

Article

Bio-Inspired Constant-Time Arithmetic Kernels in Hybrid Membrane–Neural Spiking P Systems

Eduardo Vázquez ^{1,*}, Josue J. Guillen ¹, Daniel-Eduardo Vázquez ^{2,*}, Giovanny Sanchez ¹,
Juan-Gerardo Avalos ¹, Gonzalo Duchén ¹, Gabriel Sánchez ¹ and Linda Karina Toscano ¹

¹ Instituto Politécnico Nacional (IPN), ESIME Culhuacán, Av. Santa Ana No. 1000, Mexico City 04260, Mexico

² Unidad Iztapalapa, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana, Mexico City 09340, Mexico

* Correspondence: edvazquezf@ipn.mx (E.V.); cbi2243012644@izt.uam.mx (D.-E.V.)

Abstract

This work introduces Hybrid Membrane–Neural P systems (HMN P systems), a computational model that integrates principles from membrane computing and spiking neural P systems. The resulting framework offers a versatile foundation for the development of bio-inspired arithmetic architectures. Within this setting, we propose a compact family of arithmetic kernels capable of executing signed addition, subtraction, multiplication, and division in both modular and non-modular arithmetic domains. By leveraging intrinsic spike aggregation, spike–anti-spike annihilation, and exhaustive rule application, the proposed designs achieve efficient and reliable arithmetic computation in a constant number of simulation steps under exhaustive semantics and assuming synchronized input, independent of operand values. Addition and subtraction are executed intrinsically upon spike arrival, requiring no internal computation steps, while multiplication and division are completed in a single simulation step by one neuron. Furthermore, we introduce a modular-reduction kernel that operates in two simulation steps with a single neuron, and leverage its modular structure to construct modular multiplication and division through composition with non-modular arithmetic modules. Comparative evaluations against representative SNP and SNQ arithmetic designs demonstrate that HMN kernels achieve operand-independent execution time while requiring fewer neurons. Distinct from most existing approaches, the HMN framework natively supports signed operands through a dual-spike representation, thereby eliminating the need for auxiliary sign-handling mechanisms. Asynchronous spike arrivals can be managed by an optional synchronization membrane; since this mechanism is decoupled from the arithmetic kernels, its overhead is excluded from kernel performance and reported separately. Collectively, these results establish HMN systems as an efficient and modular platform for constant-time arithmetic computation, offering reusable arithmetic kernels that serve as a foundation for higher-level constructions, including those arising in elliptic-curve and modular arithmetic.



Received: 31 December 2025

Revised: 20 February 2026

Accepted: 21 February 2026

Published: 26 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: membrane computing; spiking neural P systems; arithmetic circuits; exhaustive semantics; modular arithmetic; constant-time arithmetic; bio-inspired computation

MSC: 68Q10; 68Q07

1. Introduction

The growing computational demands of modern applications—driven by large-scale neural models, image processing, cryptographic computation, high-performance arithmetic,

and the scalability requirements of distributed systems—have motivated the development of new massively parallel computational models. *Membrane computing*, introduced by Păun [1], is a hierarchical, bio-inspired model of distributed computation whose Turing-complete operational semantics offer a structured alternative to classical sequential architectures, while also incorporating a flexible hierarchical organization and inherent multiset dynamics that implement aggregation and annihilation in a natural way. Within this framework, *SNP* [2,3] and *SNQ systems* [4] provide neuron-like computational abstractions in which arithmetic behavior is realized through the application of spike-handling rules. Although arithmetic can be expressed in SNP and SNQ systems, their usual dependence on selective-mode rule application restricts parallelism and leads to significant computation times [5–8]. In modular arithmetic, some existing systems compute $(b \bmod p)$ via iterative subtraction [9], leading to execution times that depend on operand size and thereby limit scalability. These limitations prevent constant-time execution and significantly restrict the modular reuse of arithmetic components.

Recent research has further expanded the theoretical and practical scope of spiking neural and membrane computing models, particularly in relation to arithmetic processing and structured computational design. Comprehensive surveys have highlighted the sustained development of spiking neural P systems (SN P systems), including investigations into normal forms, homogeneous structures, software tools, and emerging research directions. These studies provide an updated view of the theoretical foundations and ongoing evolution of SN P systems as a mature paradigm within membrane computing and natural computing research [10].

In parallel, increasing attention has been devoted to implementation-oriented aspects of membrane computing models and their potential for massively parallel computation. Hardware-focused analyses emphasize the suitability of membrane-based computational paradigms for FPGA- and GPU-based realizations, as well as their relevance for high-performance arithmetic and data-intensive applications. Such work underscores the importance of bio-inspired distributed models as viable alternatives to classical sequential architectures in contexts requiring scalable and modular arithmetic computation [11].

Within this broader context, several works have addressed the construction of arithmetic components in spiking neural P systems. Implementations of binary adders and related arithmetic circuits demonstrate the feasibility of designing practical arithmetic modules using spike-based representations and neuron-level rule dynamics, with potential applications in cryptographic and hardware-oriented domains [12]. At the same time, recent extensions of SN P systems have explored enriched rule semantics, including stochastic and probabilistic firing mechanisms, further expanding the modeling and computational capabilities of neural membrane systems [13]. These developments reflect a growing interest in efficient arithmetic processing and modular computational design within membrane computing.

Despite these advances, most existing approaches rely on selective or sequential rule application and often exhibit execution times dependent on operand magnitude or structural complexity. Consequently, the development of compact and reusable arithmetic kernels operating in constant time, while maintaining structural simplicity and scalability, remains an open challenge in the design of membrane–neural computational architectures.

These considerations motivate the need for architectures using *exhaustive semantics*, together with biologically inspired mechanisms like spike aggregation, spike/anti-spike annihilation, and multiset-based computation, as these features simplify arithmetic design and lower neuron counts.

To overcome these limitations, we propose *Hybrid Membrane–Neural (HMN) P systems*, combining membrane hierarchies with exhaustive rule application and spiking neural

computation. This integration yields compact, reusable arithmetic kernels operating in constant time, independent of operand values. The architecture includes:

- **A modular-reduction membrane** $\Pi_{(b \bmod p)}$, which evaluates $(b \bmod p)$ in constant time using a single neuron with exhaustive semantics. This membrane is a fundamental primitive for all modular operations.
- **A synchronization membrane** $\Pi_{(\text{Sync})}$, designed to temporally synchronize incoming spike multisets before processing. This mechanism is needed for asynchronous operands and adds at most two steps: query and synchronized emission.
- **Signed arithmetic membranes** $\Pi_{(xy)}$ and $\Pi_{(x/y)}$, which compute signed multiplication and division via spike/anti-spike encoding.
- **Composite modular kernels** $\Pi_{((xy) \bmod p)}$ and $\Pi_{((x/y) \bmod p)}$, constructed by hierarchically composing the preceding modules within the membrane structure.

A notable property of HMN systems is that *addition and subtraction operate without explicit membranes*, emerging naturally from spike aggregation and spike/anti-spike annihilation. Consequently, specialized architectures are only necessary for multiplication, division, and modular reduction.

By integrating execution under exhaustive semantics, hierarchical modular composition, and bio-inspired spike/anti-spike interaction within multiset dynamics, the HMN P systems perform all modular and non-modular arithmetic kernels in constant time, without dependence on operand size.

This level of efficiency makes it attractive for diverse application domains, such as elliptic-curve arithmetic, error-correcting codes, DSP, machine-learning primitives, finite-field computation, cryptographic hardware, and bio-inspired computational architectures, and for the systematic development of scalable arithmetic libraries within membrane computing.

Beyond the specific constructions presented, the Hybrid Membrane–Neural (HMN) P system introduced in this work is intended as a reusable architectural framework for constant-time arithmetic in membrane–neural computation. By enabling modular composition of arithmetic kernels under exhaustive semantics and hierarchical membrane organization, the HMN model provides a structured basis for scalable arithmetic design in bio-inspired distributed computing. This perspective positions HMN P systems not only as individual constructions but as a foundation for the systematic development of arithmetic and algebraic components within membrane computing.

The organization of the paper is as follows. Section 2 reviews the preliminaries on Spiking Neural P (SNP) systems, SNQ systems, and exhaustive semantics. Section 3 introduces the Hybrid Membrane–Neural (HMN) P systems, while Section 4 describes the arithmetic-kernel designs and their membrane-based implementations. Section 5 provides a detailed comparative analysis of the arithmetic designs formulated within the HMN architecture. Finally, Section 7 summarizes the primary contributions of the work and highlights promising directions for future research.

2. Preliminaries

2.1. Spiking Neural P Systems (SNP Systems)

Spiking Neural P (SNP) systems [2] are computational abstractions of biologically inspired spiking networks. Information is represented by spike symbols, and computation results from the rule-based consumption, generation, and transmission of spikes across directed synaptic pathways.

An SNP system extended to multiple spike types [14] is formally specified as

$$\Pi = (O, \sigma_1, \dots, \sigma_m, \text{syn}, \text{out}),$$

where O is a finite alphabet of spike and anti-spike symbols representing multiple spike types; each neuron $\sigma_i = (n_i, R_i)$ is equipped with an initial spike multiset and a finite set of firing and forgetting rules; and syn is the set of directed synapses. A firing rule $E/u \rightarrow v; d$ consumes the multiset u , validates the enabling condition E , and releases the multiset v after delay d . Forgetting rules delete a nonempty multiset of spikes, producing no output. Neurons evolve in synchrony under a global clock, with rule application limited to one per step. The computation halts once no rule remains applicable or once the output neuron emits its final spike.

2.2. Spiking Neural P Systems with Communication on Request (SNQ Systems)

SNQ systems [4] enhance the SNP model by enabling neurons to request spikes from presynaptic neurons during rule execution. Rules are expressed as E/Qw , where E represents a regular expression and w specifies a nonempty multiset. When a rule fires, the neuron requests the multiset w from every neuron σ_j for which $(j, i) \in syn$. Requested spikes are conceptually replicated, enabling all requests to be fulfilled. As in SNP systems, the evolution proceeds synchronously, and neurons may apply only one rule per step. Computation halts when the system reaches a configuration in which no rule is enabled.

2.3. Exhaustive Semantics in SNP Systems

Under the *exhaustive semantics* model [15], a rule of the form $E/a^c \rightarrow a^p; 0$ applied to a neuron containing k spikes of type a fires $\lfloor k/c \rfloor$ times, producing $\lfloor k/c \rfloor p$ spikes in parallel. Such massively parallel rule execution treats spike multisets as divisible computational resources, leading to a marked improvement in computational throughput.

This property makes exhaustive semantics an effective basis for arithmetic computation, due to its capacity for parallel multiset processing and spike interactions, leading to reductions in neuron count and computation time—advantages leveraged by the HMN architecture introduced in the following section.

3. Hybrid Membrane–Neural P Systems

Classical cell-like P systems [1] are based on a hierarchical membrane structure, where multisets of objects evolve through the application of rules operating within a hierarchy of nested regions. In SNP systems, biologically inspired neuronal units exchange multisets of spikes through directed synapses, and computation is performed through the application of firing and forgetting rules. SNQ systems enhance this model by enabling neurons to *request* spikes from neighboring neurons instead of only receiving transmitted spikes.

The HMN model presented in this work unifies these principles into a single architecture by preserving the hierarchical membrane structure and introducing multiset-driven spike/anti-spike dynamics alongside exhaustive neuronal rule execution. This synergy yields compact arithmetic modules and enables arithmetic operations to execute in constant time.

Formal Definition

A *Hybrid Membrane–Neural P system* (HMN P system) is a tuple

$$\Pi = (O, H, \mu, \Sigma, \sigma, syn_{\text{internal}}, syn_{\text{external}}, m_{\text{out}}),$$

where:

1. $O = \{a_1, \bar{a}_1, \dots, a_n, \bar{a}_n\}$ is the working alphabet, consisting of n spike types and their corresponding anti-spikes.
2. $H = \{1, \dots, p\}$ is the set of membrane labels, with p indicating the total number of membranes.

3. The membrane structure μ is a rooted tree on $\{1, \dots, p\}$, with the root representing the *skin* membrane, as in classical cell-like P systems.
4. For each membrane $i \in H$, $\Sigma(i)$ contains the labels of its direct submembranes.
5. Each membrane i is associated with a (possibly empty) set of neurons $\sigma(i) = \{\sigma_{i,j} \mid 1 \leq j \leq q_i\}$, with $q_i = |\sigma(i)| \geq 0$. In particular, a membrane may or may not contain neurons, depending on the specific HMN system design. Among these neurons, $\sigma_{i,\text{out}}$ denotes the *output neuron*, while the subset $\sigma_{i,\text{in}} \subseteq \sigma(i)$ defines the *input neurons* that may receive spikes from other membranes. Each neuron $\sigma_{i,j}$ is specified by the pair

$$\sigma_{i,j} = (w_{i,j}, R_{i,j}),$$

where the initial spike configuration $w_{i,j}$ is specified as

$$w_{i,j} = (a_1^{r_1} \mid \bar{a}_1^{s_1}) \cdots (a_n^{r_n} \mid \bar{a}_n^{s_n}),$$

with $r_\ell, s_\ell \in \mathbb{N}_{\geq 0}$. For each ℓ , at most one of $a_\ell^{r_\ell}$ or $\bar{a}_\ell^{s_\ell}$ appears; both are omitted if $r_\ell = s_\ell = 0$. All incoming spike or anti-spike multisets are automatically accumulated into $w_{i,j}$, regardless of their source.

6. For membrane i , the internal synaptic connections are specified by:

$$\text{syn}_{\text{internal}}(i) \subseteq \sigma(i) \times \sigma(i).$$

7. For membrane i , the synapses connecting its neurons with the neurons of membrane j are described by:

$$\text{syn}_{\text{external}}(i) \subseteq \bigcup_{j \neq i} (\sigma(i) \times \sigma_{j,\text{in}}).$$

8. The rule set $R_{i,j}$ for neuron $\sigma_{i,j}$ may include:

- *Firing and forgetting rules in SNP systems* have the form $(E/u \rightarrow v(d))$ or $(u \rightarrow \lambda)$, where E is a regular expression over O and u, v are spike patterns of the form

$$u = (a_1^{r_1} \mid \bar{a}_1^{s_1}) \cdots (a_n^{r_n} \mid \bar{a}_n^{s_n}), \quad v = a_t^p,$$

A firing rule $E/u \rightarrow v(d)$ is applicable in neuron $\sigma_{i,j}$ whenever the current multiset $w_{i,j}$ satisfies the condition encoded by E and contains all spikes or anti-spikes required by u . When the rule is applied, it consumes the multiset u and, after a delay of d computation steps, produces the multiset v , which is then sent through all outgoing synapses of $\sigma_{i,j}$. The produced spikes are interpreted as originating from an external environment that can provide an unbounded number of spikes. This assumption is standard in several variants of SNP and SNQ systems, including the formulation of Păun et al. [4].

- *SNQ P systems*, as introduced by Păun et al. [4], extend SNP models by allowing neurons to actively request spikes from other neurons instead of only receiving them passively through synapses. Under this model, a request rule is of the form $Q(a_s^n, C_k)$, where a_s^n requests n spikes of type a_s (or $\bar{a}_s$), and C_k is the membrane from which they are obtained. The rule applies only if the target membrane has at least n spikes (or anti-spikes) of that type. This mechanism is used in $\Pi_{(\text{Sync})}$ (Section 4.2) to collect spikes arriving at different computation steps and emit them together in a single synchronized step.
- *Synaptic SNP-like rules* are associated with internal or external synapses and operate according to exhaustive semantics. It is applicable only if its synapse lies in $\text{syn}_{\text{internal}}(i)$ or $\text{syn}_{\text{external}}(i)$.

All neuronal and synaptic rules operate under exhaustive semantics.

9. The output is taken from membrane m_{out} via its output neuron $\sigma_{m_{out},out}$ and is generally defined as the number of spikes of a prescribed type.

A configuration of Π provides, for every neuron $\sigma_{i,j}$, the current multiset $w_{i,j}$, and the remaining number of steps before the output of any pending rule $E/a_k^c \rightarrow a_r^p(d)$ is released.

A computation is synchronized by a global clock and executed according to the exhaustive mode [15]. During each clock step, the following actions take place:

1. At each step, each neuron checks all applicable SNP and SNQ rules. If several are enabled, one is chosen nondeterministically and applied exhaustively.
2. For each synapse, all applicable synaptic rules are applied using exhaustive semantics.
3. Spikes and anti-spikes generated by neuronal or synaptic rules are propagated through the corresponding synapses, subject to their respective delays.
4. Within a neuron, spike–anti-spike pairs annihilate automatically according to $a_k \bar{a}_k \rightarrow \lambda$ whenever both appear in the same computation step.
5. Within each neuron, spikes accumulate automatically in accordance with the standard dynamics of SNP or SNQ systems.
6. The computation halts once no SNP, SNQ, or synaptic rule is applicable anywhere in the system.

Remark 1. As in classical SNP and SNQ systems, the HMN P system model does not impose additional global restrictions on the admissible rule sets, nor does it provide global guarantees of termination, confluence, or determinism for arbitrary HMN systems. The correctness and constant-time behavior discussed in this paper are properties of the specific arithmetic membranes designed herein and are not claimed as global properties of the HMN model in its full generality.

Although nondeterministic rule selection is allowed by the HMN operational semantics, the arithmetic kernels presented in this work are designed so that such nondeterminism has no effect on either the computed result or the total number of execution steps. In these constructions, rule applicability within each neuron is either mutually exclusive or all enabled rules belong to the same implicit computation phase and are equivalent under exhaustive semantics. Moreover, exhaustive rule application induces an implicit phase structure (e.g., a reduction phase followed by output emission), which guarantees deterministic, convergent, and constant-time behavior independently of the operand values.

In our *Hybrid Membrane–Neural P system* (HMN P system), the membrane structure provides a multilayer hierarchical arrangement of membranes and neurons, while the internal dynamics of each membrane follow spiking neural P system (SNP) rules under exhaustive semantics, achieving high computational efficiency. Anti-spikes make it possible to encode and manipulate signed values, which allows the system to implement fundamental arithmetic operations. Additionally, SNQ and synaptic rules introduce mechanisms that enable the synchronization of spike multisets. The hybrid membrane–neuron framework facilitates the systematic construction of complex neuronal circuits, such as those arising in elliptic-curve arithmetic within sophisticated computational architectures.

4. Arithmetic-Kernel Designs

This section outlines the arithmetic kernels forming the computational backbone of the Hybrid Membrane–Neural P system (HMN P system). The framework supports signed multiplication and signed division, each offered in modular and non-modular form through the membranes $\Pi_{(xy)}$, $\Pi_{(x/y)}$, $\Pi_{((xy) \bmod p)}$, and $\Pi_{((x/y) \bmod p)}$. Together with these operator-specific kernels, two auxiliary modules play a key role in the architecture:

- **Modular-reduction membrane** $\Pi_{(b \bmod p)}$, responsible for computing the non-negative residue $(b \bmod p)$ from a multiset encoding a signed integer. It serves as a reusable primitive that can be composed with other modular membranes.
- **Synchronization membrane** $\Pi_{(\text{Sync})}$, which collects and synchronizes spikes arriving at different computation steps. It guarantees that the arithmetic kernels receive operands as fully synchronized multisets, incurring at most two extra steps (query and synchronized emission).

Because addition and subtraction emerge naturally from spike aggregation and the annihilation of spike/anti-spike pairs, there is no need for explicit membrane-level modules implementing these operations. Instead, addition and subtraction occur naturally from the spiking-based semantics that govern the HMN P system.

The subsections that follow outline the design of each arithmetic kernel and illustrate how the modular composition of low-level membranes (e.g., $\Pi_{(xy)}$, $\Pi_{(x/y)}$, $\Pi_{(\text{Sync})}$, $\Pi_{(b \bmod p)}$) yields compact and reusable architectures. Each kernel is realized through a compact membrane–neuron architecture operating under exhaustive semantics and exhibiting constant-time execution. Their simplicity, reusability, and structural uniformity enable seamless integration into advanced arithmetic constructions arising in elliptic-curve computation, cryptography, finite-field arithmetic, neuromorphic architectures, DSP, and low-power embedded systems.

4.1. Membrane Π_{C_i}

This membrane, denoted by Π_{C_i} , is initially empty and serves exclusively as a spike-storage membrane, passively collecting all incoming spikes. It receives multisets of the form $\{a_i^b, \bar{a}_i^b\}$, which are accumulated in neuron σ_1 while preserving both spike type and multiplicity. This membrane does not emit any output spikes. Its complete architecture is shown in Figure 1.

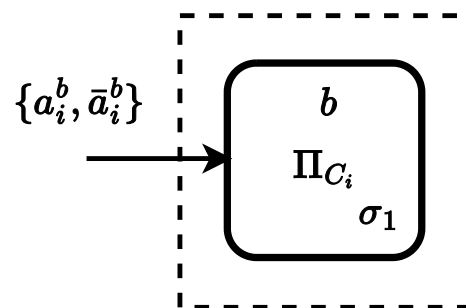


Figure 1. Container membrane Π_{C_i} used to collect and store incoming spikes within its internal neuron.

4.1.1. Formal Definition

Within the Hybrid Membrane–Neural P system model described in Section 3, the container membrane Π_{C_i} constitutes a specific instance of the system

$$\Pi = (O, H, \mu, \Sigma, \sigma, \text{syn}_{\text{internal}}, \text{syn}_{\text{external}}, m_{\text{out}}).$$

- Working alphabet is given by: $O = \{a_0, \bar{a}_0, a_1, \bar{a}_1, \dots, a_n, \bar{a}_n\}$.
- Membrane labels are: $H = \{1\}$; membrane 1 acts as the skin membrane.
- Membrane structure: $\mu = [1]$, membrane 1 contains no child membranes, i.e., $\Sigma(1) = \emptyset$.
- The neuron set is $\sigma(1) = \{\sigma_{1,1}\}$, the single neuron of Π_{C_i} .
- Initial multiset: $w_{1,1} = (a_i^b \mid \bar{a}_i^{b'})$, indicating that the neuron may accumulate spikes or anti-spikes of type a_i for $0 \leq i \leq n$.
- Rule set: $R_{1,1} = \emptyset$.
- Synapses: $\text{syn}_{\text{internal}}(1) = \emptyset$, $\text{syn}_{\text{external}}(1) = \emptyset$.

- Output membrane: $m_{out} = \emptyset$, as no output spikes are produced.

4.1.2. Computation Time

Container membranes do not introduce additional computation steps, as their internal neuron applies no rules and merely accumulates incoming spikes.

4.2. Membrane $\Pi_{(Sync)}$

The membrane $\Pi_{(Sync)}$ contains the container membranes $\Pi_{(C_2)}, \dots, \Pi_{(C_m)}$, together with the query membrane $\Pi_{(Qw)}$. Its function is to synchronize the spikes accumulated in the container membranes, independently of the simulation steps at which they are received. The number of container membranes matches the number of input multisets. A schematic representation is shown in Figure 2.

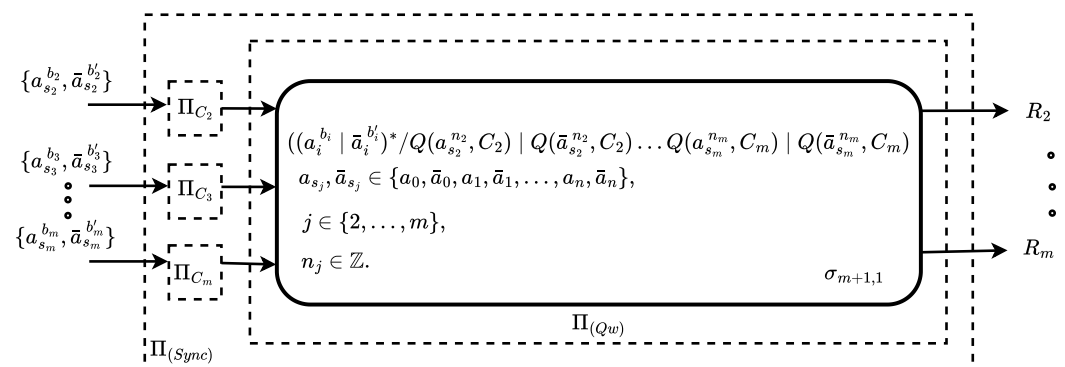


Figure 2. Design of the synchronization membrane $\Pi_{(Sync)}$. Synchronization rules R_2 and R_m ($a_{s_k}^+ / a_{s_k}^{u_k} \rightarrow a_{s_k}^{u_k}$) simultaneously emit the corresponding spike multisets to their target membranes.

4.2.1. Formal Definition

Within the Hybrid Membrane–Neural P system model described in Section 3, the system $\Pi_{(Sync)}$ is instantiated as a specific instance of the general model

$$\Pi = (O, H, \mu, \Sigma, \sigma, syn_{internal}, syn_{external}, m_{out}).$$

- The working alphabet is given by $O = \{a_0, \bar{a}_0, a_1, \bar{a}_1, \dots, a_n, \bar{a}_n\}$.
- The membrane labels are $H = \{1, \dots, m+1\}$, with membrane 1 as the skin membrane, membranes 2– m as container membranes, and membrane $m+1$ contains the synchronization neuron $\Pi_{(Qw)}$.
- The membrane structure is specified as $\mu = [1[2][3] \dots [m+1]]$, with $\Sigma(1) = \{2, \dots, m+1\}$ and $\Sigma(k) = \emptyset$ for $k \geq 2$.
- Each membrane has one neuron: $\sigma(1) = \emptyset$; $\sigma(j) = \{\sigma_{j,1}\}$ for $j = 2, \dots, m+1$.
- Initial multisets are $w_{j,1} = (a_{s_j}^{b_j} | a_{s_j}^{b'_j})$ for $j = 2, \dots, m$; the synchronization neuron starts empty, $w_{m+1,1} = \lambda$.
- There are no internal synaptic connections: $syn_{internal}(j) = \emptyset$ for all j .
- External synapses allow the synchronization neuron $\sigma_{m+1,1}$ to send the synchronized spikes to the m target neurons:

$$syn_{external}(m+1) = \{(\sigma_{m+1,1}, \sigma_{t_1,1}), \dots, (\sigma_{m+1,1}, \sigma_{t_m,1})\},$$

and $syn_{external}(j) = \emptyset$ for all $j \neq m+1$. Here, $\sigma_{t_j,1}$ is the input neuron of membrane t_j that receives the synchronized multiset from $\sigma_{m+1,1}$.

- The synchronization neuron contains the SNQ request rule

$$R_{m+1,1} = \{ Q(a_{s_2}^{n_2} \mid \bar{a}_{s_2}^{n_2}, C_2), \dots, Q(a_{s_m}^{n_m} \mid \bar{a}_{s_m}^{n_m}, C_m) \},$$

collecting the required spike multisets from the container membranes via SNQ request rules.

- Once all queries are fulfilled, $\sigma_{m+1,1}$ emits the synchronized multiset via its synaptic SNP-like rules.

$$R_j: a_{s_j}^+ / a_{s_j}^{u_j} \rightarrow a_{s_j}^{u_j}, \quad 2 \leq j \leq m,$$

each rule is associated with a synapse $(\sigma_{m+1,1}, \sigma_{t_j,1}) \in \text{syn}_{\text{external}}(m+1)$ and is applied under exhaustive semantics in a single computation step.

- Since no output is produced, $m_{\text{out}} = \emptyset$.

In accordance with the SNQ semantics introduced by Păun et al. [4], SNQ request rules in the synchronization neuron actively request and transfer the specified spike multisets from the container membranes to $\sigma_{m+1,1}$ in a single computation step. Once all requested multisets have been collected, synaptic SNP-like rules are applied to emit the synchronized spikes toward their target neurons under exhaustive semantics.

Once all SNQ queries have been satisfied, $\Pi_{(Qw)}$ retrieves the spike multisets from $\Pi_{(C_2)}, \dots, \Pi_{(C_m)}$, accumulates them in $\sigma_{m+1,1}$, and emits the synchronized operand via rules $R_2, \dots, R_m$. If multiple upstream modules provide spikes of the same type, the corresponding container membranes accumulate them independently, while the synchronization neuron queries all containers to verify completeness before emitting the synchronized output.

4.2.2. Computation Time

When $\Pi_{(\text{Sync})}$ is employed, two additional steps are required: the SNQ query step and the subsequent synaptic emission of the synchronized spikes, yielding a total of

$$T_{\Pi_{(\text{Sync})}} = 2$$

4.2.3. Input Synchronization

All arithmetic membranes assume synchronized input. If multisets arrive at different times, $\Pi_{(\text{Sync})}$ can be used as a preprocessing step, adding at most two steps (query and synchronized emission) without changing the target membrane's internal structure.

4.3. Membrane $\Pi_{(b \bmod p)}$

This membrane, denoted as $\Pi_{(b \bmod p)}$, performs modular addition or subtraction of a single signed value. Addition and subtraction are performed implicitly and therefore do not require explicit rules: two membranes sending a_i^b and a_i^c to a common target, the bio-inspired mechanism aggregates them as a_i^{b+c} . Similarly, the neuron performs mutual annihilation upon the simultaneous reception of a_i^b and $\bar{a}_i^c$, yielding a_i^{b-c} when $b > c$ and $\bar{a}_i^{c-b}$ otherwise [16,17]. Thus, this membrane receives a_i^b if $b \geq 0$ and $\bar{a}_i^b$ if $b < 0$. This membrane computes the modular reduction $(b \bmod p)$ independently of the spike type i ; the result is emitted using the same spike type that was received (see Figure 3).

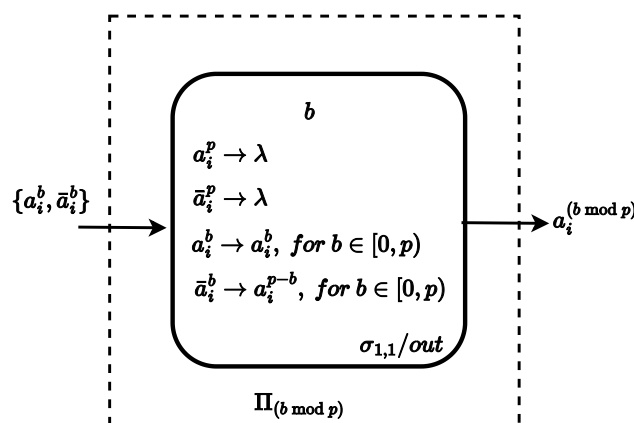


Figure 3. Membrane $\Pi_{(b \bmod p)}$ performing modular reduction on signed spike multiplicities.

4.3.1. Formal Definition

Within the Hybrid Membrane–Neural P system model described in Section 3, the membrane $\Pi_{(b \bmod p)}$ is a particular instance of the general model.

$$\Pi = (O, H, \mu, \Sigma, \sigma, \text{syn}_{\text{internal}}, \text{syn}_{\text{external}}, m_{\text{out}}).$$

- The working alphabet is $O = \{a_i, \bar{a}_i \mid 0 \leq i \leq n\}$.
- $H = \{1\}$, defining a single cell-like membrane with structure $\mu = [1]$.
- The membrane has a single neuron $\Sigma(1) = \sigma_{1,1}$, acting as both input and output.
- The neuron $\sigma_{1,1} = (w_{1,1}, R_{1,1})$, where:

$$w_{1,1} = \begin{cases} a_i^b, & b > 0, \\ \bar{a}_i^{|b|}, & b < 0, \\ \lambda, & b = 0, \end{cases} \quad i \in \{0, \dots, n\},$$

and $R_{1,1} = \{R_1, R_2, R_3, R_4 \mid 0 \leq i \leq n\}$, where:

$$R_1 : a_i^p \rightarrow \lambda, \quad R_2 : \bar{a}_i^p \rightarrow \lambda, \quad R_3 : a_i^b \rightarrow a_i^b, \quad R_4 : \bar{a}_i^b \rightarrow a_i^{p-b}, \quad b \in [0, p).$$

R_3 and R_4 are parametric rule schemata defined for all $b \in [0, p)$ and are counted as single rule types in the structural characterization.

- The membrane has no synapses, i.e., $\text{syn}_{\text{internal}}(1) = \text{syn}_{\text{external}}(1) = \emptyset$.
- The output membrane is $m_{\text{out}} = \sigma_{1,1}$.

4.3.2. Operational Semantics

The membrane computes as follows:

1. In neuron $\sigma_{1,1}$, the value b equals the total number of spikes received in the same computation step. If spikes of type a_i or $\bar{a}_i$ do not arrive simultaneously at $\Pi_{(b \bmod p)}$, a synchronization membrane $\Pi_{(\text{Sync})}$ collects same-type spikes and releases them synchronously in one step, introducing two additional steps (query and transmission).
2. Rules R_1 and R_2 are applied exhaustively, i.e., repeatedly as long as they are applicable, to delete all full multiples of p encoded in a_i or $\bar{a}_i$. Consequently, when no further application of R_1 or R_2 is possible, the remaining multiplicity is necessarily $b \in [0, p)$.
3. Only after the exhaustive application of R_1 and R_2 terminates, exactly one of R_3 or R_4 is applied, emitting $a_i^{(b \bmod p)}$ and preserving the input spike type.

4.3.3. Computation Time

Under exhaustive semantics, the computation completes in at most two steps: optional elimination via R_1 or R_2 , followed by a single application of R_3 or R_4 .

4.4. Membrane $\Pi_{(\text{Sync} \oplus (b \bmod p))}$

Within the Hybrid Membrane–Neural P system model, $\Pi_{(\text{Sync} \oplus (b \bmod p))}$ extends $\Pi_{(b \bmod p)}$ with a preliminary synchronization stage that temporally aligns incoming spikes prior to modular reduction.

4.4.1. Formal Definition

If the spikes encoding b do not arrive simultaneously at $\Pi_{(b \bmod p)}$, a synchronization stage is required. Accordingly, we define the composite system

$$\Pi_{(\text{Sync} \oplus (b \bmod p))} = (O, H, \mu, \Sigma, \sigma, \text{syn}_{\text{internal}}, \text{syn}_{\text{external}}, m_{\text{out}}),$$

which combines the synchronization module $\Pi_{(\text{Sync})}$ and the modular-reduction module $\Pi_{(b \bmod p)}$ under a common skin membrane. Formally:

- $O = O_{(\text{Sync})} \cup O_{(b \bmod p)} = \{a_0, \bar{a}_0, \dots, a_n, \bar{a}_n\}$ is the unified alphabet.
- $H = \{1, 2, 3\}$, where membrane 1 is the skin, and membranes 2 and 3 correspond to $\Pi_{(\text{Sync})}$ and $\Pi_{(b \bmod p)}$, respectively.
- $\mu = [1[2][3]]$, where membrane 2 is responsible for the synchronization stage, while membrane 3 performs the modular-reduction computation.
- $\Sigma(1) = \{2, 3\}$ and $\Sigma(2) = \Sigma(3) = \emptyset$.
- The neurons are inherited from the corresponding component modules: $\sigma(2) = \sigma_{(\text{Sync})}$ and $\sigma(3) = \sigma_{(b \bmod p)}$, while the skin membrane contains no neurons.
- Internal synapses:

$$\text{syn}_{\text{internal}}(2) = \text{syn}_{(\text{Sync})}, \quad \text{syn}_{\text{internal}}(3) = \text{syn}_{(b \bmod p)}.$$

- External synapses:

$$\text{syn}_{\text{external}}(2) = \{(2, 3)\}, \quad \text{syn}_{\text{external}}(3) = \emptyset,$$

transmitting the synchronized multiset from membrane 2 to membrane 3.

- The global rule set is given by

$$R = R_{(\text{Sync})} \cup R_{(b \bmod p)}.$$

- The output membrane is $m_{\text{out}} = 3$.

4.4.2. Computation Time

The architecture operates in two phases: $\Pi_{(\text{Sync})}$ synchronizes the spikes in two steps, after which $\Pi_{(b \bmod p)}$ performs modular reduction in two additional exhaustive steps. Hence,

$$T_{\Pi_{(\text{Sync} \oplus (b \bmod p))}} = 4 \text{ steps.}$$

4.5. Membrane $\Pi_{(xy)}$ for Signed Multiplication

This membrane, denoted $\Pi_{(xy)}$, computes the signed product of two operands x and y within the Hybrid Membrane–Neural architecture. The input is encoded as spike multisets a_2^c or $\bar{a}_2^c$ (multiplicand) and a_1^d or $\bar{a}_1^d$ (multiplier), where (c, d) are the spike-based

instantiations of the abstract operands (x, y) . The dual-spike representation $(a_i, \bar{a}_i)$ enables direct handling of signed integers without auxiliary sign membranes (see Figure 4).

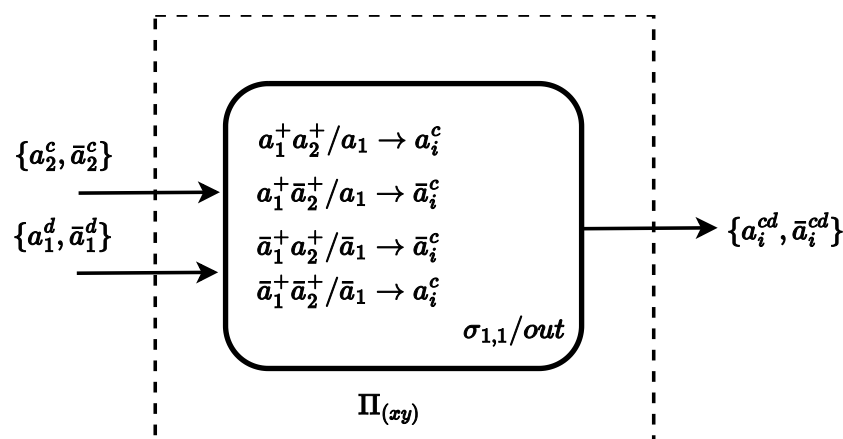


Figure 4. Membrane $\Pi_{(xy)}$ implementing the signed multiplication of two integer operands.

4.5.1. Formal Definition

Within the Hybrid Membrane–Neural P system model described in Section 3, $\Pi_{(xy)}$ is defined as an instance of the general HMN P system.

$$\Pi = (O, H, \mu, \Sigma, \sigma, syn_{internal}, syn_{external}, m_{out}).$$

Formally:

- $O = \{a_0, \bar{a}_0, \dots, a_n, \bar{a}_n\}$ is the working alphabet.
- $H = \{1\}$ with membrane structure $\mu = [1]$.
- $\Sigma(1) = \emptyset$, as no internal submembranes are present.
- The membrane contains a single neuron $\sigma_{1,1}$, acting as both input and output.
- $syn_{internal}(1) = \emptyset$ and $syn_{external}(1) = \emptyset$.
- The initial multiset encodes the signed operands as $w_{1,1} = (a_2^c \mid \bar{a}_2^c)(a_1^d \mid \bar{a}_1^d)$.
- The rule set

$$R_{1,1} = \{R_{11}, R_{12}, R_{13}, R_{14}\}$$

implements the four sign combinations $(+, +)$, $(+, -)$, $(-, +)$ and $(-, -)$ under exhaustive semantics. In a single computation step, all multiplier spikes (a_1 or $\bar{a}_1$) are consumed, and $(c \cdot d)$ spikes of type a_i or $\bar{a}_i$ are emitted according to the resulting sign.

- The output membrane is $m_{out} = \sigma_{1,1}$.

4.5.2. Computation Time

All rules are applied under exhaustive semantics within a single computation step, yielding $T_{\Pi_{(xy)}} = 1$ step.

If the spike multisets encoding (c, d) do not arrive simultaneously, a preceding synchronization phase $\Pi_{(sync)}$ may be used to collect and synchronize all same-type spikes before delivery to $\Pi_{(xy)}$. This optional preprocessing adds at most two additional steps (query and synchronized emission), without modifying the internal structure of $\Pi_{(xy)}$.

The dual-spike representation $(a_i, \bar{a}_i)$ allows the membrane to multiply signed integer values directly through its spiking dynamics, without requiring an additional sign-handling membrane. This enhances the compactness and modularity of the system, allowing the same computational structure to be reused in higher-level tasks such as modular multiplication or differential addition.

4.6. Membrane $\Pi_{(x/y)}$ for Signed Division

This membrane, denoted $\Pi_{(x/y)}$ (see Figure 5), computes the signed division of two operands x and y within the Hybrid Membrane–Neural framework. The dividend x is encoded by a spike multiset a_2^c or $\bar{a}_2^c$, where c is the spike-based instantiation of x , while the divisor y is encoded by a single spike whose type index represents its magnitude and whose polarity represents its sign. Specifically, a positive divisor of magnitude d is encoded by b_d , whereas a negative divisor of magnitude d is encoded by $\bar{b}_d$.

The membrane emits a_i^q or $\bar{a}_i^q$, where $q = \lfloor \frac{|c|}{d} \rfloor$, with the output polarity encoding the sign of the quotient. As in $\Pi_{(xy)}$, the dual-spike representation $(a_i, \bar{a}_i)$ allows direct manipulation of signed operands.

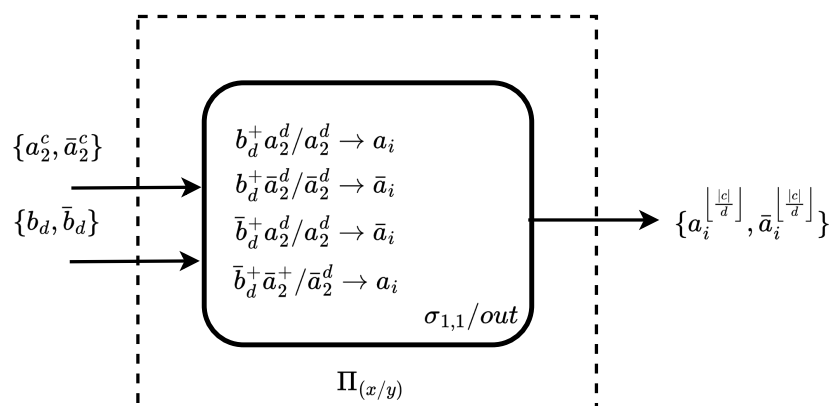


Figure 5. Membrane $\Pi_{(x/y)}$ implementing the signed division of two integer operands.

4.6.1. Formal Definition

Within the Hybrid Membrane–Neural P system model described in Section 3, $\Pi_{(x/y)}$ is instantiated as a specific instance of the general HMN P system.

$$\Pi = (O, H, \mu, \Sigma, \sigma, syn_{\text{internal}}, syn_{\text{external}}, m_{\text{out}}).$$

Formally:

- $O = \{a_0, \bar{a}_0, \dots, a_n, \bar{a}_n\}$ is the working alphabet. Spike types denoted by b_d and $\bar{b}_d$ are mnemonic names for specific elements of O , reserved to encode divisor magnitudes and assumed to be disjointed from those used for operand-value encoding.
- $H = \{1\}$ with membrane structure $\mu = [1]$.
- $\Sigma(1) = \emptyset$, since membrane 1 contains no submembranes.
- The membrane contains a single neuron $\sigma_{1,1}$, acting as both input and output.
- The initial multiset encodes the signed operands as

$$w_{1,1} = (a_2^c \mid \bar{a}_2^c)(b_d \mid \bar{b}_d),$$

where $c = |x|$ and exactly one of a_2^c or $\bar{a}_2^c$ is present, and exactly one of b_d or $\bar{b}_d$ is present.

- The rule set consists of the following finite schema, defined for each $d \in \{1, \dots, n\}$:

$$\begin{aligned} b_d^+ a_2^d / a_2^d &\rightarrow a_i, \\ b_d^+ \bar{a}_2^d / \bar{a}_2^d &\rightarrow \bar{a}_i, \\ \bar{b}_d^+ a_2^d / a_2^d &\rightarrow \bar{a}_i, \\ \bar{b}_d^+ \bar{a}_2^d / \bar{a}_2^d &\rightarrow a_i. \end{aligned}$$

- Each rule is applied under exhaustive semantics. At each application, d spikes encoding the dividend magnitude are consumed, and one output spike is produced, while the divisor spike is preserved since it appears only in the enabling condition. Under exhaustive execution, the enabled rule is applied exactly $\lfloor c/d \rfloor$ times within a single global step, yielding the correct integer quotient.
- $syn_{\text{internal}}(1) = \emptyset$ and $syn_{\text{external}}(1) = \emptyset$.
- The output membrane is $m_{\text{out}} = \sigma_{1,1}$.

4.6.2. Computation Time

Under exhaustive semantics, all applicable rules fire within a single computation step.

$$T_{\Pi_{(x/y)}} = 1 \text{ step.}$$

If the spike multisets encoding (c, d) do not arrive simultaneously, a preceding synchronization membrane $\Pi_{(\text{sync})}$ may be employed to temporally synchronize all spikes and deliver them in a single step to $\Pi_{(x/y)}$. This optional preprocessing adds at most two additional steps (query and synchronized emission), without modifying the internal structure of $\Pi_{(x/y)}$.

Remark 2. The division membrane $\Pi_{(x/y)}$ assumes a nonzero divisor ($d \neq 0$) and computes the integer quotient $q = \lfloor \frac{|c|}{d} \rfloor$.

4.7. Membrane $\Pi_{((xy) \bmod p)}$

This membrane, denoted $\Pi_{((xy) \bmod p)}$, computes the signed modular product $((xy) \bmod p)$ by composing two previously defined arithmetic modules: the signed multiplication membrane $\Pi_{(xy)}$ and the modular-reduction membrane $\Pi_{(b \bmod p)}$. In this architecture, the output of $\Pi_{(xy)}$ is directly fed as input to $\Pi_{(b \bmod p)}$, with both modules contained within the skin membrane (see Figure 6). This modular design avoids rule duplication and supports the reuse of low-level arithmetic components.

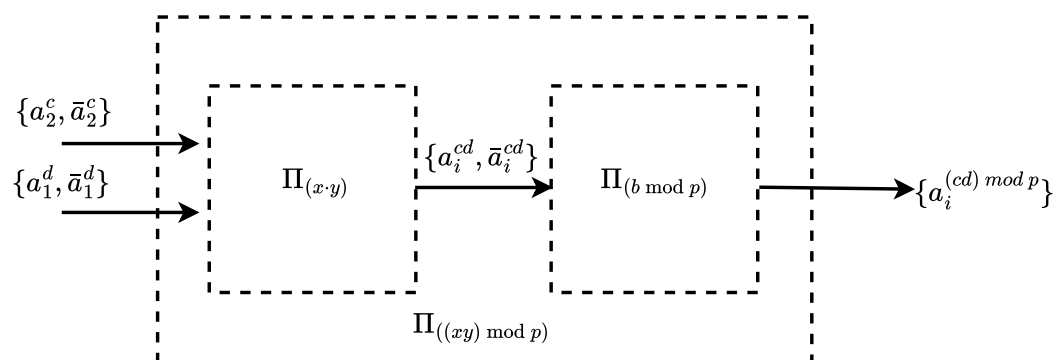


Figure 6. Three-membrane structure computing $((xy) \bmod p)$. Membrane 1 is the skin; membranes 2 and 3 instantiate $\Pi_{(xy)}$ and $\Pi_{(b \bmod p)}$, respectively.

4.7.1. Formal Definition

Within the HMN P system model described in Section 3, $\Pi_{((xy) \bmod p)}$ is defined as an instance of the Hybrid Membrane–Neural P system.

$$\Pi = (O, H, \mu, \Sigma, \sigma, syn_{\text{internal}}, syn_{\text{external}}, m_{\text{out}}).$$

Formally:

- $O = O_{(xy)} \cup O_{(b \bmod p)} = \{a_0, \bar{a}_0, a_1, \bar{a}_1, \dots, a_n, \bar{a}_n\}$ is the working alphabet.
- $H = \{1, 2, 3\}$, where membrane 1 is the skin and membranes 2 and 3 instantiate $\Pi_{(xy)}$ and $\Pi_{(b \bmod p)}$, respectively.
- The membrane structure is $\mu = [1[2][3]]$.
- The membrane contents satisfy $\Sigma(1) = \{2, 3\}$, $\Sigma(2) = \emptyset$ and $\Sigma(3) = \emptyset$.
- Each internal membrane inherits its corresponding neuron set:

$$\sigma(2) = \sigma_{(xy)}, \quad \sigma(3) = \sigma_{(b \bmod p)}.$$

The skin membrane 1 contains no neurons.

- Internal synapses are inherited from the component modules.

$$\text{syn}_{\text{internal}}(2) = \text{syn}_{(xy)}, \quad \text{syn}_{\text{internal}}(3) = \text{syn}_{(b \bmod p)}.$$

- External synapses are defined by

$$\text{syn}_{\text{external}}(2) = \{(2, 3)\}, \quad \text{syn}_{\text{external}}(i) = \emptyset \text{ for } i \neq 2,$$

transmitting the output multiset of $\Pi_{(xy)}$ to the input neuron of $\Pi_{(b \bmod p)}$.

- The global rule set is defined as $R = R_{(xy)} \cup R_{(b \bmod p)}$.
- The output membrane is $m_{\text{out}} = 3$.

4.7.2. Computation Time

All submembranes operate under exhaustive semantics; consequently,

$$T_{((xy) \bmod p)} = T_{\Pi_{(xy)}} + T_{\Pi_{(b \bmod p)}} = 1 + 2 = 3 \text{ steps.}$$

If the spike multisets corresponding to the operands arrive at $\Pi_{(xy)}$ at different computation steps, a preceding synchronization membrane $\Pi_{(\text{Sync})}$ can be employed as a preprocessing module to collect and temporally align all spikes of the same type before they are delivered to the arithmetic kernel. This optional preprocessing introduces at most two additional steps—corresponding to the SNQ query and the synchronized emission—without modifying the internal structure of $\Pi_{(xy)}$ or $\Pi_{(b \bmod p)}$.

This compositional approach preserves modularity, reduces rule overhead, and enables the direct pipeline integration of signed multiplication and modular reduction within the HMN P system framework.

4.8. Membrane $\Pi_{((x/y) \bmod p)}$

This membrane, denoted $\Pi_{((x/y) \bmod p)}$, computes the signed modular division $((x/y) \bmod p)$ by composing two arithmetic modules: the division membrane $\Pi_{(x/y)}$ and the modular-reduction membrane $\Pi_{(b \bmod p)}$. The output spikes produced by $\Pi_{(x/y)}$ are delivered directly to $\Pi_{(b \bmod p)}$, with both modules enclosed within the skin membrane (see Figure 7). This modular scheme avoids rule duplication and enables the reuse of low-level arithmetic components.

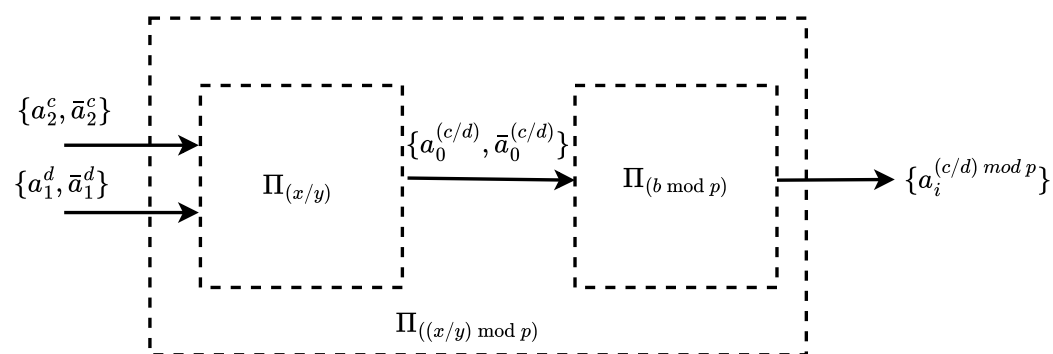


Figure 7. Three-membrane structure computing $(x/y \bmod p)$. Membrane 1 is the skin; membranes 2 and 3 instantiate $\Pi_{(x/y)}$ and $\Pi_{(b \bmod p)}$, respectively.

4.8.1. Formal Definition

Within the Hybrid Membrane–Neural P system model described in Section 3, we instantiate the kernel $\Pi_{((x/y) \bmod p)}$ as a specific instance of the HMN P system.

$$\Pi = (O, H, \mu, \Sigma, \sigma, syn_{internal}, syn_{external}, m_{out}).$$

Formally:

- $O = O_{(x/y)} \cup O_{(b \bmod p)} = \{a_0, \bar{a}_0, a_1, \bar{a}_1, \dots, a_n, \bar{a}_n\}$ is the working alphabet.
- $H = \{1, 2, 3\}$, where membrane 1 is the skin and membranes 2 and 3 implement $\Pi_{(x/y)}$ and $\Pi_{(b \bmod p)}$, respectively.
- The membrane structure is $\mu = [1[2][3]]$.
- The membrane contents satisfy $\Sigma(1) = \{2, 3\}$, $\Sigma(2) = \emptyset$, $\Sigma(3) = \emptyset$.
- Each internal membrane inherits its corresponding neuron set:

$$\sigma(2) = \sigma_{(x/y)}, \quad \sigma(3) = \sigma_{(b \bmod p)}.$$

The skin membrane 1 contains no neurons.

- Internal synapses are inherited from the component modules:

$$syn_{internal}(2) = syn_{(x/y)}, \quad syn_{internal}(3) = syn_{(b \bmod p)}.$$

- External synapses are defined by

$$syn_{external}(2) = \{(2, 3)\}, \quad syn_{external}(i) = \emptyset \text{ for } i \neq 2,$$

transmitting the output multiset of $\Pi_{(x/y)}$ to the input neuron of $\Pi_{(b \bmod p)}$.

- The global rule set is defined as

$$R = R_{(x/y)} \cup R_{(b \bmod p)}.$$

- The output membrane is $m_{out} = 3$.

4.8.2. Computation time

All submembranes operate under exhaustive semantics; consequently,

$$T_{((x/y) \bmod p)} = T_{\Pi_{(x/y)}} + T_{\Pi_{(b \bmod p)}} = 1 + 2 = 3 \text{ steps.}$$

When the spike multisets corresponding to the operands are not delivered simultaneously to $\Pi_{(x/y)}$, a preceding synchronization membrane $\Pi_{(Sync)}$ may be employed

to synchronize all same-type spikes prior to their delivery to the arithmetic kernel. This optional preprocessing introduces at most two additional steps—corresponding to the SNQ query and the synchronized emission—without modifying the internal structure of $\Pi_{(x/y)}$ or $\Pi_{(b \bmod p)}$.

This compositional design preserves modularity, minimizes redundancy, and enables direct coupling between division and modular reduction within the HMN P framework.

4.9. Worked Examples

In all examples, spike aggregation and spike–anti-spike annihilation follow the standard SNP semantics, where spikes of the same-type aggregate by multiplicity and simultaneous occurrences of a_i and $\bar{a}_i$ cancel out pairwise [16,17].

This subsection provides fully developed numerical examples that illustrate the step-by-step dynamics of the main arithmetic cores introduced in Sections 4.5–4.8. The goal is to make the exhaustive execution semantics explicit by showing how spike and anti-spike multiplicities evolve at each global computation step for concrete signed operands.

Example 1 (Signed Modular Reduction). Consider the membrane $\Pi_{(b \bmod p)}$ with modulus $p = 5$. Let $b = 7$, encoded as seven spikes of type a_i . The initial configuration of the unique neuron is

$$w_{1,1} = a_i^7.$$

Under exhaustive semantics, rule $R_1 : a_i^5 \rightarrow \lambda$ is applied once, removing one full multiple of p and leaving two spikes. Since no further application of R_1 is possible, the reduction phase terminates, and rule R_3 emits the remainder:

$$a_i^7 \Longrightarrow a_i^2.$$

The configuration-level evolution is:

Step	a_i	$\bar{a}_i$
0	7	0
1	2	0

confirming that the membrane correctly computes $7 \bmod 5 = 2$ in a single global computation step.

Example 2 (Signed Multiplication). Let $x = -3$ and $y = -4$. In the multiplication membrane $\Pi_{(xy)}$, the operands are encoded as multisets of spikes and anti-spikes of types $a_2, \bar{a}_2$ (for x) and $a_1, \bar{a}_1$ (for y). The initial multiset is

$$w_{1,1} = (\bar{a}_2^3)(\bar{a}_1^4).$$

Under exhaustive semantics, only the rule corresponding to the $(-, -)$ case is enabled:

$$R_{14} : \bar{a}_1^+ \bar{a}_2^+ / \bar{a}_1 \rightarrow a_i^c, \quad c = |x| = 3.$$

This rule is applied once for each available $\bar{a}_1$, i.e., $|y| = 4$ times, producing 12 spikes of type a_i in a single global step:

$$(\bar{a}_2^3)(\bar{a}_1^4) \Longrightarrow a_i^{12}.$$

The configuration-level trace is:

Step	$\bar{a}_2$	$\bar{a}_1$	a_i	$\bar{a}_i$
0	3	4	0	0
1	3	0	12	0

confirming that $T_{\Pi(xy)} = 1$ and that the output polarity matches the sign of xy .

Example 3 (Signed Division). Let $x = -7$ and $y = 2$, hence $x/y = -3$ under truncation-based integer division. The initial configuration of $\Pi_{(x/y)}$ is

$$w_{1,1} = (\bar{a}_2^7)(b_2).$$

The only enabled rule is

$$b_2^+ \bar{a}_2^2 / \bar{a}_2^2 \rightarrow \bar{a}_i,$$

which is applied exhaustively. Each firing consumes two dividend anti-spikes and produces one output anti-spike, while the divisor spike b_2 is preserved. The rule is applied $\lfloor 7/2 \rfloor = 3$ times, yielding

$$(\bar{a}_2^7)(b_2) \Longrightarrow \bar{a}_i^3.$$

The configuration-level trace is:

Step	$\bar{a}_2$	b_2	a_i	$\bar{a}_i$
0	7	1	0	0
1	1	1	0	3

which makes explicit the truncation-based quotient and the correct propagation of the output sign.

Example 4 (Signed Modular Multiplication). Let $x = -3$, $y = 4$, and $p = 5$. The multiplication membrane $\Pi_{(xy)}$ first computes $xy = -12$, emitting $\bar{a}_i^{12}$. This multiset is then transmitted to the modular-reduction membrane $\Pi_{(b \bmod p)}$, where exhaustive reduction rules consume 5 anti-spikes per application. Two full reductions are applied, leaving a remainder of 2 anti-spikes:

$$\bar{a}_i^{12} \Longrightarrow \bar{a}_i^2.$$

Thus, the system correctly computes $(-12) \bmod 5$, which is congruent to 3 modulo 5 under the adopted signed encoding, illustrating how previously defined arithmetic cores are composed without modifying their internal rule sets.

5. Comparative Analysis of Arithmetic Designs in the HMN Framework

Throughout this section, execution times are measured in global HMN computation steps under exhaustive semantics and are independent of the magnitude of the operands. They depend only on the fixed architecture and composition of the corresponding arithmetic kernels.

This section presents a comparative evaluation of arithmetic modules within the Hybrid Membrane–Neural (HMN) P system model under exhaustive semantics. We report neuron counts and performance in terms of simulation steps for the fundamental arithmetic modules introduced in Section 4, and contrast them with representative arithmetic constructions from the SNP and SNQ literature.

5.1. Comparison with Prior Work

Early arithmetic constructions in SNP systems [5] implement addition, subtraction, multiplication, and division using unary representations. However, they exhibit operand-dependent computation times, require multiple auxiliary neurons, and do not provide direct support for signed integers. Subsequent approaches [6,17] reduced neuron counts or relaxed timing constraints, but remained dependent on a large number of rules whose execution time grows linearly with respect to the operand values. In particular, arithmetic constant-time execution limits their scalability as reusable arithmetic components. Single-

neuron arithmetic constructions [18] achieved the lowest structural complexity, but did not eliminate operand-dependent execution. Binary-encoded arithmetic constructions [8] provided explicit complexity bounds in terms of bit-length, but still required a number of simulation steps linear in the operand bit-length. For modular arithmetic, the only previous system supporting all four operations is [9], where repeated subtraction cycles result in data-dependent computation times.

In contrast, the HMN arithmetic kernels operate in constant time for both modular and non-modular arithmetic, require the fewest neurons among existing designs, and natively support signed values via dual-spike encoding.

5.2. Summary of Comparative Results

To facilitate interpretation, we note that the compared constructions differ along several orthogonal dimensions that are not always presented uniformly in the original sources, including: (i) the underlying computational model (SNP, SNQ, or HMN), (ii) the data encoding strategy (unary multiplicity-based, binary, or multi-type spike encoding), and (iii) the assumed temporal semantics (sequential, synchronous, or exhaustive execution). For this reason, these dimensions are discussed explicitly in the text accompanying the tables, rather than being collapsed into a single tabular comparison.

Specifically, the comparison text preceding and following the tables explicitly addresses these dimensions by discussing, for each group of related works, the underlying system model, the adopted data encoding strategy, and the assumed execution semantics.

The following tables summarize and compare neuron counts and simulation-step complexities for non-modular and modular arithmetic circuits across SNP, SNQ, and HMN-based systems.

Tables 1 and 2 summarize the structural and temporal costs of non-modular arithmetic operations, quantified in terms of neuron counts and simulation steps under exhaustive semantics, respectively. Unless otherwise specified in the cited works, all designs assume that operand spikes are delivered synchronously to the corresponding neurons, a common convention in SNP and SNQ arithmetic systems that eliminates the need for explicit synchronization components. For non-modular arithmetic, addition and subtraction are realized intrinsically through spike aggregation and spike–anti-spike annihilation, requiring no internal computation steps and only a single neuron. Addition and subtraction require no internal computation steps, as spike aggregation and spike/anti-spike annihilation occur implicitly upon arrival. Multiplication and division are performed by a single neuron and terminate in one simulation step under exhaustive semantics. Consequently, the HMN arithmetic kernels achieve constant-time execution while minimizing neuron usage.

Table 1. Neuron counts for non-modular arithmetic circuits in SNP and SNQ systems (where k denotes operand bit-length).

ID	Model	Signed	Add	Sub	Mult	Div
[5]	SNP	Yes	10	12	21	25
[6]	SNP	Yes	3	10	13	–
[17]	SNP	Yes	2	2	11	10
[18]	SNP	No	1	1	1	1
[7]	SNQ	No	5	6	6	–
[8]	SNP	Yes	$k + 8$	$k + 13$	$3k + 8$	$5k + 12$
Proposed system	Hybrid	Yes	1	1	1	1

Table 2. Simulation steps for non-modular arithmetic circuits in SNP and SNQ systems.

ID	Model	Signed	Add	Sub	Mult	Div
[5]	SNP	Yes	–	–	–	–
[6]	SNP	Yes	$k + 1$	$k + 2$	$k + 7$	–
[17]	SNP	Yes	n/a	n/a	n/a	n/a
[18]	SNP	No	$\mathcal{O}(x + y)$	$\mathcal{O}(x + y)$	$\mathcal{O}(mn)$	$\mathcal{O}(\lfloor m/n \rfloor)$
[8]	SNP	Yes	$2k + 4$	$2k + 3$	$3k + 5$	$2k + \ell(q) + 4$
Proposed system	Hybrid	Yes	0	0	1	1

5.2.1. Modular Arithmetic Circuits over $GF(p)$

Entries marked as “–” or “n/a” indicate that the corresponding metric is not reported or not defined in the original source. This occurs either because the arithmetic operation is not supported in that construction or because the underlying model does not provide a directly comparable notion of simulation steps. In particular, the SNQ-based design of Jiang et al. [7] does not include a division module, and no neuron count or execution time is therefore reported for this operation in Tables 1 and 2.

Tables 3 and 4 present the corresponding results for modular arithmetic over $GF(p)$, the Galois field of order p , where p is a prime number.

Table 3. Neuron counts for modular arithmetic circuits over $GF(p)$.

ID	Model	Signed	Add Mod p	Sub Mod p	Mult Mod p	Div Mod p
[9]	SNQ	No	4	6	6	6
Proposed system	Hybrid	Yes	1	1	2	2

Table 4. Simulation steps for modular arithmetic circuits over $GF(p)$.

ID	Model	Signed	Add Mod p	Sub Mod p	Mult Mod p	Div Mod p
[9]	SNQ	No	$\mathcal{O}(\lfloor (x + y)/p \rfloor)$	$\mathcal{O}(\lfloor x - y /p \rfloor)$	$\mathcal{O}(x + \lfloor xy/p \rfloor)$	$\mathcal{O}(\lfloor z/p \rfloor)$
Proposed system	Hybrid	Yes	2	2	3	3

The modular-reduction kernel $\Pi_{(b \bmod p)}$ (operations (Add mod p) and (Sub mod p)) is realized using a single neuron and completes in two simulation steps under exhaustive semantics. Modular multiplication and division are constructed compositionally by combining this kernel with the associated non-modular arithmetic modules, resulting in architectures with two neurons and a total of three simulation steps. If operand spikes are not synchronously available, an auxiliary synchronization membrane may be introduced as a preprocessing stage; this mechanism is accounted for separately and does not alter the intrinsic structural or temporal complexity of the core arithmetic modules.

Overall, the tables indicate that the proposed HMN arithmetic kernels achieve constant-step execution for both modular and non-modular operations, while using fewer neurons than previously reported SNP and SNQ constructions and natively supporting signed operands through the dual-spike representation.

5.2.2. Non-Modular Arithmetic Circuits

Remark on Synchronization

As with most existing SNP and SNQ arithmetic designs, the results presented here assume that all spikes encoding a given operand reach the corresponding membrane within

a single computation step. When this assumption does not hold, the HMN framework allows the optional use of a synchronization membrane $\Pi_{(\text{Sync})}$, which synchronizes spikes prior to arithmetic evaluation. Since such synchronization mechanisms are not explicitly accounted for in the majority of related work, their cost is reported separately and is not included in the comparative tables.

For completeness, Table 5 reports the number of evolution rules used by each HMN arithmetic membrane. Some membranes are defined by finite rule schemata, i.e., compact rule templates parameterized over the working alphabet and instantiated, for a fixed alphabet size n , into a finite set of concrete evolution rules. For such membranes, the reported counts correspond to the number of instantiated rules. A direct comparison with rule counts from existing SNP and SNQ arithmetic systems is not always feasible, as such metrics are not uniformly reported and depend on model-specific formalizations and rule granularity. The reported values are therefore intended to characterize the internal structural complexity of the proposed HMN kernels.

Table 5. Number of evolution rules for HMN arithmetic membranes.

Membrane	Number of Rules
Π_{C_i}	0
$\Pi_{(\text{Sync})}$	1
$\Pi_{(b \bmod p)}$	4
$\Pi_{(\text{Sync} \oplus (b \bmod p))}$	5
$\Pi_{(xy)}$	4
$\Pi_{(x/y)}$	$4n$
$\Pi_{((xy) \bmod p)}$	8
$\Pi_{((x/y) \bmod p)}$	$4n + 4$

6. Discussion and Limitations

The Hybrid Membrane–Neural (HMN) arithmetic kernels are conceived as compact, constant-time building blocks within a formal membrane computing framework. While the results demonstrate advantages in neuron counts, execution time, and native support for signed arithmetic, the scope and limitations of the proposed approach must be carefully considered.

First, the HMN model is defined at an abstract level and is primarily intended as a theoretical model of computation. The arithmetic kernels are validated using formal semantic arguments and step-based analysis, rather than through full software or hardware implementations. As a consequence of this level of abstraction, practical issues related to low-level implementation are beyond the scope of the present work.

Second, following standard practice in SNP and SNQ arithmetic systems, the proposed designs assume that all spikes encoding a given operand arrive synchronously at the corresponding membrane. Although the HMN framework allows the optional use of a synchronization membrane to manage asynchronous arrivals, the associated preprocessing cost is not incorporated into the core arithmetic kernels and is reported separately. This assumption must be taken into account when comparing systems under conditions in which synchronization cannot be ensured.

Third, although the individual arithmetic kernels are structurally minimal and terminate in constant time, their composition into more complex systems may necessitate additional membranes and synchronization mechanisms. Consequently, the scalability of complex HMN-based architectures is determined primarily by the design of the surrounding system rather than by the individual kernels themselves.

Fourth, some arithmetic membranes, such as the signed division kernel $\Pi_{(x/y)}$, assume a finite and well-defined input domain induced by the working alphabet. In particular, the divisor magnitude is encoded in the spike type index b_d , with $d \in \{1, \dots, n\}$, so the corresponding rule set is defined by finite schemata instantiated over this range, resulting in $4n$ concrete evolution rules. This restriction reflects a design choice inherent to finite-alphabet HMN systems and does not affect the correctness or constant-time behavior of the kernel within its specified domain.

Finally, the scope of this work is limited to deterministic arithmetic computation under exhaustive semantics; extensions toward stochastic behavior, fault tolerance, and biologically realistic neuron models are left for future research.

7. Conclusions and Future Work

We presented a family of arithmetic kernels within the Hybrid Membrane–Neural (HMN) P system model that natively supports signed addition, subtraction, multiplication, and division, both with and without modular reduction. Exploiting the intrinsic dynamics of spike aggregation and spike–anti-spike annihilation, together with exhaustive semantics, these kernels operate in constant time under exhaustive execution and assuming synchronized input while minimizing neuron usage.

A comparison with representative SNP and SNQ arithmetic constructions demonstrates that the HMN kernels consistently require fewer neurons while achieving operand-independent execution time. In particular, non-modular addition and subtraction are performed intrinsically with zero internal computation steps, whereas multiplication and division are completed in a single simulation step using one neuron.

The proposed kernels follow a modular design that facilitates compositional arithmetic constructions while preserving low structural complexity and constant-time behavior. When operand spikes arrive at different computation steps, an auxiliary synchronization membrane may be employed to enforce the required input synchronization. This preprocessing mechanism operates independently of the arithmetic kernels and is therefore not included in the intrinsic neuron-count and time-complexity measures used for comparison.

As a result, the constant-time guarantees reported in this work apply to the arithmetic kernels themselves under exhaustive semantics and synchronized input, while the optional synchronization overhead is reported separately and does not affect the internal behavior of the kernels.

Future Work

The present work focuses on the design of constant-time arithmetic kernels within the HMN P system framework, which provide a reusable foundation for the construction of more complex computational modules within the HMN model. A natural direction for future research is the composition of these kernels into higher-level cryptographic algorithms, including elliptic-curve point addition and point doubling, as well as Montgomery-type curve arithmetic, where constant-time execution and efficient modular computation are critical.

Beyond cryptographic applications, an additional research direction concerns the investigation of hardware-oriented and neuromorphic realizations of HMN arithmetic kernels. While the exhaustive execution semantics adopted in this work are primarily intended for formal analysis, future studies may explore alternative execution strategies or hardware abstractions that preserve the functional behavior of the kernels while enabling practical implementations.

An implementation and simulation framework for HMN P systems is currently under development and will be released in a public repository to support reproducibility and experimentation.

Author Contributions: Conceptualization, E.V.; Methodology, E.V. and D.-E.V.; Investigation, J.J.G., G.S. (Giovanny Sanchez) and J.-G.A.; Formal analysis, G.S. (Giovanny Sanchez) and J.-G.A.; Validation, G.S. (Giovanny Sanchez) and J.-G.A.; Writing—original draft preparation, E.V., J.J.G. and D.-E.V.; Writing—review and editing, J.J.G., G.S. (Giovanny Sanchez), J.-G.A., G.D., G.S. (Gabriel Sánchez) and L.K.T.; Supervision, E.V. All authors have read and agreed to the published version of the manuscript.

Funding: The authors acknowledge financial support from the Instituto Politécnico Nacional.

Data Availability Statement: No new data were created or analyzed in this study.

Acknowledgments: The authors acknowledge financial support from the Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) and the Instituto Politécnico Nacional.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Păun, G. Computing with membranes. *J. Comput. Syst. Sci.* **2000**, *61*, 108–143. [\[CrossRef\]](#)
2. Ionescu, M.; Păun, G.; Yokomori, T. Spiking neural P systems. *Fundam. Inform.* **2006**, *71*, 279–308. [\[CrossRef\]](#)
3. Păun, G. *Membrane Computing: An Introduction*; Springer: Berlin/Heidelberg, Germany, 2002.
4. Pan, L.; Păun, G.; Zhang, G.; Neri, F. Spiking Neural P Systems with Communication on Request. *Int. J. Neural Syst.* **2017**, *27*, 1750042. [\[CrossRef\]](#) [\[PubMed\]](#)
5. Zeng, X.; Song, T.; Zhang, X.; Pan, L. Performing Four Basic Arithmetic Operations with Spiking Neural P Systems. *IEEE Trans. Nanobiosci.* **2012**, *11*, 366–374. [\[CrossRef\]](#) [\[PubMed\]](#)
6. Peng, X.; Liu, J.; Liang, W. Several Arithmetic Operations on Spiking Neural P Systems. In *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*; Atlantis Press: Paris, France, 2013; pp. 270–273.
7. Jiang, Y.; Kong, Y.; Zhu, C. Implementation of Arithmetic Operations by SN P Systems with Communication on Request. *Int. J. Comput. Commun. Control* **2018**, *13*, 353–364. [\[CrossRef\]](#)
8. Chen, X.; Guo, P. Spiking Neural P Systems for Basic Arithmetic Operations. *Appl. Sci.* **2023**, *13*, 8556. [\[CrossRef\]](#)
9. Rangel, J.L.I.; Arroyo, M.I.; Vázquez, E.; Avalos, J.G.; Sanchez, G. New Compact Finite-Field Arithmetic Circuits Over $GF(p)$ Based on Spiking Neural P Systems with Communication on Request Implemented in a Low-Cost FPGA. *IEEE Embed. Syst. Lett.* **2024**, *16*, 295–298. [\[CrossRef\]](#)
10. Cabarle, F.G.C. Thinking about spiking neural P systems: Some theories, tools, and research topics. *J. Membr. Comput.* **2024**, *6*, 148–167. [\[CrossRef\]](#)
11. Zhang, G.; Shang, Z.; Verlan, S.; Martínez-del-Amor, M.Á.; Yuan, C.; Valencia-Cabrera, L.; Pérez-Jiménez, M.J. An overview of hardware implementation of membrane computing models. *ACM Comput. Surv.* **2020**, *53*, 90:1–90:38. [\[CrossRef\]](#)
12. Ochirbat, O.; Ishdorj, T.-O.; Cichon, G. An error-tolerant serial binary full-adder via a spiking neural P system using HP/LP basic neurons. *J. Membr. Comput.* **2020**, *2*, 42–48. [\[CrossRef\]](#)
13. Lazo, P.P.L.; Cabarle, F.G.C.; Adorna, H.N.; Yap, J.M.C. A return to stochasticity and probability in spiking neural P systems. *J. Membr. Comput.* **2021**, *3*, 45–57. [\[CrossRef\]](#)
14. Ionescu, M.; Păun, G.; Pérez-Jiménez, M.J.; Rodríguez-Patón, A. Spiking Neural P Systems with Several Types of Spikes. *Int. J. Comput. Commun. Control* **2011**, *6*, 141–152. [\[CrossRef\]](#)
15. Zhang, X.; Luo, B.; Fang, X.; Pan, L. Sequential Spiking Neural P Systems with Exhaustive Use of Rules. *Biosystems* **2012**, *108*, 52–62. [\[CrossRef\]](#) [\[PubMed\]](#)
16. Pan, L.; Păun, G. Spiking Neural P Systems with Anti-Spikes. *Int. J. Comput. Commun. Control* **2009**, *4*, 273–282. [\[CrossRef\]](#)
17. Liu, X.; Li, Z.; Liu, J.; Liu, L.; Zeng, X. Implementation of Arithmetic Operations with Time-Free Spiking Neural P Systems. *IEEE Trans. Nanobiosci.* **2015**, *14*, 617–624. [\[CrossRef\]](#) [\[PubMed\]](#)
18. Frías, T.; Díaz, C.; Sánchez, G.; García, G.; Avalos, J.G.; Pérez, H. Four Single Neuron Arithmetic Circuits Based on SN P Systems with Dendritic Behavior, Astrocyte-like Control and Rules on the Synapses. *IEEE Lat. Am. Trans.* **2018**, *16*, 38–45. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.