INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 13(4), 574-589, August 2018.

# Arithmetic Operations with Spiking Neural P Systems with Rules and Weights on Synapses

H.F. Wang, K. Zhou, G.X. Zhang

**Huifang Wang, Kang Zhou***
Department of Math and Computer
Wuhan Polytechnic University,
430023 Wuhan, Hubei, China
*Corresponding author: zhoukang65@whpu.edu.cn
13667286986@139.com

**Gexiang Zhang**
1. School of Electrical Engineering
Southwest Jiaotong University, Chengdu, China
2. Robotics Research Center
Xihua University, Chengdu, China
zhgxdylan@126.com

**Abstract:** The application of spiking neural P systems with rules and weights on synapses to arithmetic operations is discussed in this paper. We design specific spiking neural P systems with rules and weights on synapses for successfully performing addition, multiplication and the greatest common divisor. This is the first attempt to discuss the application of the new variant of spiking neural P systems, spiking neural P systems with rules and weights on synapses, and especially the use of spiking neural P systems to perform the greatest common divisor. Comparing with the results reported in the literature, smaller number of neurons are required to fulfill the arithmetic operations.
**Keywords:** SN P systems, rules and weights on synapses, addition, multiplication, the greatest common divisor.

## 1 Introduction

Membrane computing was initiated in [12]. The distributed and parallel computing devices in membrane computing are called P systems [13]. Some applications of P systems were reported in [2, 28, 29]. Especially, applications of P systems to arithmetic operations were discussed in [1, 2, 4, 31]. Some views on membrane computing were given in [3, 14]. For more information about membrane computing, one can refer to [15, 33]. Spiking neural P systems (in short, SN P systems) are a variant of P systems, which were introduced in [5] as a class of parallel and distributed computing models.

In recent years, SN P systems have been widely investigated in theory and applications. For instance, various variants of SN P systems were constructed by considering different biological sources [9,11,17,19,20,22,27] and the computing power/computational efficiency of some variants was discussed [10,18,26]. A wide range of applications of SN P systems was also presented, such as optimization [30], knowledge representation [23], fault diagnosis [16,24], and logical gates and circuits [6, 21]. SN P systems have been also used to perform arithmetic operations [8, 25, 32]. In order to answer the open problem in [8], SN P systems with a single input neuron for the addition and multiplication were constructed in [32]. These studies indicate that arithmetic/logic operations are promising applications of SN P systems.

In this paper, the application of SN P systems with rules and weights on synapses (RWSN P systems) is discussed to perform arithmetic operations. Smaller number of neurons are required to

construct the SN P systems for addition of $n$ natural numbers and multiplication of two arbitrary natural numbers compared with the ones in [32]. This is the first attempt to discuss the use of spiking neural P systems to perform the greatest common divisor. We divide the solution for the greatest common divisor into several individual modules, and the binary representation method in [34] is used here. The inputs and outouts of these systems are natural numbers expressed in binary form, which are encoded as appropriate sequences of spikes.

This paper is organized as follows. Section 2 briefly introduces the SN P systems with rules and weights on synapses. In Section 3, SN P systems with rules and weights on synapses are used to perform arithmetic operations. Conclusions and future work are presented in Section 4.

## 2 Spiking neural P Systems with rules and weights on synapses

In this section, SN P systems with rules and weights on synapses are briefly described. More details can be referred to [7].

Such a system of degree $m \geq 1$ is a construct of the form:

$$\prod = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, \sigma_{in}, \sigma_{out}),$$

where:

- $O = \{a\}$ is the singleton alphabet ($a$ is called spike);
- $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (n_i)$, with $1 \leq i \leq m$, where $n_i$ is initial number of spikes in neuron $\sigma_i$;
- $syn$ is the set of synapses; each element in $syn$ is a pair of the form $((i,j), w, R_{(i,j)})$, where $(i,j)$ indicates that there is a synapse connecting neurons $\sigma_i$ and $\sigma_j$, with $i, j \in \{1, 2, \ldots, m\}, i \neq j$; $w \in N(w \neq 0)$ is the weight on synapse $(i,j)$; $R_{(i,j)}$ is a finite set of rules of the following two forms:
  - (1) $E/a^c \to a; d$, where $E$ is a regular expression over $O$, $c \geq 1$ and $d \geq 0$;
  - (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^c \to a; d$ from any $R_{(i,j)}$;
- $\sigma_{in}$ and $\sigma_{out}$ indicates the input neuron and the output neuron, respectively.

A rule $E/a^c \to a; d$ is called a standard firing rule; if $L(E) = \{a^c\}$, then the rule can be written in the simplified form $a^c \to a; d$. A rule of the form $a^s \to \lambda$ is called a forgetting rule. The firing rules are applied as follows. If $E/a^c \to a; d \in R_{(i,j)}$, and neuron $\sigma_i$ contains $k$ spikes such that $a^k \in L(E)$, $k \geq c$, then the rule is enabled. This means consuming (removing) c spikes (thus only $k-c$ remain in $\sigma_i$) from neuron $\sigma_i$. Neuron $\sigma_i$ is fired, and it produces one spike. This one spike is multiplied $w$ times by the weight $w$ of the synapse in the process of transmission, and then reaches neuron $\sigma_j$ after $d$ time units (that is, $w$ spikes arrive at neuron $\sigma_j$). When neuron $\sigma_i$ contains exactly $s$ spikes, then forgetting rule $a^s \to \lambda \in R_{(i,j)}$ is enabled. By using it, $s \geq 1$ spikes are removed from the neuron $\sigma_i$.

In order to express the synapse between the output neuron and the environment, the environment is represented by the letter $e$, and the synapse between the output neuron and the environment can be expressed as $(out, e)$.

# 3 Arithmetic operations with SN P systems with rules and weights on synapses

In this section, SN P systems with rules and weights on synapses are used to design specific systems to perform three types of arithmetic operations, addition, multiplication and the greatest common divisor.

## 3.1 Addition

In this subsection, we describe an SN P system with rules and weights on synapses that performs the addition of $n$ natural numbers. Such a system is called the SN P system with rules and weights on synapses for $n - addition$. The system has only one input neuron and can be used to perform the sum of arbitrary $n$ natural numbers with $k$ binary bits, where $n \geq 2$, $k \geq 1$. Compared with the standard SN P system, this type of SN P systems can reduce the number of neurons.

As shown in Fig. 1, we construct an SN P system $\prod_{Add}(n, k)$ with rules and weights on synapses to solve the $n - addition$ problem. Its formal definition is omitted.

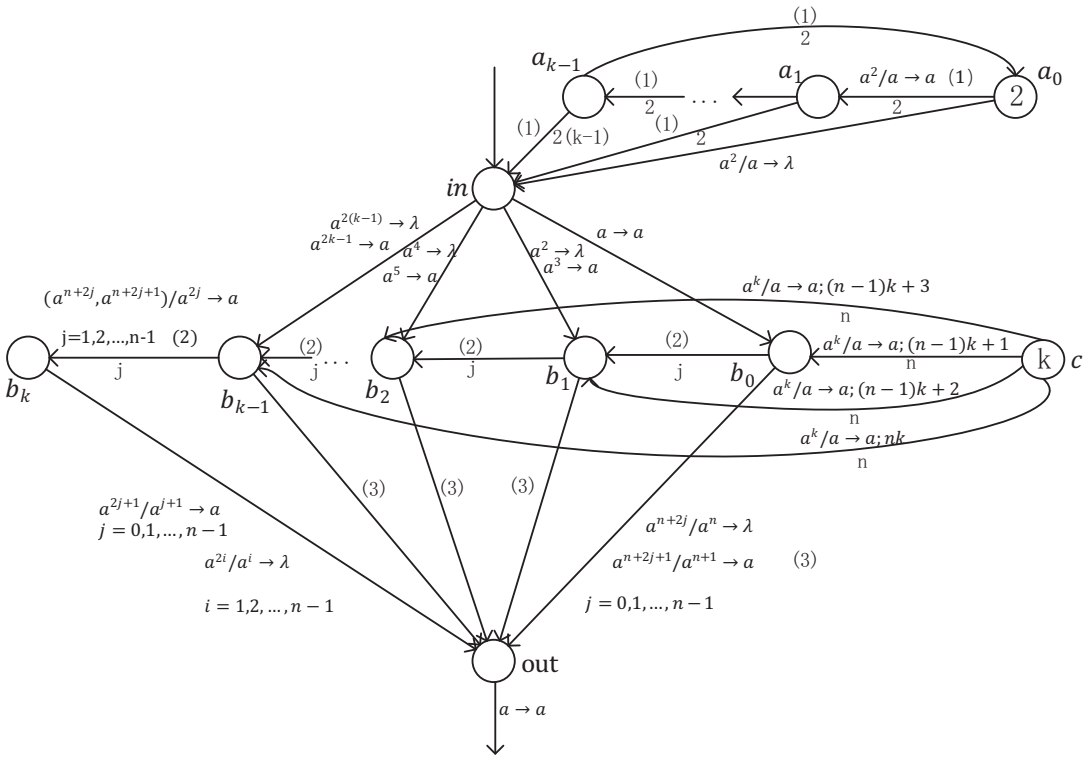


Figure 1: The SN P system $\prod_{Add}(n, k)$ with rules and weights on synapses for addition

The SN P system with rules and weights on synapses for $n - addition$ outputs the sum in binary form of $n$ natural numbers with $k$ binary bits, provided that the neuron $\sigma_{in}$ is in binary form.

Let $t$ denote the current time step. In the initial configuration ($t = 0$), only the neuron $\sigma_{a_0}$ in the system contains 2 spikes and the neuron $\sigma_c$ contains $k$ spikes. The computing procedure of the SN P system with rules and weights on synapses in Fig. 1 is composed of the following steps:

**1. Preprocessing of inputs.** At time $t = 0$, with two spikes in the auxiliary neuron $\sigma_{a_0}$, the rules on synapses $(a_0, in)$ and $(a_0, a_1)$ can be applied. One of the two spikes is consumed by the firing rule $a^2/a \to a$ on synapse $(a_0, a_1)$, and one new spike is produced. Then this new spike is multiplied by the weight 2 on synapse $(a_0, a_1)$ during the transfer. That is, at the next time, two spikes will arrive at neuron $\sigma_{a_1}$, so that the rules on the two synapses emitted by neuron $\sigma_{a_1}$ will be enabled and applied. The other spike is consumed by the forgetting rule $a^2/a \to \lambda$ on synapse $(a_0, in)$, so the number of spikes that neuron $\sigma_{in}$ receives from the auxiliary neuron $\sigma_{a_0}$ is 0 at $t = 1$.

Similarly, the above process is repeated and then we can get that when time $t = 2$, neuron $\sigma_{in}$ receives two spikes from the auxiliary neuron $\sigma_{a_1}$(the weight on synapse $(a_1, in)$ is 2); until time $t = k$, neuron $\sigma_{in}$ receives $2(k-1)$ spikes from neuron $\sigma_{a_{k-1}}$ (the weight on synapse $(a_{k-1}, in)$ is $2(k-1)$). Then the auxiliary neurons send spikes to $\sigma_{in}$ again, from $\sigma_{a_0}$ to $\sigma_{a_{k-1}}$, until the computation ends.

**2. Input and store the first $n$-1 numbers.** At time $t = 1$, the digit which is associated with the power $2^0$ in the binary representation of the first natural number is provided to the input neuron $\sigma_{in}$, while the auxiliary neuron $\sigma_{a_0}$ sends 0 spikes to $\sigma_{in}$. At this time, the number of spikes that neuron $\sigma_{in}$ contains may be 0 or 1. If there is no spike and no rule is activated, then 0 spike will be sent to neuron $\sigma_{b_0}$ at the next time. If there is one spike, then the firing rule $a \to a$ on synapse $(in, b_0)$ is triggered. This means that neuron $\sigma_{b_0}$ can receive one spike at the next time. In this way, we can store the lowest bit of the first natural number into the neuron $\sigma_{b_0}$.

Similarly, we can store the digits which are associated with the power $2^1$, $2^2$, ..., $2^{k-1}$ of the first natural number into $\sigma_{b_1}$, $\sigma_{b_2}$ , ..., $\sigma_{b_{k-1}}$, respectively. At the next time, the second natural number will begin to be input to the system, and the input process is same as that of the first number.

**3. Input the last natural number and calculate the sum of each bit.** At $t = (n-1)k + 2$, the spikes of the lowest bit of the last natural number arrive in $\sigma_{b_0}$, and at the same time, $n$ spikes transmitted by synapse $(c, b_0)$ also arrive at $\sigma_{b_0}$ after $(n-1)k + 1$ steps of delay. The rules on synapses $(b_0, out)$ and $(b_0, b_1)$ can be applied when the number of spikes in $\sigma_{b_0}$ is at least $n$ and $n + 2$, so the system starts to calculate the sum and binary carry at this step, which corresponds to the power $2^0$ of the $n$ natural numbers. Then the sum is sent to the output neuron $\sigma_{out}$, and the carry is sent to the next neuron $\sigma_{b_1}$.

By the same token, the binary digits corresponding to the power $2^i$ of the last natural number and the $n$ spikes transmitted by synapse $(c, b_i)$ arrive at $\sigma_{b_i}$ at the same time. Differing from $\sigma_{b_0}$, the carry from $\sigma_{b_{i-1}}$ also arrives at $\sigma_{b_i}$ at this time, where $i = 1, 2, \ldots, k-1$, and the neuron $\sigma_{b_k}$ only receives the carry from $\sigma_{b_{k-1}}$.

In neuron $\sigma_{b_i}$, we assume that the number of spikes receiving from the input neuron $\sigma_{in}$ is $p$, and the number of spikes that receives from the previous neuron $\sigma_{b_{i-1}}$ is $q$, then the neuron $\sigma_{b_i}$ contains $n + p + q$ spikes. The rules on the synapse $(b_i, out)$ can be divided into two cases, where $i = 0, 1, \ldots, k-1$.

- If the value of $p + q$ is odd, that is $p + q = 2j + 1$, where $j = 0, 1, \ldots, n-1$, the rule $a^{n+2j+1}/a^{n+1} \to a$ on synapse $(b_i, out)$ is applied and consumes $n+1$ spikes in neuron $\sigma_{b_i}$, then one spike is sent to the output neuron $\sigma_{out}$. The output neuron will send this one spike to the environment at next time, that is, the corresponding binary bit of sum is 1.

- If the value of $p + q$ is even, that is $p + q = 2j$, where $j = 0, 1, \ldots, n-1$, then $n$ spikes in neuron $\sigma_{b_i}$ are forgotten by the rule $a^{n+2j}/a^n \to \lambda$ on synapse $(b_i, out)$, and no new spikes are produced. That is, the corresponding binary bit of sum is 0.

At the same time of producing each bit of the sum, it is possible to generate carry. When $j \neq 0$ (without 0 or 1 spike), the rule $(a^{n+2j}, a^{n+2j+1})/a^{2j} \to a$ on synapse $(b_i, b_{i+1})$ is applied and produces one spike. Then $j$ spikes are sent to the next neuron (the weight on synapse $(b_i, b_{i+1})$ is $j$), where $j = 1, 2, \ldots, n-1$. It should be noted that the neuron $\sigma_{b_k}$ only receives the carry spikes from the previous neuron $\sigma_{b_{k-1}}$.

**4. Output the result.** Each of the binary bits of the sum can be sent to the output neuron $\sigma_{out}$ at different steps, and $\sigma_{out}$ uses the rule $a \to a$ to output the result to the environment based on the number 0 or 1 of spikes in $\sigma_{out}$.

Based on the above description, the system shown in Fig. 1 can calculate the sum of the $n$ natural numbers with $k$ binary bits, where $n \geq 2$, $k \geq 1$, and send the result to the environment.

As an example, let us consider the addition $110_2 + 011_2 = 1001_2$, that is, $n = 2$, $k = 3$. Fig. 2 depicts the SN P system $\prod_{Add}(2, 3)$ with rules and weights on synapses. Table 1 reports the spikes contained in each neuron of $\prod_{Add}(2, 3)$, as well as the number of spikes sent to the environment, at each step during the computation. The input and the output sequences are written in bold. Note that the first instance of time for which the output is valid is $t = 7$.

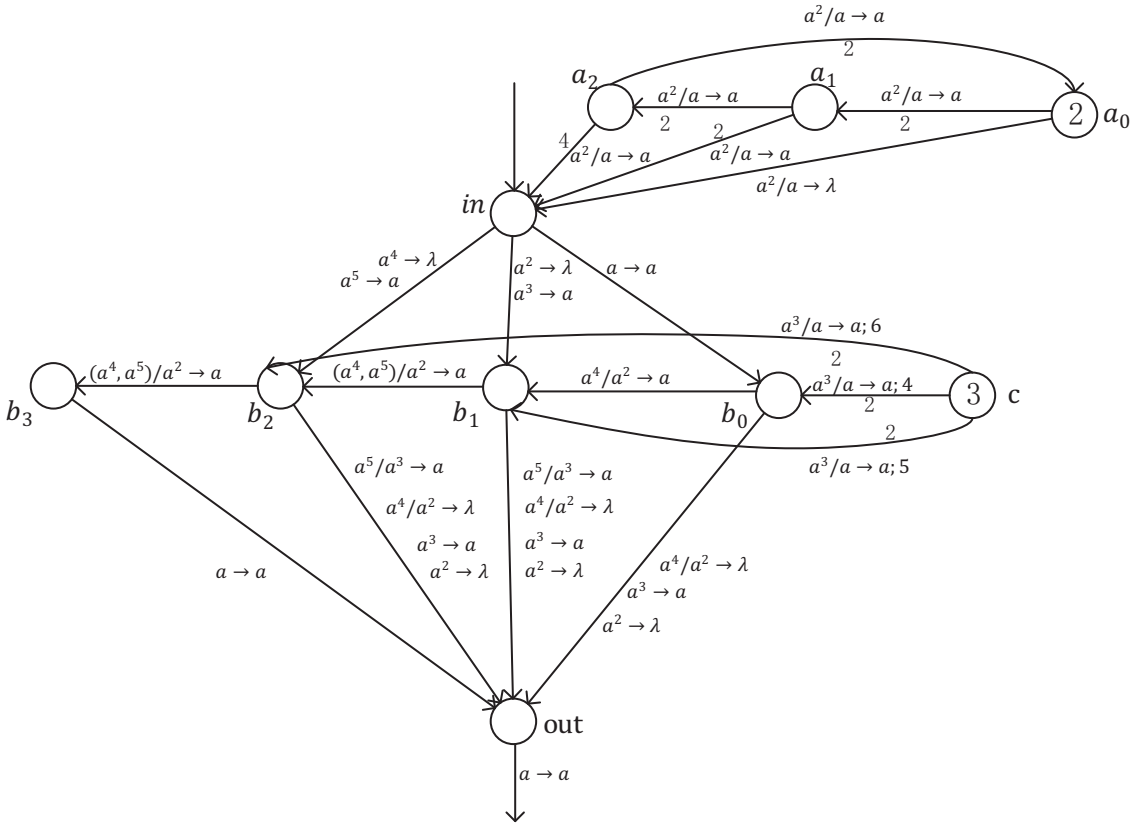


Figure 2: The SN P system $\prod_{Add}(2, 3)$ with rules and weights on synapses for addition

Compared with the example in [32], when $k = 3$, the number of neurons required to solve this problem by using the standard SN P system is 14; as shown in Fig. 2, the number of neurons required in SN P system with rules and weights on synapses is 10, which effectively reduces the number of neurons.

Table 1: The configurations and outputs of $\prod_{Add}(2,3)$ at each time step during the computation of the addition $110_2 + 011_2 = 1001_2$

| $t$ | $in$ | $a_0$ | $a_1$ | $a_2$ | $c$ | $b_0$ | $b_1$ | $b_2$ | $b_3$ | $out$ | $output$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | **0** | 0 | 2 | 0 | $O(4)O(5)O(6)$ | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | **1**+2 | 0 | 0 | 2 | $O(3)O(4)O(5)$ | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | **1**+4 | 2 | 0 | 0 | $O(2)O(3)O(4)$ | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | **1** | 0 | 2 | 0 | $O(1)O(2)O(3)$ | 0 | 1 | 1 | 0 | 0 | 0 |
| 5 | **1**+2 | 0 | 0 | 2 | $O(1)O(2)$ | 3 | 1 | 1 | 0 | 0 | 0 |
| 6 | **0**+4 | 2 | 0 | 0 | $O(1)$ | 0 | 4 | 1 | 0 | 1 | 0 |
| 7 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | **1** |
| 8 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | **0** |
| 9 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | **0** |
| 10 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |

### 3.2 Multiplication

Multiplication of two arbitrary natural numbers is also called the general binary multiplication, which has been performed on an SN P system in [32]. In order to reduce the number of neurons in the system, for two binary numbers $m$ and $n$ with $k$ binary bits, we construct the SN P system $\prod_{Multiple}(k)$ with rules and weights on synapses as shown in Fig. 3 to perform the general binary multiplication, where $k \geq 1$, $m \geq 0$, $n \geq 0$.

The SN P system with rules and weights on synapses for the general binary multiplication of Fig. 3 outputs the product $m \times n$ in binary form of two natural numbers $m$ and $n$ with $k$ binary bits, provided that the neuron $\sigma_{in}$ is in binary form. Assuming that $m$ and $n$ are two natural numbers with $k$ binary bits, and $m$ and $n$ are rewritten as follows:

$$m = \sum_{i=0}^{k-1} m_i 2^i, n = \sum_{j=0}^{k-1} n_j 2^j, \text{then:}$$

$$m \times n = n_0 m_0 2^0 + (n_0 m_1 + n_1 m_0)2^1 + \cdots + (n_1 m_{k-1} + n_2 m_{k-2} + \cdots + n_{k-1} m_1)2^k + \cdots + n_{k-1} m_{k-1} 2^{2k-2}$$

From the above expression we can see that the product of $m$ and $n$ can be decomposed into the sum of $2k-1$ terms.

Let $t$ denote the current time step. In the initial configuration ($t = 0$), the neuron $\sigma_{a_0}$ contains 2 spikes and $\sigma_{g_i}$ contains $k$ spikes, where $i = 2, 3, \ldots, k$. Fig. 3 shows the SN P system with rules and weights on synapses for the general binary multiplication, and the operation process consists of the following steps:

**1. Preprocessing of input.** It is similar to the input preprocessing part of the system $\prod_{Add}(n,k)$, so we only describe the differences.

The auxiliary neuron $\sigma_{a_i}$ only acts on the natural number $m$, which is the first input to the system, where $i = 0, 1, \ldots, k-1$. When each of the corresponding binary bits of $m$ is input to

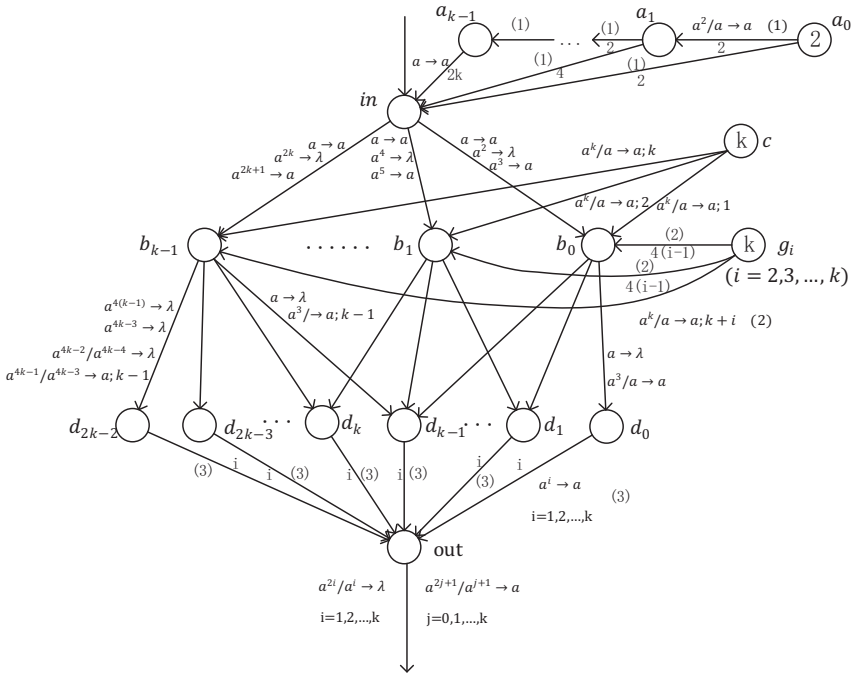


Figure 3: The SN P system $\prod_{Multiple}(k)$ with rules and weights on synapses for multiplication

$\sigma_{in}$, the auxiliary neurons $\sigma_{a_0}$, $\sigma_{a_1}$, ..., $\sigma_{a_{k-1}}$ send 2, 4 ,..., $2k$ spikes to $\sigma_{in}$, respectively.

**2. Input and store the binary bits of the first natural number** $m$**.** At $t = 1$, the digit which is associated with the power $2^0$ in the binary representation of $m$ has been provided to $\sigma_{in}$, while the auxiliary neuron $\sigma_{a_0}$ sends 2 spikes to $\sigma_{in}$. So the number of spikes that neuron $\sigma_{in}$ contains may be 2 or 3. If there are 2 spikes, then the rule $a^2 \to \lambda$ on synapse $(in, b_0)$ can be applied, so that the 2 spikes are consumed and no spike is sent out. At $t = 2$, the neuron $\sigma_{b_0}$ receives only one spike from synapse $(c, b_0)$ (after one step delay), and this spike will be forgotten by rule $a \to \lambda$ on synapse $(b_0, d_0)$ at the next step. If there are 3 spikes, where two of them come from $\sigma_{a_0}$ and the other one from the environment, then the rule $a^3 \to a$ on synapse $(in, b_0)$ is triggered; as a consequence, one spike is sent out. Then at $t = 2$, there are two spikes placed in neuron $\sigma_{b_0}$, where one of them comes from synapse $(in, b_0)$ and the other one from synapse $(c, b_0)$. These two spikes remain in $\sigma_{b_0}$ for the next steps.

Thus, $k$ binary bits of $m$ can be stored in the neurons $\sigma_{b_0}$, $\sigma_{b_1}$, ..., $\sigma_{b_{k-1}}$ by repeating the above operations. If there are 2 spikes, the $i-th$ binary bit of $m$ is 1; if there is no spike, the $i-th$ binary bit of $m$ is 0, where $i = 0, 1, \ldots, k-1$.

**3. Each binary number of the natural number** $m$ **multiplies all binary numbers of** $n$**, which can produce all possible products of binary numbers of** $m$ **and** $n$**.** At $t = k+1$, the digit which is associated with the power $2^0$ in the binary representation of $n$ has been provided to the input neuron $\sigma_{in}$. If the digit is 1, then each rule $a \to a$ on synapses $(in, b_0), (in, b_1), \ldots, (in, b_{k-1})$ is triggered at the same time, and each of neurons $\sigma_{b_0}$, $\sigma_{b_1}$, ..., $\sigma_{b_{k-1}}$ will receive one spike at next step. Otherwise, no spike will be sent to neurons $\sigma_{b_0}$, $\sigma_{b_1}$, ..., $\sigma_{b_{k-1}}$.

Now let us focus on the neuron $\sigma_{b_i}$, where $i = 0, 1, \ldots, k-1$. At $t = k+2$, the number of spikes in $\sigma_{b_i}$ may be 0,1,2,3, and we can thus consider the following four cases.

- If $\sigma_{b_i}$ contains 0 spikes, then no rule can be applied, thus no spike is sent out. This encodes the operation $n_0 m_i = 0 \times 0 = 0$.

- If $\sigma_{b_i}$ contains 1 spike, then it comes from $\sigma_{in}$. The rule $a \to \lambda$ on synapse $(b_i, d_i)$ is triggered, so that one spike is consumed and no spike is sent out. This encodes the operation $n_0 m_i = 1 \times 0 = 0$.

- If $\sigma_{b_i}$ contains 2 spikes, then they are all remained from the previous step. No rule can be applied. This encodes the operation $n_0 m_i = 0 \times 1 = 0$.

- If $\sigma_{b_i}$ contains 3 spikes, then two of them are remained from the previous step and the other one comes from $\sigma_{in}$ at this step. The rule $a^3/a \to a; i$ on synapse $(b_i, d_i)$ is triggered; as a consequence, one spike reaches the neuron $\sigma_{d_i}$ after $i$ steps of delay and one spike is consumed, thus two spikes in $\sigma_{b_i}$ are left for the next step. This encodes the operation $n_0 m_i = 1 \times 1 = 1$.

When $n_1$ reaches neuron $\sigma_{b_i}$, $\sigma_{b_i}$ also receives four spikes from neuron $\sigma_{g_2}$ (after $k+2$ steps delay). Therefore, the number of spikes contained in neuron $\sigma_{b_i}$ at this time may be 4, 5, 6, and 7, respectively, corresponding to the four cases 0, 1, 2, and 3 mentioned above, and the rules on synapses $(b_i, d_{i+1})$ can be applied and send the result $n_1 m_i$ to neuron $\sigma_{d_{i+1}}$, where $i = 0, 1, \ldots, k-1$.

Similarly, we can get $n_2 m_i, \ldots, n_{k-1} m_i$ in the same way, and $n_j m_i$ is sent to neuron $\sigma_{d_{i+j}}$, where $0 \le i \le k-1$, $0 \le j \le k-1$.

**4. The sum of terms.** At $t = k+3$, $n_0 m_0$ arrives at neuron $\sigma_{d_0}$ through synapse $(b_0, d_0)$. ... At $t = 3k+1$, $n_{k-1} m_{k-1}$ arrive at $\sigma_{d_{2k-2}}$. The number of spikes contained in neuron $\sigma_{d_i}$ above corresponds to the binary bit of $m \times n$, where $i = 0, 1, \ldots, 2k-2$.

**5. Output the result.** The spikes in neurons $\sigma_{d_0}$, $\sigma_{d_1}$, ..., $\sigma_{d_{2k-2}}$ above are sent to the output neuron $\sigma_{out}$. According to the number of spikes in the output neuron $\sigma_{out}$, we consider the following two cases.

- If the number of spikes is odd, the rule $a^{2j+1}/a^{j+1} \to a$ on synapse $(out, e)$ is applied, so that $j+1$ spikes are consumed and one spike is sent to the environment, and $j$ spikes remain in $\sigma_{out}$ for the next step.

- If the number of spikes is even, the rule $a^{2j}/a^j \to \lambda$ on synapse $(out, e)$ is applied, so that $j$ spikes are consumed and no spike is sent to the environment, and $j$ spikes remain in $\sigma_{out}$ for the next step.

From time $t = k+5$, the system begins to have output. Therefore, the system shown in Fig. 3 can perform the general binary multiplication, and output the result to the environment.

As an example, let us consider the multiplication $11_2 \times 11_2 = 1001_2$, that is $k = 2$. Fig. 4 depicts the SN P system $\prod_{Multiple}(2)$ with rules and weights on synapses. Table 2 reports the spikes contained in each neuron of $\prod_{Multiple}(2)$, as well as the number of spikes sent to the environment, at each step during the computation. Note that the time for the first instance that the output is valid is $t = 7$.

When $k = 2$, the number of neurons required in the multiplication SN P system is 17 in [32]; as shown in Fig. 4, the number of neurons required in SN P system with rules and weights on synapses is 11, which effectively reduces 6 neurons.

### 3.3 The greatest common divisor

In this subsection, the greatest common divisor of two natural numbers is performed on SN P systems with rules and weights on synapses. We divide the solution for the greatest common divisor into several individual modules, and then call the different modules according to the

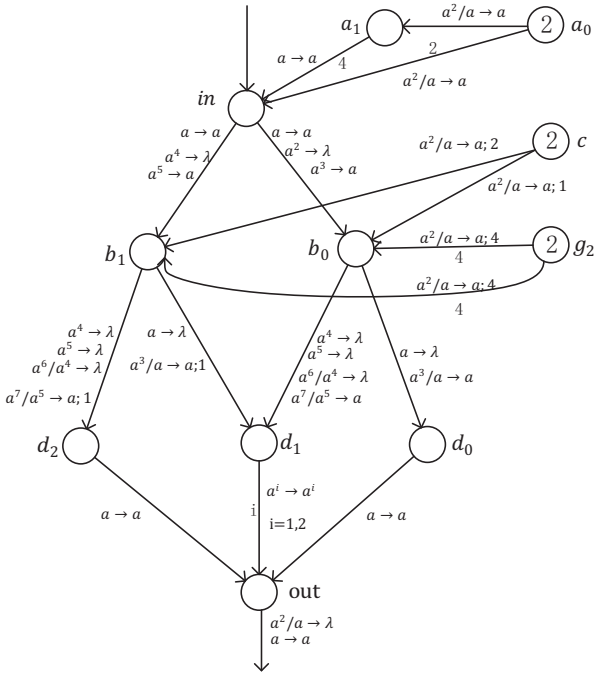


Figure 4: The SN P system $\prod_{Multiple}(2)$ with rules and weights on synapses for multiplication

method steps. The method for solving the greatest common divisor that we use is the binary method. Please refer to [34] to learn more details on the binary method.

Each of the modules required for the above method is given below. These modules are achieved by SN P systems with rules and weights on synapses.

**Module 1:** *To determine whether the natural number is even*

If a number is even, then the digit which is associated with the power $2^0$ in the binary representation is 0; otherwise, this digit is 1. So we can only use the digit which is associated with the power $2^0$ in the binary representation of the number to judge whether it is even or odd.

For a natural number, we construct the SN P system $\prod_{Even}$ with rules and weights on synapses as shown in Fig. 5 to determine whether it is even or not. If the input is even, the number of spikes that output to the environment is 0, otherwise, the output of this system is not 0. The time that the output is valid for the first instance is $t = 3$.

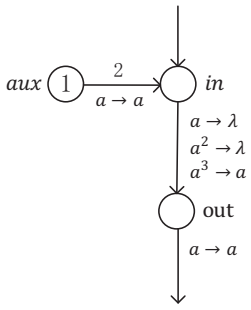


Figure 5: The SN P system $\prod_{Even}$ with rules and weights on synapses

**Module 2:** *To divide the number by 2 (to shift one bit to the right)*

The binary even number is divided by 2, which means that the last bit 0 of this binary number is removed. So if the length of binary bits of the input is $k$, then the length of binary

Table 2: The configurations and outputs of $\prod_{Multiple}(2)$ at each time step during the computation of the multiplication $11_2 \times 11_2 = 1001_2$

| $t$ | $in$ | $a_0$ | $a_1$ | $c$ | $g_2$ | $b_0$ | $b_1$ | $d_0$ | $d_1$ | $d_2$ | $out$ | $output$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | **1**+2 | 0 | 1 | $O(1)O(2)$ | $O(4)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | **1**+4 | 0 | 0 | $O(1)$ | $O(3)$ | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | **1** | 0 | 0 | 0 | $O(2)$ | 2 | 2 | 0 | 0 | 0 | 0 | 0 |
| 4 | **1** | 0 | 0 | 0 | $O(1)$ | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 7 | 7 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 2 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 1 | 2 | **1** |
| 8 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 2 | **0** |
| 9 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 1 | **0** |
| 10 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | **1** |

bits of the output is $k-1$.

For an even number, we construct the SN P system $\prod_{Divide2}$ with rules and weights on synapses as shown in Fig. 6 to divide it by 2. The time that the output is valid for the first instance is $t = 4$.

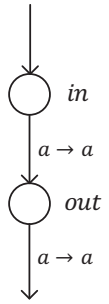


Figure 6: The SN P system $\prod_{Divide2}$ with rules and weights on synapses

**Module 3:** *To add 1 to the number*

For a natural number, we construct the SN P system $\prod_{Add1}$ with rules and weights on synapses as shown in Fig. 7 to add 1 to it. The first time step that the output starts to be emitted by the system is $t = 3$.

**Module 4:** *To compare the value of two natural numbers*

We construct the SN P system $\prod_{Comp}$ with rules and weights on synapses as shown in Fig. 8 to compare two natural numbers. The inputs of this system are two appropriate sequences of spikes of two natural numbers in binary form, but what is different from the other modules is that the input here starts from the highest bit to the lower.

For two natural numbers in binary form with the same length of binary bits, we compare

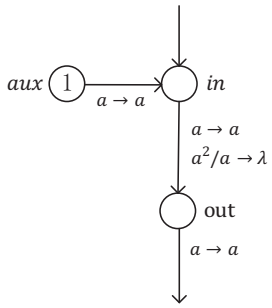


Figure 7: The SN P system $\prod_{Add1}$ with rules and weights on synapses

them from the highest bit of each number to the lower order, until the system outputs 2 or 1. The result is the last bit of the output. If the result is 0, then $in_1 = in_2$; if the result is 1, then $in_1 < in_2$; if the result is 2, then $in_1 > in_2$.

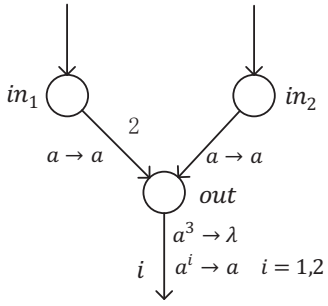


Figure 8: The SN P system $\prod_{Comp}$ with rules and weights on synapses

**Module 5:** *Subtraction*

The design principle of the subtraction module is referred to [8]. For two natural numbers, we construct the SN P system $\prod_{Sub}$ with rules and weights on synapses as shown in Fig. 9 to perform subtraction. The time that the output is valid for the first instance is $t = 4$.

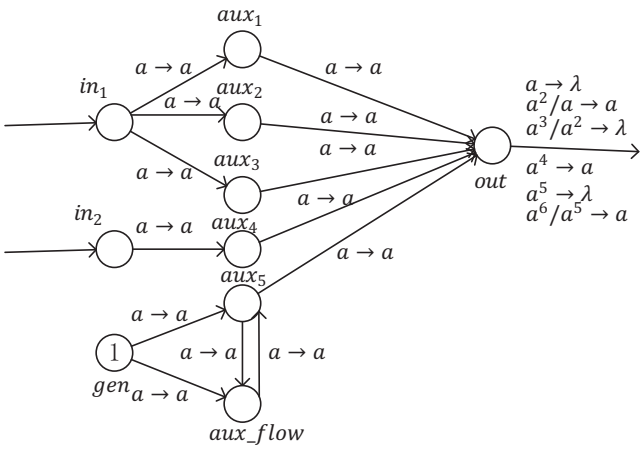


Figure 9: The SN P system $\prod_{Sub}$ with rules and weights on synapses

**Module 6:** *To calculate the result (to multiply by 2 with given times)*

By calling the above five modules, the numbers $d$ and $g$ are obtained. This module uses $d$ and $g$ to calculate the greatest common divisor $g \times 2^d$. That is, move $g$ to the left by $d$ bits,

Table 3: The configurations and outputs of $\prod_{Even}$ at each time step during the process of judging whether $a = 01100_2$ and $b = 10010_2$ is even

| $t$ | $in(a)$ | $in(b)$ | $aux$ | $out$ | $output$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | **0**+2 | **0**+2 | 0 | 0 | 0 |
| 2 | **0** | **1** | 0 | 0 | 0 |
| 3 | **1** | **0** | 0 | 0 | **0** |
| 4 | **1** | **0** | 0 | 0 | **0** |
| 5 | **0** | **1** | 0 | 0 | **0** |
| 6 | 0 | 0 | 0 | 0 | **0** |
| 7 | 0 | 0 | 0 | 0 | **0** |

where $d$ represents the number of times that all the two numbers are even.

We construct the SN P system $\prod_{Multiply2}$ with rules and weights on synapses as shown in Fig. 10 to calculate the result. The input is the number $g$ in binary form. The first time step that the output starts to be emitted by the system is $t = 3$.

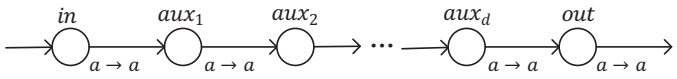


Figure 10: The SN P system $\prod_{Multiply2}$ with rules and weights on synapses

By calling above modules, we can get the greatest common divisor of two natural numbers. In what follows, an example is provided to detail the process.

As an example, let us consider the greatest common divisor of the numbers $a = 01100_2$ and $b = 10010_2$. The calling process of modules to calculate $gcd(01100_2, 10010_2)$ is as follows. In the initial time, we assume that $g = 0$ and $d = 0$.

**Step 1.** *Call the module 1 to determine whether a and b are even*

When the input of system $\prod_{Even}$ is $a = 01100_2$ or $b = 10010_2$, the spikes contained in each neuron of $\prod_{Even}$, as well as the number of spikes sent to the environment at each step during the process that judge whether $a$ or $b$ is even, are reported in Table 3.

From Table 3, we can know that the output of $\prod_{Even}$ is 0, so $a = 01100_2$ is an even number, and b=$10010_2$ is an even number, too.

**Step 2.** *Call the module 2 to divide a and b by 2*

When the input of system $\prod_{Divide2}$ is $a = 01100_2$, the output is 0110. When the input of system $\prod_{Divide2}$ is $b = 10010_2$, the output is 1001. So, after dividing $a$ and $b$ by 2, we can get $a = 0110_2$ and $b = 1001_2$.

**Step 3.** *Call the module 3 to add 1 to d*

In the initial time, $d = 0$, so the input of system $\prod_{Add1}$ is 0. After adding 1 to $d$, we can get $d = 1$.

**Step 4.** *Call the module 1 to determine whether a and b are even*

Table 4: The configurations and outputs of $\prod_{Comp}$ at each time step during the process of comparing the value of $a = 0011_2$ and $b = 1001_2$

| $t$ | $in_1$ | $in_2$ | $out$ | $output$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | **0** | **1** | 0 | 0 |
| 2 | **0** | **0** | 1 | 0 |
| 3 | **1** | **0** | 0 | 0 |
| 4 | **1** | **1** | 2 | **1** |

When the input of system $\prod_{Even}$ is $a = 0110_2$, the output is 0, so $a = 0110_2$ is an even number. When the input of system $\prod_{Even}$ is $b = 1001_2$, the output is 1, so $b = 1001_2$ is odd.

**Step 5.** *Call the module 2 to divide a by 2*

The input of system $\prod_{Divide2}$ is $a = 0110_2$. After dividing $a$ by 2, we get $a = 011_2$.

**Step 6.** *Call the module 1 to determine whether a is even*

The input of system $\prod_{Even}$ is $a = 011_2$, and then the output of $\prod_{Even}$ is 1, so $a = 011_2$ is an odd number.

**Step 7.** *Call the module 4 to compare the value of a and b*

In the system $\prod_{Comp}$, $in_1 = a = 0011_2$ and $in_2 = b = 1001_2$. Table 4 reports the spikes contained in each neuron of $\prod_{Comp}$, as well as the number of spikes sent to the environment at each step during the process of comparing the value of $a = 0011_2$ and $b = 1001_2$. The output of the system is 1, so $a < b$.

**Step 8.** *Call the module 5 and module 2 to calculate* $(b - a)/2$

Because $a < b$, we can get $in_1 = b = 1001_2$ and $in_2 = a = 0011_2$ in the system $\prod_{Sub}$. After computing in $\prod_{Sub}$, we get $b - a = 1001_2 - 0011_2 = 0110_2$.

Then the input of system $\prod_{Divide2}$ is $0110_2$. After dividing it by 2, we can get $b = (b-a)/2 = 0110_2 \div 10_2 = 011_2$.

**Step 9.** *Call the module 4 to compare the value of a and b*

In the system $\prod_{Comp}$, $in_1 = a = 011_2$ and $in_2 = b = 011_2$. The output of the system is 0, so $a = b$.

**Step 10.** *Call the module 6 to calculate the result*

From the above steps, we can get $g = a = b = 011_2$ and $d = 1$. So the input of system $\prod_{Multiply2}$ is $g = 011_2$, and the spikes contained in each neuron of $\prod_{Multiply2}$, as well as the number of spikes sent to the environment at each step during computation of the multiplication $011_2 \times 2^1 = 110_2$ are reported in Table 5.

So the greatest common divisor of $01100_2$ and $10010_2$ is $g \times 2^d = 011_2 \times 2^1 = 110_2$.

Through the above analysis we can see that the greatest common divisor of two arbitrary natural numbers is available by calling the corresponding modules of method.

### 3.4 Discussion

For addition and multiplication, both the systems constructed in [32] and the ones in this paper have only one input neuron, and the integers encoded in binary form should be input in order. The system in [32] to perform the addition of $n$ natural numbers with 3 binary bits

Table 5: The configurations and outputs of $\prod_{Multiply2}$ at each time step during the computation of the multiplication $011_2 \times 2^1 = 110_2$

| $t$ | $in$ | $aux_1$ | $out$ | $output$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | **1** | 0 | 0 | 0 |
| 2 | **1** | 1 | 0 | 0 |
| 3 | **0** | 1 | 1 | **0** |
| 4 | 0 | 0 | 1 | **1** |
| 5 | 0 | 0 | 0 | **1** |
| 6 | 0 | 0 | 0 | **0** |

requires 14 neurons, while 10 neurons are enough by using SN P systems with rules and weights on synapses. When the multiplication of two natural numbers with 2 binary bits is performed, the system designed in [32] requires 17 neurons, while in this paper, we use only 11 neurons. Thus, we can conclude that the systems with a single input neuron for addition and multiplication in [32] are improved by using the SN P systems with rules and weights on synapses. It is worth pointing out that the designed SN P systems in this paper use larger number of synapses and delays than the systems in [32]. For addition, due to the constraints of input conditions, the systems need to calculate the sum of each bit until the last number is input, which makes the addition system relatively complicated. In addition, this study is the first attempt to design an SN P system to calculate the greatest common divisor of two natural numbers.

## 4 Conclusions and future work

The use of the SN P systems with rules and weights on synapses to design specific systems for fulfilling the arithmetic operations is focused in this paper, including addition, multiplication and the greatest common divisor. Comparing with the results reported in [32], smaller number of neurons are required to perform the addition and multiplication. Calculating the greatest common divisor of two natural numbers is not an easy task. This paper achieved a good design by using several modules. Following this work, we will consider how to use the SN P system with rules and weights on synapses to perform the division operation.

## Acknowledgments

This work was supported by National Natural Science Foundation of China (61179032, 61672437, 61702428), the Special Scientific Research Fund of Food Public Welfare Profession of China (201513004-3), the Guiding Scientific Research Project of Hubei Provincial Education Department (2017078), the Humanities and Social Sciences Fund Project of Hubei Provincial Education Department (17Y071) and Sichuan Science and Technology Program (2018GZ0185, 2018GZ0085, 2017GZ0159).

## Bibliography

[1] Atanasiu, A.; Martinvide, C. (2001); Arithmetic with membranes, *Romanian Journal of Information Science and Technology*, 4(12), 5–20, 2001.

[2] Ciobanu, G.; Păun, Gh.; Pérez-Jiménez, M.J. (2006); *Applications of membrane computing*, Springer, 2006.

[3] Dzitac, I. (2015); Impact of membrane computing and P systems in ISI WoS. Celebrating the 65th Birthday of Gheorghe Păun, *International Journal of Computers Communications & Control*, 10(5), 617–626, 2015.

[4] Guo, P.; Chen, H.; Zhang, H. (2015); An integrated P system for arithmetic operations, *Journal of Computational & Theoretical Nanoscience*, 12(10), 3346–3356, 2015.

[5] Ionescu, M.; Păun, Gh.; Yokomori, T. (2006); Spiking neural P systems, *Fundamenta Informaticae*, 71, 279–308, 2006.

[6] Ionescu, M.; Sburlan, D. (2012); Some applications of spiking neural P systems, *Computing and Informatics*, 27(3), 515–528, 2012.

[7] Kong, Y. (2005); *Study on the computational power of spiking neural P system based on different biological background*, Huazhong University of Science and Technology, 2005.

[8] Naranjo, M.A.G.; Leporati, A.; Pan, L. (2009); Performing arithmetic operations with spiking neural P systems, *Proc. of the Seventh Brainstorming Week on Membrane Computing*, 11(4), 181–198, 2009.

[9] Pan, L.; Păun, Gh. (2009); Spiking neural P systems with anti-spikes, *International Journal of Computers Communications & Control*, 4(3), 273–282, 2009.

[10] Pan, L.; Păun, Gh.; Zhang, G.; Neri, F. (2017); Spiking neural P systems with communication on request, *International Journal of Neural Systems*, 27(8), 2017.

[11] Pan, L.; Zeng, X.; Zhang, X.; Jiang, X. (2012); Spiking neural P systems with weighted synapses, *Neural Processing Letters*, 35(1), 13–27, 2012.

[12] Păun, Gh. (2000); Computing with membranes, *Journal of Computer and System Sciences*, 61(1), 108–143, 2000.

[13] Păun, Gh. (2002); Membrane computing: an introduction, *Theoretical Computer Science*, 287(1), 73–100, 2002.

[14] Păun, Gh. (2016); Membrane computing and economics: A general view, *International Journal of Computers Communications & Control*, 11(1), 105–112, 2016.

[15] Păun, Gh.; Rozenberg, G.; Salomaa, A. (2010); *The oxford handbook of membrane computing*, Oxford University Press, 2010.

[16] Peng, H.; Wang, J.; Pérez-Jiménez, M.J.; Wang, H.; Shao, J.; Wang, T. (2013); Fuzzy reasoning spiking neural P system for fault diagnosis, *Information Sciences*, 235(6), 106–116, 2013.

[17] Song, T.; Gong, F.; Liu, X.; Zhao, Y.; Zhang, X. (2016); Spiking neural P systems with white hole neurons, *IEEE Transactions on Nanobioscience*, 15(7), 666–673, 2016.

[18] Song, T.; Pan, L. (2015); Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy, *IEEE Transactions on NanoBioscience*, 14(1), 38–44, 2015.

[19] Song, T.; Pan, L.; Păun, Gh. (2014); Spiking neural P systems with rules on synapses, *Theoretical Computer Science*, 529, 82–95, 2014.

[20] Song, T.; Rodríguez-Patón, A.; Zheng, P.; Zeng, X. (2017); Spiking neural P systems with colored spikes, *IEEE Transactions on Cognitive & Developmental Systems*, doi: 10.1109/TCDS.2017.2785332, 2017.

[21] Song, T.; Zheng, P.; Wong, M.L.D.; Wang, X. (2016); Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control, *Information Sciences*, 372, 380–391, 2016.

[22] Wang, J.; Hoogeboom, H.J.; Pan, L.; Păun, Gh.; Pérez-Jiménez, M.J. (2010); Spiking neural P systems with weights, *Neural Computation*, 22(10), 2615–2646, 2010.

[23] Wang, J.; Shi, P.; Peng, H.; Pérez-Jiménez, M.J.; Wang, T. (2013); Weighted fuzzy spiking neural P systems, *IEEE Transactions on Fuzzy Systems*, 21(2), 209–220, 2013.

[24] Wang, T.; Zhang, G.; Zhao, J.; He, Z.; Wang, J.; Pérez-Jiménez, M.J. (2015); Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems, *IEEE Transactions on Power Systems*, 30(3), 1182–1194, 2015.

[25] Zeng, X.; Song, T.; Zhang, X.; Pan, L. (2012); Performing four basic arithmetic operations with spiking neural P systems, *IEEE Transactions on Nanobioscience*, 11(4), 366–374, 2012.

[26] Zeng, X.; Xu, L.; Liu, X.; Pan, L. (2014); On languages generated by spiking neural P systems with weights, *Information Sciences*, 278(10), 423–433, 2014.

[27] Zeng, X.; Zhang, X.; Song, T.; Pan, L. (2014); Spiking neural P systems with thresholds, *Neural Computation*, 26(7), 1340–1361, 2014.

[28] Zhang, G.; Cheng, J.; Gheorghe, M.; Meng, Q. (2013); A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, *Applied Soft Computing*, 13(3), 1528–1542, 2013.

[29] Zhang, G.; Pérez-Jiménez, M.J.; Gheorghe, M. (2017); *Real-life applications with membrane computing*, Springer International Publishing, 2017.

[30] Zhang, G.; Rong, H.; Neri, F.; Pérez-Jiménez, M.J. (2014); An optimization spiking neural P system for approximately solving combinatorial optimization problems, *International Journal of Neural Systems*, 24(5), 450–3642, 2014.

[31] Zhang, G.; Rong, H.; Ou, Z.; Pérez-Jiménez, M.J.; Gheorghe, M. (2014); Automatic design of deterministic and non-halting membrane systems by tuning syntactical ingredients, *IEEE Transactions on NanoBioscience*, 13(3), 363–371, 2014.

[32] Zhang, X.; Zeng, X.; Pan, L.; Luo, B. (2009); A spiking neural P system for performing multiplication of two arbitrary natural numbers, *Chinese Journal of Computers*, 32(12): 2362–2372, 2009.

[33] [Online]. Available: ppage.psystems.eu.

[34] [Online]. Available: en.wikipedia.org/wiki/Greatest_common_divisor.