International Journal of Neural Systems, Vol. XX, No. X (2020) 1–16

A Complete Arithmetic Calculator Constructed from Spiking Neural P Systems and its Application to Information Fusion

Gexiang Zhang*
Research Center for Artificial Intelligence,
Chengdu University of Technology, Chengdu, 610059, China
E-mail: zhgxdylan@126.com

Haina Rong, Prithwineel Paul, Yangyang He
School of Electrical Engineering, Southwest Jiaotong University, Chengdu, 610031, China
Email: ronghaina@126.com, prithwineelpaul@gmail.com, he.yangyang@foxmail.com

Ferrante Neri*
COL Laboratory, School of Computer Science, University of Nottingham, Nottingham, UK
E-mail:ferrante.neri@nottingham.ac.uk

Mario J. Pérez-Jiménez
Department of Computer Science and Artificial Intelligence
University of Sevilla, Avda. Reina Mercedes s/n, 41012, Spain
Email: marper@us.es

Several variants of spiking neural P systems (SNPS) have been presented in the literature to perform arithmetic operations. However, each of these variants was designed only for one specific arithmetic operation. In this paper, a complete arithmetic calculator implemented by spiking neural P systems is proposed. An application of the proposed calculator to information fusion is also proposed. The information fusion is implemented by integrating the following three elements: 1) an addition and subtraction SNPS already reported in the literature; 2) a modified multiplication and division SNPS; 3) a novel storage SNPS, i.e., a method based on spiking neural P systems is introduced to calculate basic probability assignment of an event. This is the first attempt to apply arithmetic operation SNPS to fuse multiple information. The effectiveness of the presented general arithmetic SNPS calculator is verified by means several examples.

Keywords: Membrane computing; spiking neural P system; arithmetic calculator; information fusion.

1. Introduction

Membrane computing[44] is a research field in computer science that aims at abstracting computing models from the structure and functioning of living cells[41]. These computing models are usually referred to as membrane systems or P systems [64; 45]. They can be categorised into three classes:

- cell-like P systems [44; 67; 40; 38; 58];
- tissue-like P systems [17; 66; 70];
- neural-like P systems [27; 39; 42; 52; 65; 50; 57; 29].

Spiking neural networks (SNNs) can efficiently mimic biological neural networks. These networks are also called as third generation neural networks [9; 18; 26; 55; 68; 20; 7; 6; 22; 21; 4] and massively parallel algorithms can be developed using these models [2; 1; 5; 3]. The neural-like P systems are well-known as spiking neural P systems which are inspired from the structure and functioning of biological neurons.

Moreover the concepts of SNNs are incorporated into these models. Many variants of spiking neural P systems have been introduced and computational power [11; 60; 13; 35; 42; 53], applications [50; 57; 64; 65; 37] of these models have been investigated.

Arithmetic is a branch of mathematics dealing with natural numbers and relations among them. A popular field of application of P systems is the modelling of arithmetic operations with natural numbers, i.e. addition, subtraction, multiplication and division[33; 56; 19; 35; 15]. Moreover, a new type of ALU(arithmetic logic unit) can be constructed using spiking neural P systems which further can be used for designing of CPU (central processing unit). Furthermore, constructing a complexity-effective ALU unit is an interesting area of research. This work is an attempt towards that direction. Another motivation of this paper is to provide a framework combining a relatively new variant of spiking neural network, i.e., spiking neural P systems and Dempster-Shafer (D-S) evidence theory which can be further used to solve real-world problems. Spiking neural P systems have been very efficient to solve many real-world problems such as fault diagnosis, image processing, computational biology etc. Many authors in [30; 8; 16; 61] have investigated new frameworks by combining different variants of neural networks and D-S evidence theory to solve real-world problems. The information obtained from any single source can be doubtful. Hence information fusion can reduce inaccuracy and uncertainty in the information received from neural networks. In this work, a novel framework combining the SN P systems and D-S evidence theory is proposed to expand the scope of application of the bio-computing models, specifically membrane computing models.

The arithmetic operation models can be divided into two categories:

- arithmetic operation models based on cell-like P systems [62; 24; 25];
- arithmetic operation models based on spiking neural P systems [63; 69; 32; 46; 31; 36; 12].

The focus of this paper is on the arithmetic operation models based on spiking neural P systems (SNPS). An SNPS is a class of neural-like P systems which is inspired by the neurophysiological behavior of neurons sending short electrical impulses (spikes), identical in shape, along axons from presynaptic neurons to postsynaptic neurons in a distributed and parallel manner [27]. SNPS can be viewed as computational model embedding elements of a spiking neural network[23; 28; 48; 54].

The basic arithmetic operations models based on SNPS include two encoding methods. The first method represents the number as the interval of elapsed time between two spikes [63; 69; 32]. The second method uses a spike sequence (spike train [49]) to encode the number. Numbers are represented in binary and at each step, zero or one spike will be supplied to an input neuron, depending upon whether the corresponding bit of the binary number is 0 or 1 [31; 36; 11]. The arithmetic operation models used in this paper are based on the second encoding method.

In Ref.[36], the operation models based on SNPS were designed for the addition of N arbitrary natural numbers, the subtraction of two arbitrary natural numbers, and the multiplication of an arbitrary natural number by a fixed arbitrary natural number, respectively. In Ref.[36], an open problem was also suggested, that is, how to design an SNPS to solve the multiplication of any two arbitrary natural numbers. In Ref.[69], SNPS were designed to solve the multiplication operation of two natural numbers with specified length. But in Ref.[69], there is only one input neuron, while in Ref.[36] there are multiple input neurons. In Ref.[63] and [32], an adder, subtracter, multiplier and divider based on SNPS were designed, respectively. But the operation models based on SNPS in Ref. [63] and [32] encode the interval of elapsed time between two spikes, but the models in Ref.[36] use binary codes, that is, the binary code "1" represents one spike and the binary code "0" represents no spike. In Ref.[46], SNPS were used to solve the operations of addition and multiplication, but the multiplication could not output correct results for some numbers. An SNPS for solving the division operation of two signed integer was presented in Ref.[12], but it could not output correct results for some numbers.

Several variants of SNPS have been discussed in the literature to perform arithmetic operations. However, each of these variants was designed only for a specific arithmetic operation. In the previous studies, no integration of several arithmetic operations based on SNPS have ever been used to perform a complex computation. Also many SNP sim-

ulators have been introduced [10; 34]. Unlike the previous studies, this paper makes the first attempt to implement information fusion by integrating several arithmetic operations based on SNPS, including addition and subtraction SNPS reported in Ref.[31], a modified version of the multiplication and division SNPS proposed in Ref.[46; 31; 11], and a novel SNPS, referred to as storage SNPS. The five types of operations designed by SNPS cooperate to fulfill the computation of D-S evidence information fusion [14; 51]. Also, the construction of SNPS simulator and the information fusion implementation discussed in this paper can be performed with less number of neurons because we considered a new variant of extended spiking neural P systems for performing the arithmetic operations. Extended spiking neural P systems are computationally universal and hence the model considered in this paper is also computationally universal. One major advantage of considering this model is that it can perform the arithmetic operations with less number of neurons, i.e., it is a good model from the descriptional complexity point of view.

The remainder of this paper is organized as follows. Section 2 briefly introduces spiking neural P systems and the notation used throughout this article. Section 3 presents four arithmetic units (addition, subtraction, multiplication and division units), storage unit and their simulator. In section 4, the information fusion SNPS is proposed to achieve two instances of D-S evidence information fusion. Finally, conclusions are drawn in Section 5.

2. Spiking neural P systems

Definition 1. A spiking neural P system of degree $m \geq 1$ is a computational model identified by the tuple[27; 13]:

$$\Pi = (O, \sigma_1, ..., \sigma_m, syn, in, out)$$

where

(1) $O = \{a\}$ is a **singleton alphabet** where a is called **spike**;

(2) $\sigma_1, \ldots \sigma_m$ are **neurons**, where $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, being

(a) n_i is the initial number of spikes contained in σ_i;

(b) R_i is the set of rules associated with σ_i of the following two types:

(i) **spiking rule**: $E/a^c \rightarrow a; d$
where E is a regular expression over O, and $c \geq 1, d \geq 0$. The spiking rule is applicable if a neuron σ_i contains k spikes, with $k \geq c$ and $a^k \in L(E)$. This rule removes c spikes after a delay of d steps and after a spike is sent to the neurons connected with the neuron where the rule is applied. If $d = 0$, then one spike is sent to the connected neurons immediately after application of the rule. Furthermore, if $L(E) = \{a^c\}$ then the rule can be simply written as $a^c \rightarrow a; d$. It is important to note that if $d \geq 1$ and the rule is applied at any time t, then during the period $t, t+1, t+2, \ldots, t+d-1$ the neuron becomes closed, i.e., no spike can arrive and no spike can leave the neuron during this period. However, the neuron becomes open again at the step $t + d$.

(ii) **forgetting rule**: $a^s \rightarrow \lambda$
with λ the empty string and s an integer ≥ 1. The forgetting rule works with the following restriction, i.e., $a^s \notin L(E)$ that for any spiking rule $E/a^c \rightarrow a; d$ of type (i) from R_i. Moreover this rule is applicable in a neuron σ_i when it contains s spikes and after application s spikes are removed from the system.
A generalization of spiking neural P systems is known as extended spiking neural P systems where the rules are of the form $E/a^c \rightarrow a^p; d$, $c \geq 1, p \geq 1$ and $p \leq c$. The above mentioned spiking rules are special cases of extended spiking neural P systems with $p = 1$.

(3) $syn \subseteq \{\sigma_1, \sigma_2, \ldots, \sigma_m\} \times \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ with $(\sigma_i, \sigma_i) \notin syn$, $1 \leq i \leq m$ represents the synaptic connection between the neurons. In fact, syn is a directed graph of synapses between the linked neurons;

(4) $in, out \subseteq \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ indicate the input neuron set and the output neuron set of Π, respectively.

Note: Forgetting rules used in this paper are of the form $a^m/a^n \rightarrow \lambda$ where $m \geq n, a^m \notin L(E)$ for any spiking rule of the form $E/a^c \rightarrow a^p$. Furthermore, after application of this rule n spikes are consumed and $(m - n)$ spikes will remain inside the neuron.

3. Arithmetic operation units

In this section, a natural number is coded in binary and a series of spikes is used to represent the coded number. For the input and output neurons, one spike represents the binary bit "1" and no spike represents the binary bit "0". The following subsections outline the four arithmetic operation units (addition, subtraction, multiplication and division) as well as the storage unit and their simulator.

3.1. *Add unit*

The Add unit initially introduced in Ref.[31] and used in this paper, performs the addition operation of two arbitrary natural numbers (each natural number is represented by a binary string). The Add unit is identified with the following SNPS:

$$\Pi_{Add} = (O, \sigma_1, \sigma_2, \sigma_3, syn, in, out)$$

where

(1) $O = \{a\}$;
(2) $\sigma_i = (0, R_i)$, $R_i = \{a \to a\}$, $i = 1, 2$;
(3) $\sigma_3 = (0, R_3)$, $R_3 = \{a \to a, a^2/a \to \lambda, a^3/a \to a\}$;
(4) $syn = \{(\sigma_1, \sigma_3), (\sigma_2, \sigma_3)\}$;
(5) $in = \{in_1, in_2\}$, being $in_1 = \sigma_1, in_2 = \sigma_2$;
(6) $out = \{\sigma_3\}$.

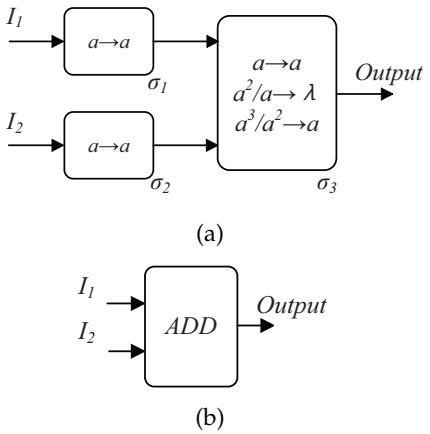

Fig. 1. Add unit.

The add functioning can be described in the following way. When two binary strings with m and n bits are given as input to the input neurons from low to high bits at step t, respectively, the neuron σ_3 starts to output the sum of binary strings from low to high bits at step t+3. Fig. 1(a) and Fig. 1(b) show the detailed and simplified structures of the Add unit, respectively.

In order to illustrate how the Add unit works, we take $(1110)_2 + (101)_2 = (10011)_2$ as an example. Table 1 shows the changes of the numbers of spikes for neurons $\sigma_1, \sigma_2, \sigma_3$, respectively, and the output.

Table 1. Changes of the numbers of spikes in Add unit

No. of steps	σ_1	σ_2	σ_3	output
0	0	0	0	-
1	**0**	**1**	0	-
2	**1**	**0**	1	-
3	**1**	**1**	1	**1**
4	**1**	0	2	**1**
5	0	0	2	**0**
6	0	0	1	**0**
7	0	0	0	**1**

3.2. *Sub unit*

The Sub unit used in this paper was introduced in Ref.[31]. The Sub unit can perform the subtraction operations of two arbitrary natural numbers represented by binary strings under the condition that the minuend must be greater than subtrahend. The SNPS of the Sub unit

$\Pi_{Sub} = (O, \sigma_1, \ldots, \sigma_{10}, syn, in, out)$, where

(1) $O = \{a\}$;
(2) $\sigma_i = (0, R_i)$, $R_i = \{a \to a\}$, $1 \le i \le 9, i \ne 3; \sigma_3 = (a, R_3), R_3 = \{a \to a\}$;
(3) $\sigma_{10} = (0, R_{10})$, $R_{10} = \{a \to \lambda, a^2/a \to a, a^3/a^2 \to \lambda, a^4 \to a, a^5 \to \lambda, a^6/a^5 \to a\}$;
(4) $syn = \{(\sigma_1, \sigma_4), (\sigma_1, \sigma_5), (\sigma_1, \sigma_6), (\sigma_2, \sigma_7), (\sigma_3, \sigma_8), (\sigma_3, \sigma_9), (\sigma_8, \sigma_9), (\sigma_9, \sigma_8), (\sigma_4, \sigma_{10}), (\sigma_5, \sigma_{10}), (\sigma_6, \sigma_{10}), (\sigma_7, \sigma_{10}), (\sigma_8, \sigma_{10})\}$;
(5) $in = \{in_1, in_2\}$, being $in_1 = \sigma_1, in_2 = \sigma_2$;
(6) $out = \{\sigma_{10}\}$.

When two binary strings with m and n bits are entered into the input neurons from lower to higher bits at step t, respectively, the neuron σ_{10} will start to output the subtraction of binary strings from lower to higher bits at step t+4. Fig. 2(a) and Fig. 2(b) show the detailed (10 neurons) and simplified Sub unit models, respectively.

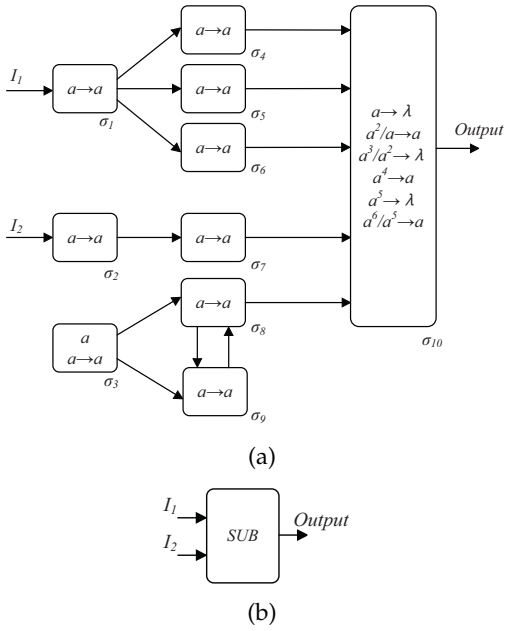

Fig. 2. Sub unit.

In order to illustrate the functioning of the Sub unit, the operation $(1110)_2 - (101)_2 = (1001)_2$ is taken as an example. Table 2 shows the changes of the numbers of spikes for Sub unit neurons and the output of neuron σ_{10}.

Table 2. Changes of the numbers of spikes in Sub unit

No. of steps	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	output
0	0	0	1	0	0	0	0	0	0	0	-
1	**0**	**1**	0	0	0	0	0	1	1	0	-
2	**1**	**0**	0	0	0	0	1	1	1	1	-
3	**1**	**1**	0	1	1	1	0	1	1	2	0
4	**1**	0	0	1	1	1	1	1	1	5	**1**
5	0	0	0	1	1	1	0	1	1	5	**0**
6	0	0	0	0	0	0	0	1	1	4	**0**
7	0	0	0	0	0	0	0	1	1	1	**1**

3.3. *Mul unit*

The Mul unit used in this paper is a modified version of the unit presented in Ref.[46]. The modified Mul unit is as shown in Fig. 3 and Fig. 4, in its detailed and simplified schemes, respectively. The novel/modified rules are highlighted in Fig. 3 within a dashed frame.

This version extends to any pair of arbitrary natural numbers the applicability of the Mul unit. In order to understand the functioning of this unit, let us consider two natural numbers X and Y expressed as binary strings

$$(x_0, x_1, \ldots, x_{m-1})$$
$$(y_0, y_1, \ldots, y_{n-1})$$

and they can be represented as sum of m and n terms in the following manner

$$X = \sum_{i=0}^{m-1} x_i 2^i \ , \ Y = \sum_{j=0}^{n-1} y_j 2^j.$$

The multiplication number Z is

$$Z = X \cdot Y = \sum_{i=0}^{m-1} y_0 x_i 2^{i+0} + \sum_{i=0}^{m-1} y_1 x_i 2^{i+1} + \cdots + \sum_{i=0}^{m-1} y_{n-2} x_i 2^{i+n-2} + \sum_{i=0}^{m-1} y_{n-1} x_i 2^{i+n-1}$$

Hence, the multiplication of $Z = X \cdot Y$ can be fulfilled by performing n addition operations.

The jth addition operation

$$\sum_{i=0}^{m-1} y_j x_i 2^{i+j}$$

expresses the product of X and $y_j 2^j$. If $y_j = 1$, the computing result of the expression is $X \cdot 2^j$, which means j bits of X are moved left. If $y_j = 0$, the computing result of the expression is 0. Hence we can conclude that $X.Y$ can be represented as sum of X copies of number of 1's in the binary representation of Y. As shown in Fig. 3, the neurons, $\sigma_{a10}, \sigma_{a11}$ and σ_{d1} can obtain the first addition operation

$$\sum_{i=0}^{m-1} y_0 x_i 2^{i+0}$$

Similarly, the kth addition operation

$$\sum_{i=0}^{m-1} y_k x_i 2^{i+k}$$

can be achieved by neurons $\sigma_{ak0}, \sigma_{ak1}, \ldots, \sigma_{akk}, \sigma_{dk}$. In Fig. 3, Part 1 with the dashed frame obtains the input X for n addition operations and Part 2 with the dashed frame achieves y_j input for n addition operations. The neuron σ_3 gains the sum of n additions.

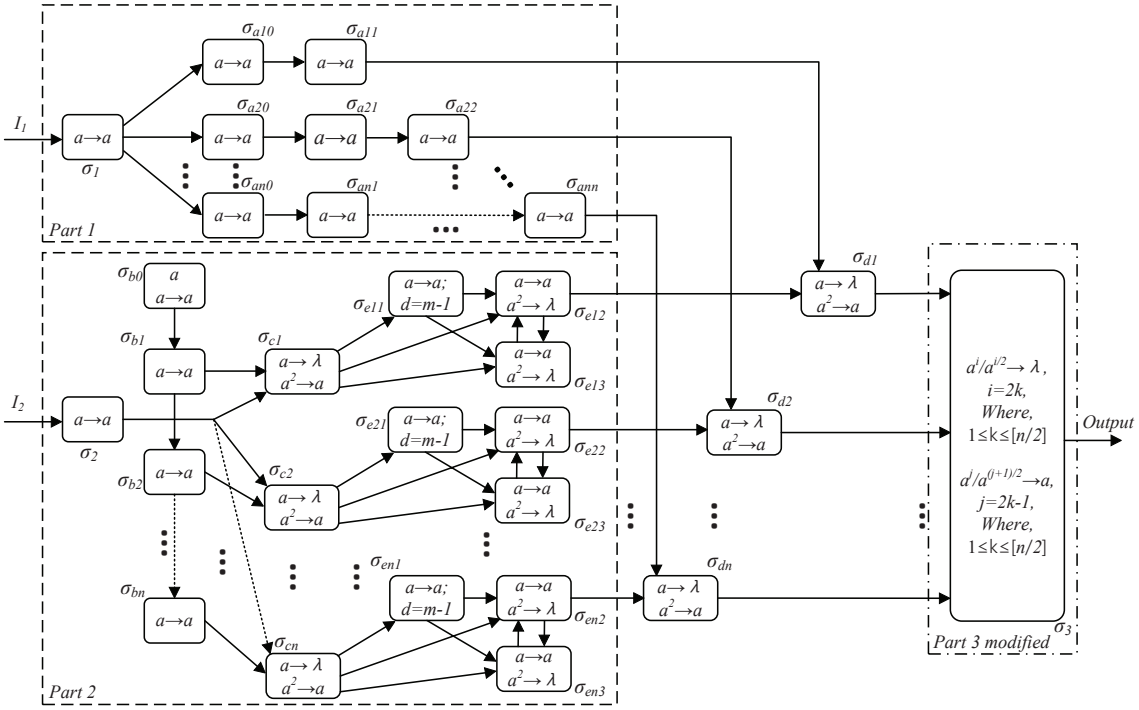

Fig. 3. Mul unit.

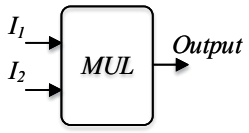

Fig. 4. Simplified version of Mul unit.

The proposed Mul unit shown in Fig. 3 is general and needs to be adjusted according to the number of binary bits representing different natural numbers. In our example, the Mul unit is used to calculate $Z = X \cdot Y$, where X is the binary string with m bits and Y is the binary sequence with n bits. Without a loss of generality, let us assume that $n \leq m$. First of all, we should adjust the Mul unit according to the general unit which is shown in Fig. 3. To simplify the Mul unit, the Y which has the smaller number of binary bits should be used as the multiplier input through neuron σ_2. Hence, the Mul unit includes $\frac{n^2+15n+8}{2}$ neurons because the neurons in Part 1 are associated with position of 1's in X. Secondly, X and Y with binary strings from lower to higher bits are inputted into neurons σ_1 and σ_2 at step t, respectively. Finally, σ_3 starts to output the product through binary sequence from low to high bits at step t+6.

From the above explanation we know that in the Mul unit the string with bigger length is considered as the multiplicand and the other string is considered as multiplier. In Part 2, if any input is 1, i.e., $y_j = 1$ for any $0 \leq j \leq (n-1)$, then in the next step the neurons $\sigma_{c1}, \sigma_{c2}, \ldots, \sigma_{cn}$ receive two spikes. Subsequently, one spike is sent to the neurons $\sigma_{e11}, \sigma_{e21}, \ldots, \sigma_{en1}$ and $\sigma_{ei2}, \sigma_{ei3} (1 \leq i \leq n)$. In the next step $\sigma_{i2} (1 \leq i \leq n)$ spikes and one spike is sent to $\sigma_{dj} (1 \leq j \leq n)$. Neuron σ_{d1} stores the value of the summation $\sum_{i=0}^{m-1} y_0 x_i 2^{i+0}$. Similarly, σ_{d2} stores the value of the summation $\sum_{i=0}^{m-1} y_1 x_i 2^{i+i}$. Finally the n-th neuron stores the value $\sum_{i=0}^{m-1} y_n x_i 2^{i+n}$.

In Part 3, the output of the multiplication of any two binary numbers X and Y is computed, i.e., sum

of the binary numbers stored in $\sigma_{d1}, \sigma_{d2}, \ldots, \sigma_{dn}$ is calculated. If $i = 2k$ where $1 \leq k \leq [\frac{n}{2}]$ then the forgetting rule $a^i/a^{\frac{i}{2}} \rightarrow \lambda$ is applied. After application of this rule $\frac{i}{2}$ number of spikes are consumed and $\frac{i}{2}$ number of spikes remains inside the neuron σ_3 (This rule helps us to identify with sum of two 1's).

Similarly, the rule $a^j/a^{\frac{j+1}{2}} \rightarrow a$ where $j = 2k-1$ is an odd number and $1 \leq k \leq [\frac{n}{2}]$ helps us to relate with the sum of 1 and 0.

To clearly illustrate the operation process of the Mul unit, $(1110101)_2 \times (11101)_2 = (110101000001)_2$ is taken as an example. Table 3 shows the changes of the numbers of spikes for the key neurons in Mul unit and the output of neuron σ_3.

The total number of neurons in the Mul unit can be derived in the following manner.

Example 1.

Total number of neurons in Part 1 is $1 + 2 + 3 + \ldots + n + (n + 1) = \frac{(n+1)(n+2)}{2}$. Total number of neurons in Part 2 is $1[\sigma_2] + (n + 1)[\sigma_{b0}, \ldots, \sigma_{bn}] + n[\sigma_{c1}, \ldots, \sigma_{cn}] + n[\sigma_{e11}, \ldots, \sigma_{en1}] + 2n[\{\sigma_{e12}, \sigma_{e13}\}, \ldots, \{\sigma_{en2}, \sigma_{en3}\}] = 5n + 2$. Total number of neurons outside Part 1 and Part 2 is $n[\sigma_{d1}, \ldots, \sigma_{dn}] + 1[\sigma_3]$. So total number of neurons in Mul unit is $\frac{(n+1)(n+2)}{2} + 5n + 2 + n + 1 = \frac{n^2+15n+8}{2}$.

Table 3. Changes of the numbers of spikes in Mul unit

No. of steps	σ_1	σ_2	σ_{a11}	σ_{a22}	σ_{a33}	σ_{a44}	σ_{a55}	σ_{d1}	σ_{d2}	σ_{d3}	σ_{d4}	σ_{d5}	σ_3	output
0	0	0	0	0	0	0	0	0	0	0	0	0	0	-
1	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	-
2	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	-
3	**1**	**1**	1	0	0	0	0	0	0	0	0	0	0	-
4	**0**	**1**	0	1	0	0	0	2	0	0	0	0	0	-
5	**1**	**1**	1	0	1	0	0	1	1	0	0	0	1	-
6	**1**	0	0	1	0	1	0	2	0	2	0	0	0	**1**
7	**1**	0	1	0	1	0	1	1	1	1	2	0	2	**0**
8	0	0	1	1	0	1	0	2	0	2	1	2	2	**0**
9	0	0	1	1	1	0	1	2	1	1	2	1	4	**0**
10	0	0	0	1	1	1	0	2	1	2	1	2	4	**0**
11	0	0	0	0	1	1	1	0	1	2	2	1	5	**0**
12	0	0	0	0	0	1	1	0	0	2	2	2	4	**1**
13	0	0	0	0	0	0	1	0	0	0	2	2	5	**0**
14	0	0	0	0	0	0	0	0	0	0	0	2	4	**1**
15	0	0	0	0	0	0	0	0	0	0	0	0	3	**0**
16	0	0	0	0	0	0	0	0	0	0	0	0	1	**1**
17	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**

3.4. *Div unit*

The Div unit used in this paper is a modified version of the unit proposed in Ref.[12]. The Div unit in Ref.[12] has two drawbacks. Firstly, the correct quotient of any two signed non-zero integers cannot be obtained. Secondly, the output is usually not clear because the Div unit returns multiple outputs at different steps. Due to this second drawback, it is necessary to identify the correct quotient from the multiple sets according to the marked spike of neuron σ_s. With reference to the notation used for the Add unit, the simplified scheme of the Div unit highlighting the multiple outputs is shown in Fig. 5. In this case, the lengths of the two binary strings of both dividend and quotient is the same ($m = n$).

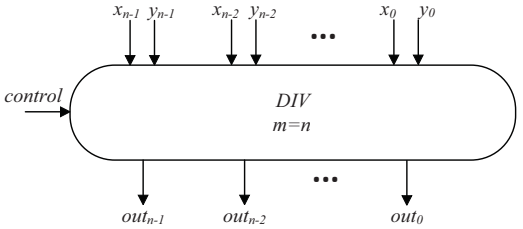

Fig. 5. Simplified version of Div unit.

The modified Div unit is shown in Fig. 6. The modified section is highlighted within the dotted frame where the correct quotient ranks at the top of the outputted sets. In other words, when the output neurons have spiking rules to apply, the first output is the quotient. Thus, the modified version avoids the identification process of several outputs.

Table 4. Changes of the numbers of spikes in Div unit

No. of steps	σ_{a0}	σ_{a1}	σ_{a2}	σ_{a3}	σ_{s1}	σ_{p0}	σ_{p1}	σ_{p2}	σ_{p3}	σ_{b4}	σ_8	out_0	out_1	out_2	out_3
1	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-
2	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-
3	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-
4	2	0	2	1	0	0	0	0	0	1	0	-	-	-	-
5	2	1	0	2	0	0	0	0	0	1	0	-	-	-	-
6	0	1	0	0	0	0	0	0	0	1	0	-	-	-	-
7	0	1	0	0	0	0	0	0	0	1	0	-	-	-	-
8	2	1	1	1	0	0	0	0	0	1	0	-	-	-	-
9	0	2	1	1	0	0	0	0	0	1	0	-	-	-	-
10	0	0	2	1	0	1	0	0	0	1	0	-	-	-	-
11	0	0	0	2	0	1	0	0	0	1	0	-	-	-	-
12	2	0	1	1	0	1	0	0	0	1	0	-	-	-	-
13	0	1	1	1	0	1	0	0	0	1	0	-	-	-	-
14	0	1	1	1	0	2	0	0	0	1	0	-	-	-	-
15	0	1	1	1	0	0	1	0	0	1	0	-	-	-	-
16	2	1	2	2	0	0	1	0	0	1	0	-	-	-	-
17	0	2	0	1	1	0	1	0	0	1	1	-	-	-	-
18	0	0	1	1	0	3	4	3	3	1	0	-	-	-	-
19	0	0	1	1	0	0	3	3	2	1	0	-	-	-	-
20	2	0	2	2	0	0	0	2	2	1	0	-	-	-	-
21	0	0	0	1	0	0	0	0	1	1	0	**0**	**1**	**0**	**0**

The scheme in Fig. 6 refers to a general model with $9n + 25$ neurons for calculating the quotient of two numbers x and y, namely x/y, with n bits, where $x = (x_{n-1}, \ldots, x_0)$ and $y = (y_{n-1}, \ldots, y_0)$ are signed nonzero binary integers. The values x_{n-1}, $\ldots$, x_0 and $y_{n-1}, \ldots, y_0$ are the inputs of neurons

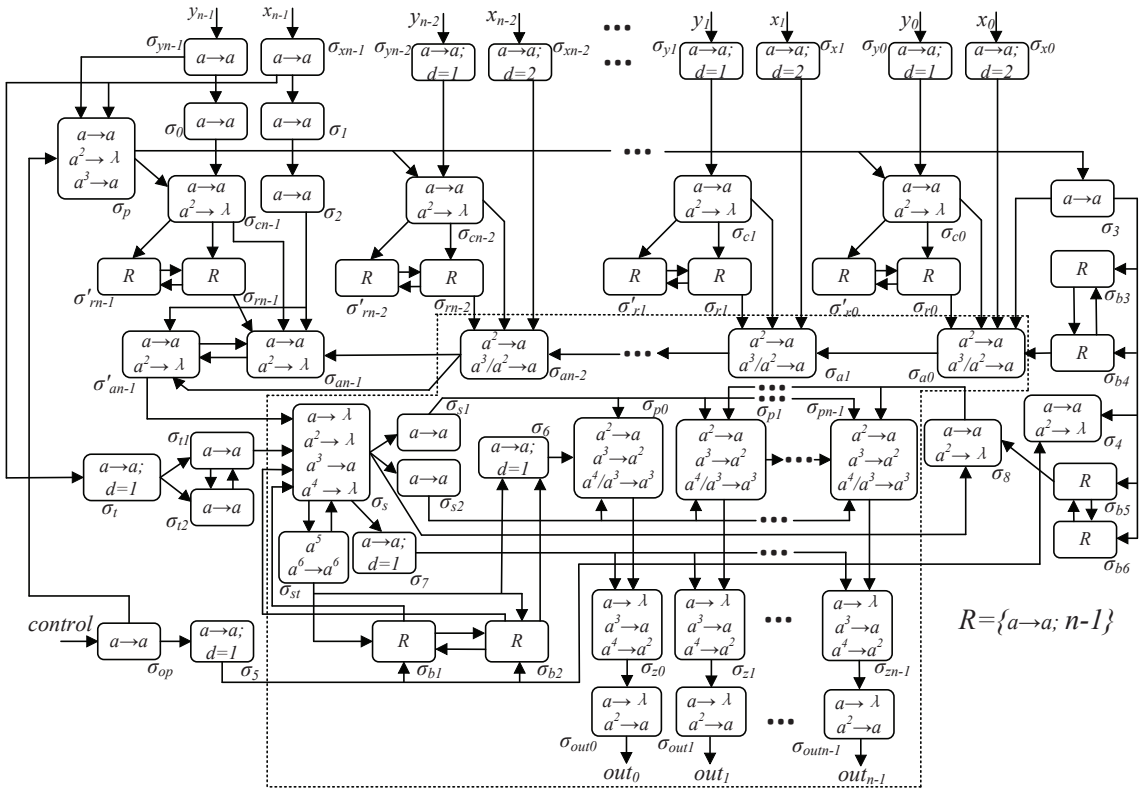

Fig. 6. Div unit.

$\sigma_{xn-1}, \ldots, \sigma_{x0}$ and $\sigma_{yn-1}, \ldots, \sigma_{y0}$, respectively. The result of the computation is collected through neurons $\sigma_{out0}, \ldots, \sigma_{outn-1}$ from lower to higher bits. Note that x_{n-1}, y_{n-1} and z_{n-1} are sign bits. Moreover, non-restoring division algorithm has been performed to calculate the quotient in the framework of SN P systems. This algorithm has advantages in hardware implementation.

Non-restoring division algorithm contains three registers Q, M and A where initially $Q =$ dividend $= x, M =$ divisor, $A = 0$. In the next step, the sign bit of the register A is checked. If it is 1, AQ is shifted left and A is updated as $A + M$ and similarly if it is 0, A is updated as $A - M$. This task is performed by the neurons $x_i(0 \leq i \leq n-1)$, $y_j(0 \leq j \leq n-1)$, $\sigma_{ci}, (0 \leq i \leq n-1)$, $\sigma'_{ri}, 0 \leq i \leq n-1$, $\sigma_{ai}, 0 \leq i \leq n-1, \sigma'_{an-1}$, $\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_p, \sigma_{b3}$ and σ_{b4}.

The neurons $\sigma_{x_i}(0 \leq i \leq n-1)$ and $\sigma_{y_j}(0 \leq j \leq n-1)$ are the input neurons. The delays $d = 2$ and $d = 1$ are associated with these neurons respectively. After receiving the inputs, these neurons cannot fire immediately. However, in the next step the control is activated and it further activates the neuron σ_p in the next step. Subsequently, σ_p sends one spike to the neurons $\sigma_{ci}(0 \leq i \leq n-1)$ and σ_3. At the same time the rules present in the neurons σ_{y_j} send one spike to the neurons $\sigma_{ci}, 0 \leq i \leq n-1$. In the next step, the neurons $\sigma_{ai}, 1 \leq i \leq n-1$ receive the spikes from σ_{ci} and σ_{x_i} for $1 \leq i \leq n-1$. Furthermore, σ_{a0} receives spikes from $\sigma_{c0}, \sigma_{x_0}$ and σ_3. This process is further continued by sending spikes from the neuron σ_{a0} to σ_{a1}, σ_{a1} to σ_{a2}, and eventually the neuron σ_{an-2} sends one spike to σ'_{an-1} and σ_{an-1} where these two neurons send spike to each other using the rule $a \to a$ and $a^2 \to \lambda$. Note that the neurons σ_0, σ_1 and σ_2 are introduced to adjust the delay $d = 1$ and $d = 2$ of the neurons $y_j(0 \leq j \leq n-2)$ and $x_i(0 \leq i \leq n-2)$ respectively.

In the next step of non-restoring algorithm, the sign bit of the register A is again checked. If the bit is 1, then $Q[0]$ is 0, otherwise it is 1. Next, N is decremented by 1, where $|Q|$ = number of bits in $Q = N$. If $N \neq 0$, then the previous steps are re-

peated. Otherwise, if the sign beat of A is 1, it is updated by $A + M$. This task is performed by the neurons $\sigma_t, \sigma_{t1}, \sigma_{t2}, \sigma_s, \sigma_{s1}, \sigma_{s2}, \sigma_{st}, \sigma_6, \sigma_7, \sigma_8, \sigma_{b5}, \sigma_{b6}$ and $\sigma_{pi}, 0 \leq i \leq n-1$.

After receiving the inputs in $\sigma_{x_{n-1}}$ and $\sigma_{y_{n-1}}$, one spike is sent to σ_p and $\sigma_{x_{n-1}}$ sends one spike to σ_t. In the next step, σ_p spikes according to the rules present inside it. If it contains 1 or 3 spikes then it sends one spike to $\sigma_{cj}, 0 \leq j \leq (n-1)$ and the above mentioned process is continued.

When the neuron σ_t receives one spike from $\sigma_{x_{n-1}}$, the rule $a \rightarrow a$ with delay 1 is applied. At $t = 3$, it sends spikes to σ_{t1} and σ_{t2}. Next, the neuron σ_{t1} sends one spike to σ_s. At the same time $\sigma_{an-1}^{'}$ sends one spike to σ_s. Also the supply of spikes to the neuron σ_s is continued by the neurons σ_{t1} and σ_{t2} and $\sigma_{an-1}^{'}$ and σ_{an-1} by sending one spike to each other.

Note that the inputs received in the neurons are initially processed in $\sigma_{aj}, (0 \leq j \leq n-1)$ and $\sigma_{an-1}^{'}$. Again the processed input is sent to the neuron σ_s which further sends it to $\sigma_{p_0}, \sigma_{p_1}, \ldots, \sigma_{p_{n-1}}$ and further processed using the rules present in the system.

Finally, in the non-restoring algorithm the register Q contains the quotient. In the framework of SN P systems, the neurons $\sigma_{out0}, \sigma_{out1}, \ldots, \sigma_{outn-1}$ and $\sigma_{zi}, 0 \leq i \leq n-1$ process the spikes received from the neurons in the previous steps and the output is collected depending on the spiking/ non-spiking of the neurons in the neurons $\sigma_{out0}, \sigma_{out1}, \ldots, \sigma_{outn-1}$. More specifically, the neurons σ_7 and $\sigma_{pj}, 0 \leq j \leq (n-1)$ spike at the same time and send spikes to the neurons $\sigma_{zj}, 0 \leq j \leq (n-1)$. The neurons in these neurons contain three types of rules, i.e., $a \rightarrow \lambda, a^3 \rightarrow a$ and $a^4 \rightarrow a^2$ and they are applied non-deterministically and send one, two or zero spikes to the neurons $\sigma_{out_j}, 0 \leq j \leq (n-1)$. Moreover, these neurons contain only the rules $a \rightarrow \lambda$ and $a^2 \rightarrow a$ and non-deterministic application of these rules can obtain the output of the division of two binary numbers.

To clearly illustrate the operation functioning of the Div unit, $(0100)_2/(0010)_2 = (0010)_2$ is taken as an example. At the beginning, $y = (0010)_2$ and $x = (0100)_2$ are considered as the inputs of neurons, i.e. $y_0 = 0$, $y_1 = 1$, $y_2 = 0$, $y_3 = 0$, $x_0 = 0$, $x_1 = 0$, $x_2 = 1$ and $x_3 = 0$. Table 4 shows the changes of the numbers of spikes for the key neurons in Div unit and the output of neurons σ_{out0}, σ_{out1}, σ_{out2} and σ_{out3}.

3.5. *Storage unit*

Within the process of implementing the information fusion, the calculation of polynomials often requires complex arithmetic operations which involves multiple units. Thus, it is necessary to store the results of intermediate calculations, which could be regarded as the inputs to subsequent calculations. This paper proposes a novel storage unit based on SNPS. The storage unit can keep a certain number of binary strings and can update the stored values according to the input and control neurons. The simplified scheme of the storage unit is depicted in Fig. 7.

Fig. 8 displays the detailed model of the storage unit. In the storage unit, n binary bits for a binary number are considered as the inputs of neurons $\sigma_0, \ldots, \sigma_{n-1}$ from low to high bits, respectively. Then, three spikes are inputted into the neurons σ_c as the control signal. After two steps, n binary bits are saved into neurons $\sigma_{s0}, \ldots, \sigma_{sn-1}$, and the number is outputted at each step.

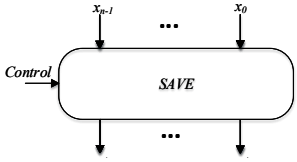

Fig. 7. Simplified version of storage unit.

3.6. *Simulator for SNPS*

As described in the Subsections above, Mul, Div and storage units contain dozens or even hundreds of neurons which perform a large number of calculations since each unit may contain many steps. Hence, a manual calculation of the SNPS is extremely impractical and a proper simulating software is required.

Fig. 9 describes the pseudocode of the novel SNPS simulator developed in this study to automatically accomplish the computation of arithmetic units. The simulator is made by considering neuron's behaviors such as receiving spikes, applying rules, spiking and sending spikes out of the neuron, saving spikes inside the neuron and sending spikes out of the neuron at every step. Furthermore, the simulator builds the connection between neu-

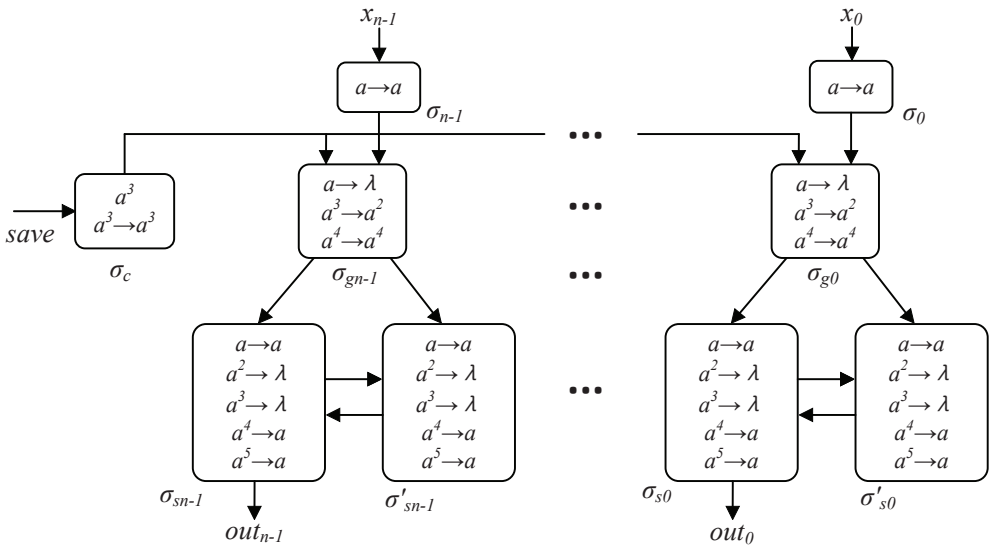

Fig. 8. Storage unit.

rons. Thus, the simulator can fulfill a computation of an arithmetic unit. The simulator discussed in the paper is different from the simulators present in the literature because the SN P systems used in this work contain a special variant of the forgetting rule of the form $a^m/a^n \to \lambda$ where after application of this rule n spikes are consumed and $(m-n)$ remain inside the neuron. The neurons in the arithmetic unit can be divided into three components, i.e., input, processing and output. The neurons in the input component receive input in the form of spikes. After receiving the inputs they are stored inside the neurons. Next, the neurons process the inputs $(n_1, n_2, \ldots, n_t)$ in parallel using the spiking rules $E/a^c \to a; d$ and forgetting rules $a^m/a^n \to \lambda$.

Input: $(n_1, n_2, \ldots, n_t), n_i \in \{0,1\}, t =$ Number of input neurons
1: Receive the spikes in input neurons $\sigma_i (1 \leq i \leq t)$
2: Save the spikes in the input neurons $\sigma_i (1 \leq i \leq t)$
3: Perform the steps 4-12 in all the neurons at any time instance in parallel
4: **while** a rule can be applied **do**
5: **if** the delay is null: $d == 0$ **then**
6: Apply spiking or forgetting rule and save the new spikes
7: **else**
8: **while** the delay is not null: $d \neq 0$ **do**
9: $d = d - 1$
10: **end while**
11: **end if**
12: **end while**
13: Send spikes outside the neuron
Output: $(n_1, n_2, \ldots, n_s), n_i \in \{0,1\}, s =$ Number of output neurons

Fig. 9. Pseudocode of the SNPS simulator

At any instance, the applicability of the rules are checked. If $d = 0$, then the spiking rules are applied immediately. When $d \neq 0$, then the delay is reduced by 1 in every subsequent steps, until $d = 0$ is obtained. Furthermore, all these operations happen in parallel. Finally, at any time instance depending on the applicability of the rules in output neurons, the output $(n_1, n_2, \ldots, n_s)$ is obtained.

4. Information fusion implementation

In this section, the theoretical aspects of information fusion are outlined and then, the implementation of information fusion for the proposed arithmetic calculator is described in details. More specifically, we constructed a method based on spiking neural P systems to calculate basic probability assignment (BPA) m of an event A where m can be derived by fusing two BPAs m_1 and m_2. The novel idea of this paper is that the algebraic operations addition, substation, multiplication and division of any two binary numbers can be derived using spiking neural P systems, i.e., the spiking neural P systems constructed in this paper can perform addition, subtraction, multiplication and division and for any given input, desired output can be obtained . Since any integer can be represented as a string over 0 and 1, the value $m(A)$ also can be derived using the spiking neural P systems. We have also showed

that BPA m of any event X can be represented as $\frac{m_1(X)m_2(X)}{1+2m_1(X)m_2(X)-(m_1(X)+m_2(X))}$.

4.1. *Dempster-Shafer evidence theory*

Dempster-Shafer (D-S) evidence theory, which was first proposed by A. Dempster [14] and popularized by his student G. Shafer [51], is a method for information fusion characterised by the following definition.

Definition 2. Let the set Θ contains finite number of elements where the elements can be a hypothesis, an object etc. In this paper, Θ is a set containing distinct hypotheses and it is called *frame of discernment*[51]. The set of all the possible subsets of Θ is said *power set* of Θ and is indicated with 2^{Θ}.

Definition 3.

Let Θ is a discernment frame. The *Basic Probability Assignment* (BPA) is a function m: $2^{\Theta} \rightarrow [0, 1]$, which satisfies the following conditions:

$$\sum_{A \subseteq \Theta} m(A) = 1; \quad m(\phi) = 0.$$

$m(A)$ is called *basic probability number* expressing a belief measure of the proportion A. If $m(A) > 0$, then A is said *focal element*.

Definition 4. Let m_1 and m_2 be BPA based on the same frame of discernment Θ, and $A_1, \ldots, A_n$ and $B_1, \ldots, B_l$ be focal elements. The *rules of evidence combination* are described as follows:

$$m(A) = \begin{cases} \dfrac{\sum\limits_{A_i \cap B_j = A} m_1(A_i)m_2(B_j)}{1-K} \text{ if } A \neq \phi \\ 0 \text{ if } A = \phi. \end{cases}$$

$$\sum_{A \subseteq \Theta} m(A) = 1$$

where $m(A)$ is the BPA which fuses m_1 and m_2; $K = \sum_{A_i \cap B_j = \emptyset} m_1(A_i)m_2(B_j)$ represents the amount of conflict between m_1 and m_2.

In this paper, the fusion of two evidences m_1 and m_2 is obtained by using SNPS. m_1 and m_2 represent the basic probability assignment to two events X and Y respectively. The BPA sum of two events equals 1, i.e., $m_1(X) + m_1(Y) = 1$, $m_2(X) + m_2(Y) = 1$.

By applying the rules of evidence combination in Definition 4 the fused BPA of m_1 and m_2 is represented as

$$m(X) = \frac{m_1(X)m_2(X)}{1 - m_1(X)m_2(Y) - m_1(Y)m_2(X)}$$

$$= \frac{m_1(X)m_2(X)}{1 + 2m_1(X)m_2(X) - (m_1(X) + m_2(X))}.$$

(using $m_2(Y) = 1 - m_2(X), m_1(Y) = 1 - m_1(X)$)

4.2. *Information fusion with SNPS*

In this subsection, the information fusion shown in Subsection 4.1 is realized by integrating Add, Sub, Mul and Div units and three storage units. The integration system is shown in Fig. 10, where $m_1(x)$ and $m_2(x)$ are the inputs to Add and Mul units through the $input_x$ and $input_y$ at the same time, respectively.

As shown in Fig. 10, the system for implementing information fusion consists of five parts designated by dashed frames.

- **Part 1** is used to calculate the product of $m_1(x)$ and $m_2(x)$, and also save the product by using storage unit Mul_out.
- **Part 2** moves one bit to left of the calculated result of Part 1, that is, the computation of $2m_1(x)m_2(x)$ is performed by applying $m_1(x)m_2(x)$.
- The result of $1 + 2m_1(x)m_2(x)$ is gained by **Part 3** and further the result of $1 + 2m_1(x)m_2(x)$ is kept in storage unit $x1x2\_one\_out$.
- **Part 4** executes two steps operations: the calculation of the sum of $m_1(x)$ and $m_2(x)$ and the receipt of the result of $1 + 2m_1(x)m_2(x) - (m_1(x) + m_2(x))$. In addition, Part 4 also saves the result in storage unit add_out.
- **Part 5** is a Div unit for calculating the final division and outputting the fused BPA. It is worth pointing out that the Div unit in this paper can only calculate the quotient of any two signed nonzero integer numbers. On the other hand, the formula of fused BPA contains the decimal division. To make

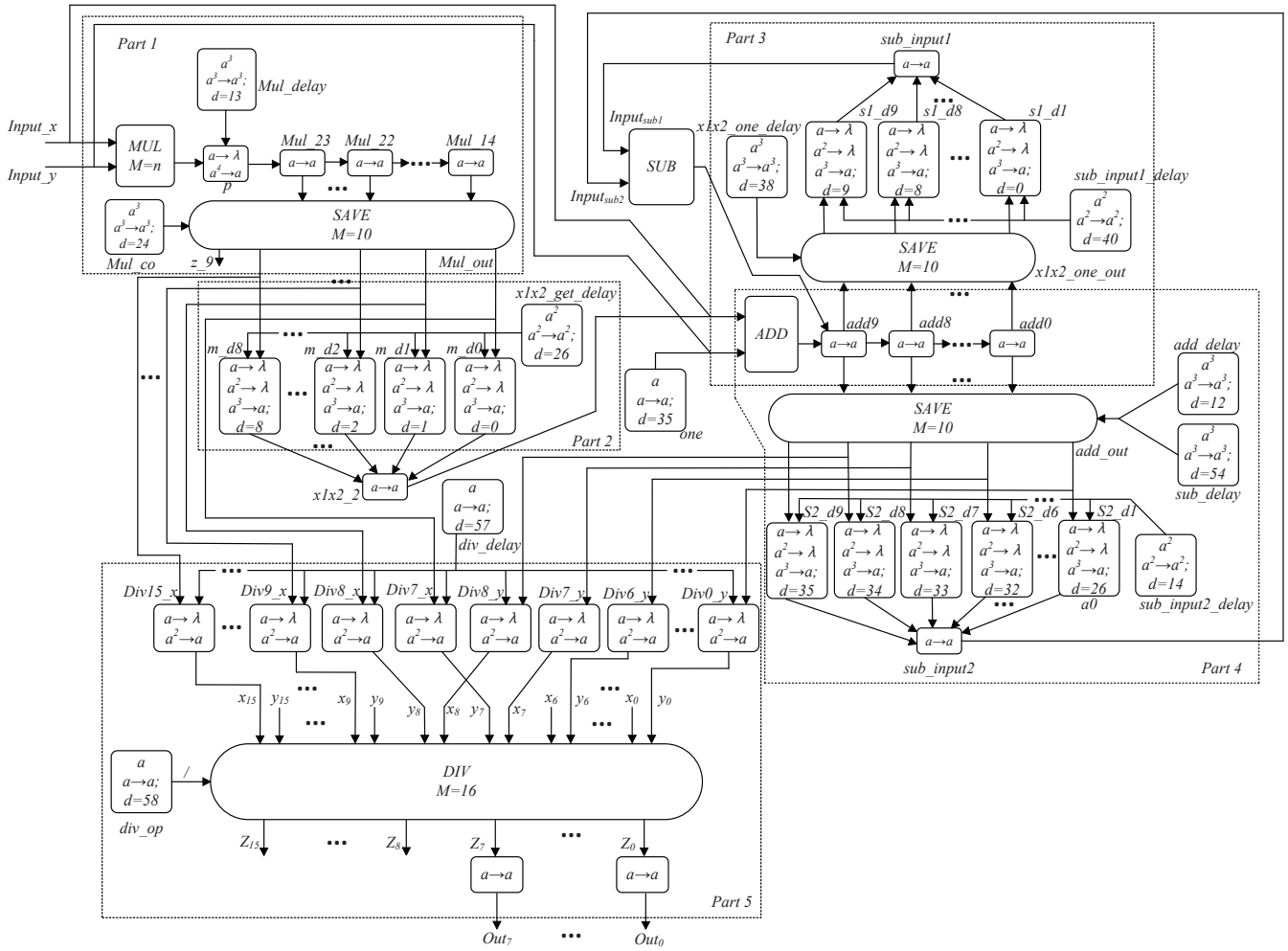

Fig. 10. Information fusion implemented by SNPS.

the system shown in Fig. 10 perform decimal divisions, we magnify the dividend 128 times, that is, we move 7 bits to left of the dividend. At the same time, to reduce the calculations steps, we discard the lowest bits of $1 + 2m_1(x)m_2(x) - (m_1(x) + m_2(x))$ and $m_1(x)m_2(x)$. The 16-bits Div unit is used in **Part 5**. The dividend x and divisor y are as inputs of σ_7, σ_8, ..., σ_{15} and σ_0, σ_1, ..., σ_8.

In this study, $m_1(x)$ **Part 5** and $m_2(x)$ are represented by ten binary bits and are inputted from lower to higher bits, respectively. For example, 0.75 can be represented as $(0.11000000)_2$, and the highest bit is sign bit. The result can be obtained from the output neurons $out_0, out_1, \ldots, out_7$ at step S, that is, $0.75 = out_7 \cdot 2^0 + out_6 \cdot 2^{-1} + out_5 \cdot 2^{-2} + \ldots + out_0 \cdot 2^{-7}$. It is worth noting that S is the first step that there are outputs of neurons out_0, ..., out_7.

As shown in Table 5, the first output values of out_0, ..., out_7 are correct. At Steps 1 to $S-1$, the symbol "-" in Table 5 means that the output neurons do not have spiking rules to apply and therefore they do not have any output. At Step S, the output neurons output the number $(0.11000000)_2$, as mentioned above, this is the first output values, so it is the correct result of the information fusion. Although at Steps $S+1$ and $S+2$ and so on, the output neurons output values, but they are not the first output numbers of out_0, ..., out_7. So they are

not the fusion result.

Table 5. Outputs of neurons $out_0, \ldots, out_7$

No. of steps	out_7	out_6	out_5	out_4	out_3	out_2	out_1	out_0
1	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-
...	-	-	-	-	-	-	-	-
$S-1$	-	-	-	-	-	-	-	-
S	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
$S+1$	0	1	1	0	1	0	0	1
$S+2$	1	0	1	1	0	0	0	0

4.3. *Numerical Examples*

Since there are many neurons in the system for information fusion with SNPS, this paper uses the simulator described in Section 3.6 to build the model of information fusion with SNPS and to verify its correctness. Table 6 shows three cases as examples. We can easily know that the information fusion results are correct. Moreover, the first example listed in the first row in Table 6 is taken as an example to briefly introduce the calculation process.

First of all, the binary sequence 0000001100 for representing $m_1(x)$= $(00.11000000)_2$ is inputted by $Input\_x$ and the binary sequence 0100101100 for representing $m_2(x) = (00.11010010)_2$ is inputted by $Input\_y$. Then, from Steps 6 to 21, Mul unit will output binary sequence 1001110110000000, which represents the value of $(1100000)_2 \times (11010010)_2$, because Mul unit can calculate only signed nonzero integer numbers, so in fact, the outputted binary sequence represents $m_1(x)m_2(x) \times 2^{16}$.

Table 6. Three numerical examples

Data		$m_1(x)$	$m_2(x)$	$m(x)$
First set	Decimal	0.75	0.82	0.93
	Binary	00.11000000	00.11010010	0.1110111
Second set	Decimal	0.3	0.91	0.81
	Binary	00.01001101	00.11101001	0.1101000
Third set	Decimal	0.82	0.87	0.97
	Binary	00.11010010	00.11011111	0.1111100

Considering the system size, the precision 2^{-8} is considered, so we save the sequence 0010011101 which represents $m_1(x)m_2(x)$ to the storage unit Mul_out. Add unit outputs the binary sequence 0110010010 from Steps 3 to 12, which represents the sum of $m_1(x)$ and $m_2(x)$. Part 2 moves one bit to left of the $m_1(x)m_2(x)$, which produces the result of $2m_1(x)m_2(x)$ from the result of $m_1(x)m_2(x)$. From Steps 28 to 37, the neuron $x1x2\_2$ outputs the binary sequence 0100111010, which represents $2m_1(x)m_2(x)$ to Add unit, and the Add unit will calculate $1 + 2m_1(x)m_2(x)$ and outputs the binary sequence 1000111010 from Steps 29 to 38. Then, the Sub unit receives binary sequences from neurons sub_input1 and sub_input2 representing $1 + 2m_1(x)m_2(x)$ and $m_1(x) + m_2(x)$. Sub unit will output binary sequence 0010101000 of $2m_1(x)m_2(x) - (m_1(x) + m_2(x))$ from Steps 45 to 54 and save it to storage unit add_out. Finally, the Div unit will calculate the quotient of $m_1(x)m_2(x)$ and $1 + 2m_1(x)m_2(x) - (m_1(x) + m_2(x))$, and output the result 11101110 at Step 1986 by neurons $out_0, \ldots, out_7$, which represents the binary number $(0.1110111)_2$.

4.4. *Applicability of the proposed Arithmetic calculator in the real-world*

The proposed calculator can be viewed as the building block of complex devices, such as membrane controllers to perform navigation tasks on board of autonomous mobile robots in complex environments [59]. Numerical P systems [43]/ enzymatic numerical P systems are generally used for this purpose, see Ref. [47]. In these cases, the task of robot navigation is performed by the membrane controller which helps to identify the different environment classifier for smooth navigation of the robots. One of the main drawbacks of numerical P systems has been that the cells can communicate with the only upper and lower membranes. This feature restricts the capability of membrane controllers of identifying complex environment classifiers. Unlike Numerical P Systems, SN P systems do not have such restrictions and arithmetic operations also can be performed. Hence spiking neural P systems can be considered as a better framework to model membrane controllers and environment classifiers.

Numerical P systems can perform the simple arithmetic operations such as addition, subtraction, multiplication with less numbers of candidates efficiently and quickly. However, a large number of cells are required for to perform these operations in large scale. SN P systems can perform large scale operations with much lower number of neurons thanks to its parallel distributed architecture and communication rules.

In order to have an understanding of the potential of the proposed arithmetic calculator, let us consider two numbers x_1 and x_2 of n and m digits

respectively:

$$|x_1| = n$$
$$|x_2| = m$$

with $n \leq m$. In order to perform the multiplication the numerical P System requires at least n^2 cells. On the other hand, the multiplication unit of Section 3.3 requires at least

$$\frac{n^2 + 13n + 6}{2}$$

neurons. We can easily calculate that

$$n^2 - \frac{n^2 + 13n + 6}{2} \geq 0$$

for $n \geq 14$. Hence, the SNPS requires a smaller number of neurons for arithmetic operations with multiple bits.

5. Conclusion

This paper proposes a new arithmetic calculator based on SNPS. This calculator makes uses of existing Add and Sub units, proposes modified versions of Mul and Div units and proposes a novel storage unit based on SNPS. Furthermore this study implements the information fusion coming by the five units by applying the Dempster-Shafer evidence theory. Moreover, during the implementation of information fusion, the framework combining the SNPS and D-S evidence theory calculates the basic probability assignment (BPA) of an event. Finally, we developed a simulator to validate the correctness of information fusion with SNPS. Numerical experiments verify the corrected of the proposed calculator.

Future work will include two new designs. The first is the design of SNPS Mul and Div units with the capability of changing their sizes in response to the input numbers. The second is the design of a simplified Div unit since the current state-of-art SNPS Div unit is remarkably complex. Finally, the reset function for the Div unit is also an ongoing issue.

Acknowledgment

The work of the first four authors is supported by the National Natural Science Foundation of China (61972324, 61672437,61702428), by Beijing Advanced Innovation Center for Intelligent Robots and Systems (2019IRS14), New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0043) and Artificial Intelligence Key Laboratory of Sichuan Province (2019RYJ06). The work of the last author was supported by the research project TIN2017-89842-P (MABICAP), co-financed by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union.

References

1. H. Adeli, *Parallel Processing in Computational Mechanics* (Marcel Dekker, New York, 1992).
2. H. Adeli, *Supercomputing in Engineering Analysis* (Marcel Dekker, New York, 1992).
3. H. Adeli, High-performance computing for large-scale analysis, optimization, *Journal of Aerospace Engineering, ASCE* **13**(1) (2000) 1–10.
4. H. Adeli and S. Ghosh-Dastidar, *Automated EEG-based Diagnosis of Neurological Disorders - Inventing the Future of Neurology* (CRC Press, Taylor & Francis, Boca Raton, Florida, 2007).
5. H. Adeli and S. Kumar, Concurrent structural optimization on a massively parallel supercomputer, *Journal of Structural Engineering, ASCE* **121**(11) (1995) 1588–1597.
6. A. Antonietti, J. Monaco, E.ĎAngelo, A. Pedrocchi and C. Casellato, Dynamic redistribution of plasticity in a cerebellar spiking neural network reproducing an associative learning task perturbed by tms, *International Journal of Neural Systems* **28**(9) (2018).
7. G. Antunes, S. F. D. Silva, and F. S. D. Souza, Mirror neurons modeled through spike-timing dependent plasticity are affected by channelopathies associated with autism spectrum disorder, *International Journal of Neural Systems* **28**(5) (2018) p. 1750058.
8. O. Basir and X. Yuan, Engine fault diagnosis based on multi-sensor information fusion using dempster–shafer evidence theory, *Information Fusion* **8** (2007) 379–386.
9. M. Bernert and B. Yvert, An attention-based spiking neural network for unsupervised spike- sorting, *International Journal of Neural Systems* **29**(8) (2019) p. 1850059.
10. J. Carandang, J. Villaflores, F. Cabarle, H. Adorna and M. M. del Amor, CuSNP: Spiking neural P systems simulators in CUDA, *Romanian Journal of Information Science and Technology* **20**(1) (2017) 57–70.
11. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, G. Păun and M. J. Pérez-Jiménez, Spiking neural P systems with extended rules: universality and languages, *Natural Computing* **7**(2) (2008) 147–166.
12. Y. Chen, G. Zhang and X. Huang, Automatic design

of a P system for basic arithmetic operations, *Asian Conference on Membrane Computing*, 2012, pp. 124–138.
13. R. T. A. de la Cruz, F. G. Cabarle and H. N. Adorna, Generating context-free languages using spiking neural P systems with structural plasticity, *Journal of Membrane Computing* **1**(3) (2019) 161–177.
14. A. Dempster, Upper and lower probabilities induced by multivalued mapping., *Ann. Math. Stat* **38** (1967) 325–339.
15. C. Diaz, T. Frias, G. Sanchez, H. Perez, K. Toscano and G. Duchen, A novel parallel multiplier using spiking neural P systems with dendritic delays, *Neurocomputing* **239**(C) (2017) 113–121.
16. R. Fay, F. Schwenker, C. Thiel and G. Palm, Hierarchical neural networks utilising dempster-shafer evidence theory, *ANNPR 2006,Springer-Verlag Berlin Heidelberg*, 2006, pp. 198–209.
17. R. Freund, G. Păun and M. J.Pérez-Jiménez, Tissue P systems with channel states, *Theoretical Computer Science* **330**(1) (2005) 101–116.
18. F. Galan-Prado, J. Font and J. Rossello, Compact hardware synthesis of stochastic spiking neural networks international journal of neural systems, *International Journal of Neural Systems* **29**(8) (2019) p. 1950004.
19. X. Gao and H. Z. Chen, Signed integer arithmetic on spiking neural P system, *Applied Mechanics and Materials* **20-23** (2010) 779–784.
20. A. Geminiani, C. Casellato, A. Antonietti, E. ĎAngelo and A. Pedrocchi, A multiple-plasticity spiking neural network embedded in a closed-loop control system to model cerebellar pathologies, *International Journal of Neural Systems* **28**(5) (2018) p. 1750017.
21. S. Ghosh-Dastidar and H. Adeli, Improved spiking neural networks for EEG classification and epilepsy and seizure detection, *Integrated Computer-Aided Engineering* **14**(3) (2007) 187–212.
22. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *International Journal of Neural Systems* **19**(4) (2007) 295–308.
23. S. Ghosh-Dastidar and H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, *Neural Networks* **22**(10) (2009) 1419–1431.
24. P. Guo, H. Chen and H. Zheng, Arithmetic expression evaluations with membranes, *Chinese Journal of Computers Electron* **23**(1) (2014) 55–60.
25. P. Guo, H. Zhang, H. Z. Chen and J. Chen, Fraction arithmetic operations performed by P systems, *Chinese Journal of Electronics* **22**(4) (2013) 689–694.
26. R. Hu, S. Chang, Q. Huang, H. Wang and J. He, Monitor-based spiking recurrent network for the representation of complex dynamic patterns, *International Journal of Neural Systems* **29**(8) (2019) p. 1950006.
27. M. Ionescu, G. Păun and T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae* **71**(2–3) (2006) 279–308.
28. Y. Jiang, Y. Su and F. Luo, An improved universal spiking neural P system with generalized use of rules, *Journal of Membrane Computing* **1**(4) (2019) 270–278.
29. Z. B. Jimenez, F. G. C. Cabarle, R. T. A. de la Cruz, K. C. Buño, H. N. Adorna, N. H. S. Hernandez and X. Zeng, Matrix representation and simulation algorithm of spiking neural P systems with structural plasticity, *Journal of Membrane Computing* **1**(3) (2019) 145–160.
30. S. Li, G. Liu, X. Tang, J. Lu and J. Hu, An ensemble deep convolutional neural network model with improved D-S evidence fusion for bearing fault diagnosis, *Sensors (Basel)* **17**(8) (2017) p. PMID: 28788099.
31. S. S. Liu, Q. S. Dou and K. Fu, Broadcast model based on membrane computing system, *Computer Engineering* **38**(4) (2012).
32. X. Liu, Z. Li, J. Liu, L. Liu and X. Zeng, Implementation of arithmetic operations with time-free spiking neural P systems, *IEEE Transactions on NanoBioscience* **14**(6) (2015) 617 – 624.
33. J. Luan and X. Liu, Arithmetic operation in spiking neural P system with chain structure, *WSEAS Transactions on Computers* **12**(6) (2013) 65–75.
34. L. Macías, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia, M. Pérez-Jiménez and A. Riscos-Núnez, A P-lingua based simulator for spiking neural P systems, *Lecture Notes in Computer Science*, 2012, pp. 257–281.
35. V. P. Metta, K. Krithivasan and D. Garg, Computability of spiking neural P systems with anti-spikes, *New Mathematics and Natural Computation* **8**(3) (2012) 283–295.
36. G. Naranjo, M. Ángel and A. Leporati, Performing arithmetic operations with spiking neural P systems, *Proceedings of the Seventh Brainstorming Week on Membrane Computing*, **I,**2009, pp. 181–198.
37. O. Ochirbat, T.-O. Ishdorj and G. Cichon, An error-tolerant serial binary full-adder via a spiking neural P system using HP/LP basic neurons, *Journal of Membrane Computing* **2** (2020) 42–48.
38. D. Orellana-Martín, L. Valencia-Cabrera, A. Riscos-Núñez and M. J. Pérez-Jiménez, Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems, *Journal of Membrane Computing* **1**(2) (2019) 85–92.
39. L. Pan and Gh. Păun, Spiking neural P systems: An improved normal form, *Theoretical Computer Science* **411**(6) (2010) 906–918.
40. L. Pan, A. Alhazov and T.-O. Ishdorj, Further remarks on P systems with active membranes, separation, merging, and release rules, *Soft Computing* **9**(9) (2005) 686–690.
41. L. Pan, G. Păun and G. Zhang, Foreword: Starting JMC, *Journal of Membrane Computing* **1** (Mar 2019) 1–2.
42. L. Pan, G. Păun, G. Zhang and F. Neri, Spiking neural P systems with communication on request, *International Journal of Neural Systems* **27**(8) (2017) p. Article No. 1750042.
43. G. Păun and R. Păun, Membrane computing and eco-

nomics: Numerical p systems, *Fundamenta Informaticae* **73** (2006) 213–227.
44. G. Păun, Computing with membranes, *Journal of Computer and System Sciences* **61**(1) (2000) 108–143.
45. G. Păun, G. Rozenberg and A. Salomaa, *The Oxford Handbook of Membrane Computing* (Oxford University Press, Inc., New York, NY, USA, 2010).
46. X. Peng, X. Fan, J. Liu and H. Wen, Spiking neural P systems for performing signed integer arithmetic operations., *Journal of Chinese Computer Systems* **34**(2) (2013) 360–364.
47. I. Pérez-Hurtado, M. Á. Martínez-del-Amor, G. Zhang, F. Neri and M. J. Pérez-Jiménez, A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning, *Integr. Comput. Aided Eng.* **27**(2) (2020) 121–138.
48. F. Ponulak and A. Kasinski, Introduction to spiking neural networks: information processing, learning and applications, *Acta Neurobiologiae Experimentalis* **71**(4) (2011) 409–433.
49. G. Păun, M. J. Pérez-Jiménez and G. Rozenberg, Spike trains in spiking neural P systems, *International Journal of Foundations of Computer Science* **17**(4) (2006) 975–1002.
50. H. Rong, K. Yi, G. Zhang, J. Dong, P. Paul and Z. Huang, Automatic implementation of fuzzy reasoning spiking neural P systems for diagnosing faults in complex power systems, *Complexity* **2019** (2013) Article ID 2635714, 16 pages.
51. G. Shafer, *A Mathematical Theory of Evidence* (Princeton University Press, 1976).
52. T. Song, L. Pan and Gh. Păun, Asynchronous spiking neural P systems with local synchronization, *Information Sciences* **219** (2013) 197–207.
53. T. Song, L. Pan and G. Păun, Asynchronous spiking neural P systems with local synchronization, *Information Sciences* **219** (2013) 197–207.
54. S. Stefan, K. Nikola and D.-P. Michael, On the probabilistic optimization of spiking neural networks, *International Journal of Neural Systems* **20**(6) (2010) 481–500.
55. Y. Todo, Z. Tang, H. Todo, J. Ji and K. Yamashita, Neurons with multiplicative interactions of nonlinear synapses, *International Journal of Neural Systems* **29**(8) (2019) p. Article No. 1950012.
56. H. Wang, K. Zhou and G. Zhang, Arithmetic operations with spiking neural P systems with rules and weights on synapses., *International Journal of Computers, Communications & Control* **13**(4) (2018,) 574–589.
57. T. Wang, G. Zhang, J. Zhao, Z. He, J. Wang and M. J. Pérez-Jiménez, Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems, *IEEE Transactions on Power Systems* **30** (ISSN 0885–8950, 2015) 1182–1194.
58. X. Wang, G. Zhang, F. Neri, T. Jiang, J. Zhao, M. Gheorghe, F. Ipate and R. Lefticaru, Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots, *Integrated Computer-Aided Engineering* **23**(1) (2016) 15–30.
59. X. Wang, G. Zhang, F. Neri, T. Jiang, J. Zhao, M. Gheorghe, F. Ipate and R. Lefticaru, Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots, *Integr. Comput. Aided Eng.* **23**(1) (2016) 15–30.
60. T. Wu, F. Bilbie, A. Paun, L. Pan and F. Neri, Simplified and yet turing universal spiking neural P systems with communication on request, *International Journal of Neural Systems* **28**(8) (2018).
61. H. Xiang and D. Wang, The application of genetic bp neural network and d-s evidence theory in the complex system fault diagnosis, *Recent Advances in Computer Science and Information Engineering, Springer, Berlin, Heidelberg*, 2012, pp. 219–224.
62. R. Yang, P. Guo, J. Li and P. Gu, Arithmetic P systems based on arithmetic formula tables, *Chinese Journal of Electronics* **24**(3) (2015) 542–549.
63. X. Zeng, T. Song, X. Zhang and L. Pan, Performing four basic arithmetic operations with spiking neural P systems, *IEEE Transactions on Nanobioscience* **11** (ISSN 1558–2639 2012) 366–374.
64. G. Zhang, M. J. Pérez-Jiménez and M. Gheorghe, *Real-life Applications with Membrane Computing: Emergence, Complexity and Computation* (Springer International Publishing, 2017).
65. G. Zhang, H. Rong, F. Neri and M. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, *International Journal of Neural Systems* **24** (ISSN 1793–6462 2014) 1–16.
66. G. Zhang, J. Cheng, M. Gheorghe and Q. Meng, A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, *Applied Soft Computing* **13**(3) (2013) 1528–1542.
67. G. Zhang, M. Gheorghe, L. Pan and M. J. Pérez-Jiménez, Evolutionary membrane computing: A comprehensive survey and new results, *Information Sciences* **279** (2014) 528–551.
68. X. Zhang, G. Foderaro, C. Henriquez and S. Ferrari, A scalable weight-free learning algorithm for regulatory control of cell activity in spiking neuronal networks, *International Journal of Neural Systems* **28**(2) (2018) p. 1750015.
69. X. Zhang, X. Zeng, L. Pan and B. Luo, A spiking neural P system for performing multiplication of two arbitrary natural numbers, *Chinese Journal of Computers* **32** (ISSN 0254–4164 2009) 2362–2372.
70. X. Zhang, Y. Liu, B. Luo and L. Pan, Computational power of tissue P systems for generating control languages, *Information Sciences* **278** (2014) 285–297.