and k units should be added to the computation of the next digit. This operation is performed by the rule $a^i/a^k \to \lambda$. By applying this rule, no spike is sent to the environment ($b_t = 0$) and k spikes are consumed, so that $i - k = 2k - k = k$ spikes remain for the next step. □

As an example, let us consider the addition of the numbers 3, 4, 2, 7 and 1, whose binary representations are 11_2, 100_2, 10_2, 111_2 and 1_2, respectively. Table 3 shows the evolution of the number of spikes in the neurons of the SN P system $\Pi_{Add}(5)$ (illustrated in Figure 4), as well as the number of spikes sent to the environment at each computation step, when performing such an addition. The input and the output sequences are written in bold. According with the computation, the result of the addition is $17 = 10001_2$.

5.2 Multiplication by a fixed multiplier

We now describe a family $\{\Pi_{Mult}(n)\}_{n \in \mathbb{N}}$ of SN P systems, one for each natural number n, that operate as multiplier devices. Precisely, the system $\Pi_{Mult}(n)$ takes as input a number in binary form, and outputs the input multiplied by n. The output is also expressed in binary form.

Given a natural number n, the SN P system $\Pi_{Mult}(n)$ is defined as follows. It consists of one *input* neuron, σ_{Input}, linked to k neurons $\sigma_{aux_{1_1}}, \ldots, \sigma_{aux_{k_1}}$, where k is the number of occurrences of the digit 1 in the binary

Time step	$Input_1$	$Input_2$	$Input_3$	$Input_4$	$Input_5$	Add	Output
$t=1$	**1**	**0**	**0**	**1**	**1**	0	0
$t=2$	**1**	**0**	**1**	**1**	0	3	0
$t=3$	0	**1**	0	**1**	0	4	**1**
$t=4$	0	0	0	0	0	4	**0**
$t=5$	0	0	0	0	0	2	**0**
$t=6$	0	0	0	0	0	1	**0**
$t=7$	0	0	0	0	0	0	**1**

Table 3: Number of spikes in each neuron of $\Pi_{Add}(5)$ (the system illustrated in Figure 4) and number of spikes sent to the environment, at each time step during the computation of the addition $11_2+100_2+10_2+111_2+1_2=10001_2$

representation of n. For each $i\in\{1,\ldots,k\}$, neuron $\sigma_{aux_{i_1}}$ is connected with a new neuron $\sigma_{aux_{i_2}}$, which is connected with $\sigma_{aux_{i_3}}$, etc. This sequence of neurons is a path of linked neurons that extends until reaching $\sigma_{aux_{ij_i}}$, where j_i is the number of order of the corresponding digit in the binary representation of n, where the first digit corresponds to 2^0, the second one corresponds to 2^1, and so on. All the last neurons of the k sequences are connected with a final neuron σ_{Add}, which is the same as the output neuron of the k-addition SN P system $\Pi_{Add}(k)$ described above. This neuron has the rules for the addition of k natural numbers. All the other neurons have only the rule $a\to a$.

For example, let us consider $n=26$, whose binary representation is 11010_2. Such a representation has three digits equal to 1, at the positions 2, 4 and 5. The system $\Pi_{Mult}(26)$, illustrated in Figure 5, has 13 neurons: σ_{Input}, σ_{Add}, and three sequences of neurons associated with the three digits equal to 1: $\sigma_{aux_{11}}$ and $\sigma_{aux_{12}}$, corresponding to the 1 in the second position (corresponding to the power 2^1); $\sigma_{aux_{21}}$, $\sigma_{aux_{22}}$, $\sigma_{aux_{23}}$ and $\sigma_{aux_{24}}$, corresponding to the 1 in the fourth position (corresponding to the power 2^3); $\sigma_{aux_{31}}$, $\sigma_{aux_{32}}$, $\sigma_{aux_{33}}$, $\sigma_{aux_{34}}$ and $\sigma_{aux_{35}}$, corresponding to the 1 in the fifth position (corresponding to the power 2^4).

The last neurons of these sequences, namely $\sigma_{aux_{12}}$, $\sigma_{aux_{24}}$ and $\sigma_{aux_{35}}$, are linked to neuron σ_{Add}, which is also the output neuron. The rules of this neuron are $a\to a$, $a^2/a\to\lambda$ and $a^3/a^2\to a$, which are the same as in the addition neuron of the 3-addition SN P system $\Pi_{Add}(3)$ described in the previous section.

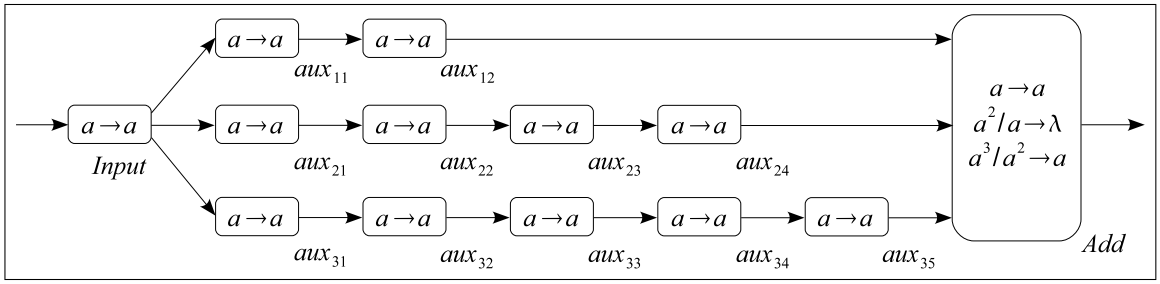

Figure 5: An SN P system that computes the product among the natural number given as input (in binary form) and the fixed multiplier $26=11010_2$, encoded in the structure of the system

Theorem 4. *The SN P system $\Pi_{Mult}(n)$ built as above takes as input a number m in binary form and outputs the result of the multiplication $m\cdot n$ in binary form.*

Proof. Since we already proved that the neuron σ_{Add} performs the addition of several numbers in binary form, it only remains to transform the multiplication $m\cdot n$ (where n is a fixed parameter) into an appropriate addition. To this aim, let $n=\sum_{j=0}^{q}n_j2^j$. Then we can write

$$m\cdot n=m\cdot\left(\sum_{j=0}^{q}n_j2^j\right)=\sum_{j=0}^{q}\left(m\cdot 2^j\right)n_j=\sum_{0\le j\le q\,\wedge\,n_j=1}\left(m\cdot 2^j\right)$$

According to this expression, $m\cdot n$ can be calculated as the addition of as many copies of m as the number of digits n_j equal to 1 that appear in the binary representation of n. Such copies have to be padded with j zeros

Time step	*Input*	aux_{12}	aux_{24}	aux_{35}	*Add*	*Out*
$t=1$	**1**	0	0	0	0	0
$t=2$	**0**	0	0	0	0	0
$t=3$	**1**	1	0	0	0	0
$t=4$	**1**	0	0	0	1	**0**
$t=5$	**1**	1	1	0	0	**1**
$t=6$	0	1	0	1	2	**0**
$t=7$	0	1	1	0	3	**0**
$t=8$	0	0	1	1	3	**1**
$t=9$	0	0	1	1	3	**1**
$t=10$	0	0	0	1	3	**1**
$t=11$	0	0	0	0	2	**1**
$t=12$	0	0	0	0	1	**0**
$t=13$	0	0	0	0	0	**1**

Table 4: Number of spikes in neurons $\sigma_{aux_{12}}$, $\sigma_{aux_{24}}$, $\sigma_{aux_{35}}$ and σ_{Add} of $\Pi_{Mult}(26)$ (the system illustrated in Figure 5) and number of spikes sent to the environment, at each time step during the computation of the multiplication $11101_2 \cdot 11010_2 = 1011110010_2$

on the right (that is, they have to be multiplied by 2^j), to take into account the correct weight of n_j. Hence, if $k = \sum_{j=0}^{q} n_j$ then to compute $m \cdot n$ it suffices to provide k copies of m — each shifted in time of a number of steps that corresponds to the weight of a bit n_j equal to 1 — to a neuron that computes the addition of k natural numbers. □

6 Conclusion and Future Work

In this paper we have presented some simple SN P systems that perform the following operations: addition, multiple addition, comparison, and multiplication by a fixed factor. All the numbers given as inputs to these systems are expressed in binary form, encoded as a spike train in which at each time instant the presence of a spike denotes 1, and the absence of a spike denotes 0. The outputs of the computations are also expelled to the environment in the same form.

The motivation for this work lies in the fact that we would like to implement a CPU using only spiking neural P systems. To this aim, the first step is to design the Arithmetic Logic Unit of the CPU, and hence to study a compact way to perform arithmetical and logical operations by means of spiking neural P systems. Ours is certainly not the unique possible way to approach the problem; other two possibilities are: (1) implementing the CPU as a network composed of AND/OR/NOT Boolean gates, and (2) simulating the CPU by means of register machines. In both cases, using techniques widely known in the literature, one could design an SN P system that simulates the Boolean network (resp., the register machine), thus implementing the CPU.

In any case, an interesting extension to the present work is to try to design an SN P system for the multiplication, where both the numbers m and n to be multiplied are supplied as inputs. And, of course, we would also need a system to compute the integer division between two natural numbers; probably, this last system is the most difficult to design.

Acknowledgement

The first author wishes to acknowledge the support of the project TIN2006–13425 of Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200. The second author was partially supported by the MIUR project “Mathematical aspects and emerging applications of automata and formal languages" (2007).

Bibliography

[1] M.J. Flynn. *Advanced computer arithmetic design*. John Wiley Publisher, 2001.

[2] M.A. Gutiérrez-Naranjo and A. Leporati. Performing arithmetic operations with spiking neural P systems. In *Proc. of the Seventh Brainstorming Week on Membrane Computing*, Vol. I, Fénix Editora, Seville, Spain, 2009, 181–198. Availble at `http://www.gcn.us.es`.

[3] M. Ionescu, Gh. Păun and T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, **71**(2-3):279–308, 2006.

[4] M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez. Computing with spiking neural P systems: Traces and small universal systems. In *DNA Computing,* 12^{th} *International Meeting on DNA Computing (DNA12)*, Revised Selected Papers, LNCS 4287, Springer, 2006, 1–16.

[5] Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**:108–143, 2000. See also Turku Centre for Computer Science — TUCS Report No. 208, 1998.

[6] Gh. Păun. *Membrane computing. An introduction*. Springer–Verlag, 2002.

[7] The P systems web page: `http://ppage.psystems.eu/`

Miguel A. Gutiérrez-Naranjo is an associate professor at the Department of Computer Science and Artificial Intelligence of the University of Seville in Spain. He obtained his doctoral degree in Mathematics in 2002. His main research area is Natural Computing, with a special interest in Membrane Computing.

Alberto Leporati obtained a Ph.D. in Computer Science from the University of Milano (Italy) in 2002. Since 2004, he is assistant professor at the University of Milano – Bicocca. His research interests are in Membrane Computing, Theoretical Computer Science and Computational Complexity.