Reaction Systems and Synchronous Digital Circuits

Zeyi Shang, Sergey Verlan, Ion Petre, Gexiang Zhang

molecules

MDPI

Article

Reaction Systems and Synchronous Digital Circuits

Zeyi Shang [1,2,†], Sergey Verlan [2,†], Ion Petre [3,4] and Gexiang Zhang [1,*]

1 School of Electrical Engineering, Southwest Jiaotong University, Chengdu 611756, Sichuan, China; zeyi.shang@lacl.fr
2 Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est Créteil, 94010 Créteil, France; verlan@u-pec.fr
3 Department of Mathematics and Statistics, University of Turku, FI-20014 Turku, Finland; ion.petre@utu.fi
4 National Institute for Research and Development in Biological Sciences, 060031 Bucharest, Romania
* Correspondence: zhgxdylan@126.com
† These authors contributed equally to this work.

Academic Editors: Quan Zou, Xiangxiang Zeng and Alfonso Rodríguez-Patón
Received: 25 March 2019; Accepted: 28 April 2019; Published: 21 May 2019

Abstract: A reaction system is a modeling framework for investigating the functioning of the living cell, focused on capturing cause–effect relationships in biochemical environments. Biochemical processes in this framework are seen to interact with each other by producing the ingredients enabling and/or inhibiting other reactions. They can also be influenced by the environment seen as a systematic driver of the processes through the ingredients brought into the cellular environment. In this paper, the first attempt is made to implement reaction systems in the hardware. We first show a tight relation between reaction systems and synchronous digital circuits, generally used for digital electronics design. We describe the algorithms allowing us to translate one model to the other one, while keeping the same behavior and similar size. We also develop a compiler translating a reaction systems description into hardware circuit description using field-programming gate arrays (FPGA) technology, leading to high performance, hardware-based simulations of reaction systems. This work also opens a novel interesting perspective of analyzing the behavior of biological systems using established industrial tools from electronic circuits design.

Keywords: reaction systems; synchronous digital circuits; field-programming gate arrays

1. Introduction

Reaction systems have been introduced in [1] as a formalism for describing the functioning of the living cell by following the interactions between biochemical reactions and the cellular environment, see also [2] for a recent survey. These interactions are fundamentally based on two mechanisms, facilitation and inhibition: the products of reactions may be used by other reactions and may inhibit others, while the environment may add additional ingredients in every step of the process. The framework of reaction systems is a model of biocomputations where the configuration of the system is created by the product of all reactions that were enabled in the previous step, plus the additional contribution of the environment. The facilitation-inhibition mechanism gives reaction systems the ability to follow up explicitly on the cause-effect relationships in a biochemical process, helping to answer questions around why a certain property arises [3,4].

The research on reaction systems has flourished in the last few years along two broad directions. On the one hand, reaction systems have been investigated for their mathematical and computational properties as a model for interactive biocomputation, on topics such as minimal systems [5,6], functions and state sequences [7,8], timed versions [9–12], modular decompositions [13], equivalence properties [14–16]. On the other hand, reaction systems have been studied a modeling

framework for capturing realistic biological processes and for enhancing their analytic capabilities; topics include model checking properties [17–19], modeling of the heat shock response [20], of the self-assembly of intermediate filaments [21], and of the ErbB signaling pathway [22], bifurcation and multi-stability properties [23].

There has been a growing interest also in the simulation of the behavior of reaction systems, supporting both lines of research described above. There are currently two different simulators of interactive processes of reaction systems. The `brsim/WEBRSIM` simulator is a Haskell-based implementation with a user-friendly web interface, providing also a few simple model checking analysis options. It is currently the fastest available central processing unit (CPU)-based simulator. The `HERESY` simulator [22] is based on a graphics processing unit (GPU) implementation with compute unified device architecture (CUDA) and is especially efficient for very large reaction systems with hundreds of reactions. It also has a slower CPU-based implementation.

The main contribution of this paper is the first attempt to implement reaction systems in the hardware. We first present the links between two different models: reaction systems and synchronous digital circuits. Our main observation is that interactive processes in reaction systems are similar with the calculations of digital circuits and we establish this in a formal sense. Based on this we build a new hardware-based simulator for reaction systems by simply translating a reaction system into a digital synchronous circuit, and simulate it using a fast FPGA-based implementation. This implementation is much faster than the previous software-based simulators, bringing speeds of more than 10^8 steps per second, thus allowing a speed-up of order 2.5×10^5 with respect to best existing hardware (and software) implementations. Hence, this work shows us how to perform high-speed and large-scale simulations of reaction systems, which opens a way for an efficient investigation of big biological models, like for example the ErbB signaling pathway [24]. This can potentially speed-up the drug discovery by pointing out different possible drug targets having the maximal desired effect helping to target the further experiments [25,26]. At the same time, this paper is the first study that shows how formal tools like Mealy and Moore automata can be used to investigate the behavior of reaction systems and thus of corresponding biological systems. This also allows to use high-quality industrial tools from the circuit design area to analyze and optimize the obtained circuits (thus the initial system). By performing slight modifications, the proposed method can be used to simulate arbitrary Boolean networks, which opens many interesting perspectives. While there are several works on FPGA simulation of Boolean networks, e.g., [27–29], our article differs from them by highlighting the theoretical link between the model of Boolean networks and sequential switching circuits, hence implementing the corresponding simulation in most efficient way.

2. Preliminaries

We assume basic familiarity with the notions of Boolean (switching) functions and formulas. There are many books available presenting these notions, we suggest [30] for an introduction. Below we recall the differences between Boolean and switching algebras and circuits.

A Boolean algebra [31] is a distributive and complemented lattice. A switching algebra firstly developed by Shannon in [32] can be seen as a restriction of a Boolean algebra to two elements: 0 and 1. It is also called a two-element Boolean algebra. The primary applications of switching algebra are in digital circuit design and Boolean (two-valued) logic, see [33] for more details. Traditionally, in switching algebra symbols $\cdot$, $+$, and $'$ are used for conjunction, disjunction and negation, respectively. In this paper we will use the standard logical notations for these functions: $\vee$, $\wedge$, and $\bar{x}$.

A switching function is any expression in switching algebra. The evaluation of the function is done in an ordinary way by assigning values 0 or 1 to corresponding arguments and after that evaluating the result by performing ordinary Boolean transformations. A switching circuit having n inputs and m outputs is computing a switching function of the form $f : \{0,1\}^n \to \{0,1\}^m$.

We remark that in the present-day literature the term switching is often replaced by Boolean, which has the meaning of the former one.

2.1. Sequential Circuits

There are two types of (switching) circuits: combinatorial, where the value of the output is a function of only current input values and sequential, where the value of the output depends on the input and also on the state of the circuit. The state allows us to memorize past values and to perform decisions based on the partial history of the computation. Hence, in this case the value of the function may be different for same inputs at different time steps (usually corresponding to the master clock pulses that drive the circuit).

So the functioning of a sequential switching circuit with n inputs and m outputs and s binary-state variables can be described by the following equations:

$$\begin{aligned} Q(t+1) &= F(Q(t), X(t)) \\ Y(t) &= G(Q(t), X(t)), \end{aligned} \tag{1}$$

where $X(t) = (x_1(t), \ldots, x_n(t))$ is the vector of input variables at time $t \geq 0$, $Y(t) = (y_1(t), \ldots, y_m(t))$ is the vector of output variables at time t, $Q(t) = (q_1(t), \ldots, q_s(t))$ is the vector of internal states at time t, $F : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^s$ and $G : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^m$.

Each sequential circuit is associated with a truth-table with its columns headed (in order):

$$Q(t); X(t); Q(t+1); Y(t).$$

An equivalent description of such circuits was shown by Mealy using Mealy automata [34]. It corresponds to a finite state automaton with input and output where the transitions are labeled by two values corresponding to the input and the output. The transition is applicable if the current input corresponds to the one indicated on the transition, and then the automaton outputs the corresponding output value. A similar representation can be done by Moore automata [35]. In this case the output depends only on the previous state and it is indicated beside the state label. Both models are equivalent, however there is a one step output delay if using Moore model.

Example 1. *Let us consider the example presented in [34]. The sequential circuit is described by the following equations ($n = 1$, $m = 1$ and $s = 2$).*

$$\begin{aligned} q_1(t+1) &= \bar{q}_1(t) \wedge \bar{q}_2(t) \vee \bar{x}_1(t) \wedge \bar{q}_2(t) \\ q_2(t+1) &= q_1(t) \wedge \bar{q}_2(t) \vee x_1(t) \wedge q_1(t) \\ y_1(t) &= \bar{q}_1(t) \wedge \bar{q}_2(t). \end{aligned} \tag{2}$$

These equations correspond to the truth table (computed by listing all possible combinations of values for $q_1(t)$, $q_2(t)$ and $x(t)$) presented in Table 1. The corresponding Mealy and Moore automata are depicted on Figure 1. It can be seen that these automata are a convenient way to represent the truth table. For example, being in state 10 *and following a transition labeled by* 0*, yields the state* 11 *and the output* 0 *(at the next step in the case of Moore automaton), corresponding to the fifth line of the truth table.*

Table 1. The truth table for Example 1.

$q_1(t)$	$q_2(t)$	$x_1(t)$	$q_1(t+1)$	$q_2(t+1)$	$y_1(t)$
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	0

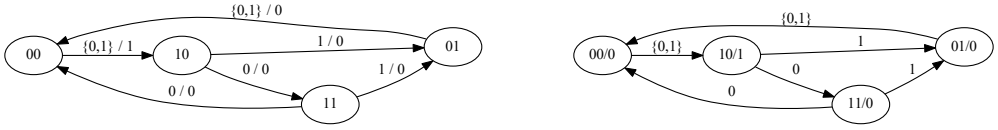

Figure 1. The Mealy (**left**) and Moore (**right**) automata for the circuit described by Equation (2). The label of the state corresponds to the value of the vector (q_1, q_2). In the case of Moore automaton it additionally contains the value of the output variable y. The label of the transition corresponds to the value of the input variable x. In the case of Mealy automaton, it additionally contains the value of the output variable y.

2.2. Reaction Systems

Below we briefly recall the definition of reaction systems given in [1,2].

Definition 1. *A reaction is a triplet $a = (R, I, P)$, where R, I, P are finite nonempty sets with $R \cap I = \emptyset$. If S is a set such that $R, I, P \subseteq S$, then a is a reaction in S.*

Informally, reaction allows us to define causal effects between the production of the result P and the presence of reactants R and the absence of inhibitors I. Formally, we defined $res_a(X) = P$, if and only if $R \subseteq X$ and $I \cap X = \emptyset$. This operation can be generalized to a set of reactions $\mathcal{A}$ in a standard manner. Finally, the set of all reactions over a set S is denoted as $rac(S)$.

Definition 2. *A reaction system (RS), is an ordered pair $\mathcal{A} = (S, A)$ such that S is a finite set, and $A \subseteq rac(S)$.*

Definition 3. *Let $\mathcal{A} = (S, A)$ be a RS and let $n \geq 0$ be an integer. An (n-step) interactive process in $\mathcal{A}$ is a pair $\pi = (\gamma, \delta)$ of finite sequences such that $\gamma = C_0, \ldots, C_n$ and $\delta = D_0, \ldots, D_n$, where $C_0, \ldots, C_n, D_0, \ldots, D_n \subseteq S$, $D_0 = \emptyset$, and $D_i = res_{\mathcal{A}}(D_{i-1} \cup C_{i-1})$ for all $i \in \{1, \ldots, n\}$.*

Informally, an interactive process allows to compute a time series of the values of variables from S based on the input provided at each step t by the context C_t.

The reaction systems model abstracts away from the various numerical details of biochemical reactions, and rather it only indicates whether a resource is present or not in the system, and how they trigger or block the execution of a certain reaction. This is best described as seen in the definitions above through a set-theoretical framework, where each configuration of the system is a subset of the reactant set. The environment is an active present of the semantic of reaction system and in each step of the interactive process of a reaction system, it contributes potentially new, additional resources to the current configuration. The current configuration then determines the next one by triggering all the enabled reactions and having them generate through their products the next configuration of the system. All the currently existing resources are excluded from the next configuration, unless they were produced by one of the enabled reactions. This is the basic "non-permanency" principle of reaction systems, proposing the idea that maintaining a resource in a system is a matter that has to be actively supported by the system.

Many versions of reaction systems have been proposed, adding various features to the basic model, such as a numerical dimension of the reactants, describing how many are available in a configuration [9], and systems with durations, where there is an explicit, potentially non-zero life duration of a reactant, keeping it into the system for several steps without additional support from the enabled reactions [10]. All of these versions were proved to be equivalent with the basic model from the point of view of their computing power [2]. We focus in this article only on the basic version of the reaction systems, as defined above.

3. Results

In this section we will show how it is possible to transform a reaction system to a switching circuit and conversely.

3.1. From Reaction Systems to Switching Circuits

First we introduced a normal form for a reaction system under an interactive process.

Definition 4. *A reaction system $\mathcal{A} = (S, A)$ is said to be in a normal form with respect to the interactive process $\pi = (\gamma, \delta)$ if*

- *for any $a = (R, I, P) \in A$ it holds $|P| = 1$ (i.e., only one product is allowed per reaction),*
- $\bigcup_{(R,I,P)\in A} P \cap \bigcup_{i\geq 0} C_i = \emptyset$ *(i.e., the set of products is disjoint with the set of contexts).*

Theorem 1. *For any reaction system $\mathcal{A} = (S, A)$ and any interactive process $\pi = (\gamma, \delta)$ such that $C \subseteq Z$, for any $C \in \delta$ it is possible to construct an iterative process $\pi' = (\gamma, \delta')$ and a reaction system $\mathcal{A}' = (S', A')$ in normal form with respect to π' such that δ is a projection of δ' with respect to S.*

Proof. Initially $S' = S$. First we add to A' all reactions from A that have a single product.

Next, if $a = (R, I, P)$ with $|P| > 1$, than we add to A' the reactions $(R, I, x), x \in P$.

Let $T = \{x \mid (R, I, x) \in A'\}$. Now, if there is a symbol $X \in Z \cap T$ then we add a new background symbol X' to S' and we replace the reaction (R, I, X) by (R, I, X'). We also add to A' the set of reactions $\{(R \setminus \{X\} \cup \{X'\}, I, P) \mid X \in R\} \cup \{(R, I \setminus \{X\} \cup \{X'\}, P) \mid X \in I\}$ (i.e., we made a copy of each reaction involving X replacing it by X').

Now consider the interactive process π', which is obtained by feeding $\mathcal{A}'$ the contexts γ. Obviously, the reaction system $\mathcal{A}'$ is in the normal form with respect to π'. Moreover, because no rule was deleted and the added rules introduce primed symbols, which act as aliases for their non-primed versions it is easy to see that the projection of δ' over S yields δ. □

We can define the input for a reaction system $inp(\mathcal{A})$ as the set of all possible context symbols: $inp(\mathcal{A}) = \{Z \in \delta \mid$ for any interactive process $\pi = \{\gamma, \delta\}\}$. Similarly, we might be interested by particular symbols that can appear in the result. So we define can define the output of a RS as a projection of S: $out(\mathcal{A}) \subseteq S$.

Now we construct the Equation (1) using the method from [1], that transforms a reaction system to a Boolean formula in disjunctive normal form (DNF).

Suppose that we have a reaction system $\mathcal{A} = (S, A)$ in a normal form with input $\mathcal{I}$. Then each group of reactions having the same product $A_p = \{(R_a, I_a, p) \mid a \in A\}$ can be seen as the following equation:

$$p(t+1) = \bigvee_{(R_a,I_a,p)\in A_p} \left(\bigwedge_{X\in R_a} X(t) \wedge \bigwedge_{Y\in I_a} \bar{Y}(t) \right). \tag{3}$$

In order to compute the output of the switching circuit we add the following equation (since it is a projection, we may omit it if it is clear from the context):

$$y(t) = y(t), \quad \text{for all } y \in out(\mathcal{A}). \tag{4}$$

Equations (3) and (4) are of the form of Equation (1), so they define a switching circuit.

Moreover, since sets of reactants and inhibitors are disjoint, formula (3) is in DNF.

Example 2. *Consider the reaction system* $\mathcal{A}_n = (S_n, A_n)$, $n > 1$ *from [1] that defines a binary counter. For* $n > 1$, *let* $S_n = \{e_0, \ldots, e_n\}$. *The set of reactions is defined as follows.*

$$\begin{aligned} A_n &= \{(\{e_i\}, \{e_j\}, \{e_i\}) \mid 0 \leq j < i \leq n\} \cup \\ &\quad \{(\{e_0, \ldots, e_{i-1}\}, \{e_i\}, \{e_i\}) \mid 0 < i \leq n\}. \end{aligned}$$

In the interactive process e_0 *is the input and* e_i, $1 \leq i \leq n$ *are the output symbols. So the system is in the normal form. We construct the equations for the switching circuit:*

$$e_i(t+1) = \bigvee_{0 \leq j < i \leq n} (e_i(t) \wedge \bar{e}_j(t)) \vee \bigvee_{1 \leq i \leq n} \left(\bar{e}_i(t) \wedge \bigwedge_{0 \leq k \leq i-1} e_k(t) \right)$$

The corresponding Moore machine for $n = 3$ *is given on Figure 2. From the picture it can be clearly seen that this circuit implements a binary counter.*

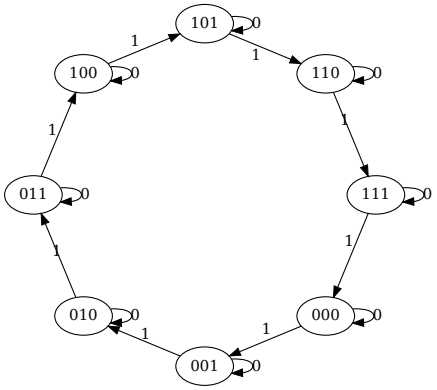

Figure 2. Moore machine for Example 2 and $n = 3$. The state label corresponds to the values of the vector (e_3, e_2, e_1) and the transitions are labeled by the value of e_0. The output is the label of the state.

Example 3. *In [21], a model for the self-assembly of intermediate filaments from vimentin tetramers is presented. We consider the first model from that paper (the other more complex variants of the model can be consulted as examples provided with our compiler in [36]). It is defined as follows.*

The background set is $S = \{O, H, F, d\}$ *and the input set is* $\{T\}$. *The reactions are the following (d is the dummy inhibitor):*

$$\begin{aligned} &(\{T\}, \{d\}, \{O\}), \\ &(\{O\}, \{d\}, \{H\}), \\ &(\{H\}, \{d\}, \{F\}), \\ &(\{F\}, \{d\}, \{F\}). \end{aligned}$$

Using Equations (3) and (4) we obtain the following sequential circuit:

$$\begin{aligned} d(t+1) &= 0 \\ O(t+1) &= T(t) \wedge \bar{d}(t) \\ H(t+1) &= O(t) \wedge \bar{d}(t) \\ F(t+1) &= (H(t) \wedge \bar{d}(t)) \vee (F(t) \wedge \bar{d}(t)). \end{aligned}$$

The Moore machine for this circuit is depicted on Figure 3. It can be immediately deduced that there is a steady loop between states 101, 110 *and* 111 *corresponding to the last rule that keeps F indefinitely once produced.*

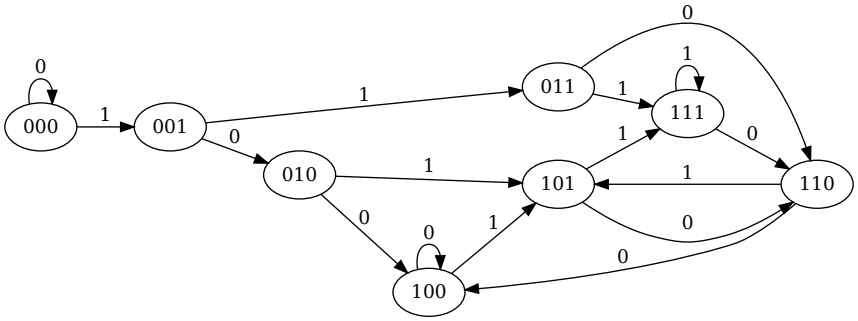

Figure 3. Moore machine for Example 3. The state label corresponds to the values of the vector (F, H, O) and the transitions are labeled by the value of T. The output is the label of the state.

3.2. From Switching Circuits to Reaction Systems

Now we will show how to construct a reaction system starting from a sequential switching circuit. Let C be a circuit with n inputs, m outputs and s internal states described by Equation (1). Without loss of generality we can suppose that F and G are in disjunctive normal form.

Then we construct a reaction system with input $\mathcal{A} = (S, A, I)$, where $S = Q \cup Y$ and $I = X$. The reactions from A are defined as follows:

Let a be a conjunction $a = a_1 \wedge \cdots \wedge a_k$, $k > 0$. We define

$pos(a) = \{a_s \mid 1 \leq s \leq k$ and k is a positive literal$\}$

$neg(a) = \{a_s \mid 1 \leq s \leq k$ and k is a negative literal$\}$.

Then an equation $q_i(t+1) = F_i(Q(t), X(t)) = \bigvee_{1 \leq s \leq p} c_s$, where c_s is a conjunction of literals from $Q(t)$ and $X(t)$ and $p > 0$ can be written as following set of reactions:

$$(pos(c_s), neg(c_s), \{q_i\}), \quad 1 \leq s \leq p.$$

Now, the initial values of state variables of the circuit give the value C_0 of the initial context for any interactive process π for this RS. It can be immediately seen that for any sequence of input values for C, feeding same values as contexts for $\mathcal{A}$ give the same output sequence.

Example 4. *Consider the circuit that implements a sequence detector and outputs 1 if the sequence* 1101 *is detected as input. The corresponding Mealy machine is depicted in Figure 4 and the corresponding truth table is given in Table 2.*

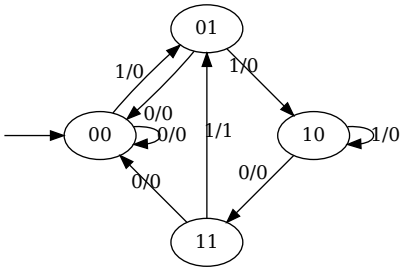

Figure 4. Mealy machine for the 1101 sequence detector. It outputs 1 when the corresponding sequence is encountered as input. The state label corresponds to the vector (q_2, q_1).

From this table we can deduce the state equations of the circuit (x being the input bit and y the output result):

$$\begin{aligned} q_2(t+1) &= \bar{q}_2(t) \wedge q_1(t) \wedge x \vee q_2(t) \wedge \bar{q}_1(t) \\ q_1(t+1) &= \bar{q}_2(t) \wedge \bar{q}_1(t) \wedge x \vee q_2(t) \wedge \bar{q}_1(t) \wedge \bar{x} \vee q_2(t) \wedge q_1(t) \wedge x(t) \\ y(t) &= q_2(t) \wedge q_1(t) \wedge x(t) \end{aligned}$$

Using the above algorithm these equations are transformed to the following reaction system (where d is the dummy inhibitor and initially the system is empty):

$$\{x\}\{q_1, q_2\}\{q_1\},$$
$$\{q_2\}\{q_1, x\}\{q_1\},$$
$$\{q_1, q_2, x\}\{d\}\{q_1\},$$
$$\{q_1, x\}\{q_2\}\{q_2\},$$
$$\{q_2\}\{q_1\}\{q_2\},$$
$$\{q_1, q_2, x\}\{d\}\{y\}.$$

Table 2. The truth table for Example 4.

$q_2(t)$	$q_1(t)$	$x(t)$	$q_2(t+1)$	$q_1(t+1)$	$y(t)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1

4. Discussion

The translation of reaction systems to synchronous circuits gives several advantages. First, the Mealy/Moore machine representation might allow a better understanding of the functioning of the system and of its invariants. Next, there exist many powerful industrial tools allowing the analysis, the minimization and the verification of digital circuits, hence they can be used to transform or minimize the circuit. For example, applying tools from Vivado Design Suite 2018.2 [37] on the translation of the reaction system describing ErbB signaling pathway taken from [22] allowed to reduce the size of the model by 50% by performing cell identification and constant propagation. Finally, the translation to circuits allows us to implement reaction systems in digital hardware in order to perform fast experiments of systems of huge size. In most of the cases, the running speed of FPGA clock can be achieved, yielding a simulation performing at 1–10 ns per step.

The converse translation is also interesting as it allows to use industrial tools for circuit design in order to produce reaction systems. Moreover, many of such tools come with a library of predefined circuits and it should be relatively simple to design complex reaction systems, e.g., simulating a RAM or an Ethernet controller.

Another interesting implication of our research is that, due to the similarity between RS and Boolean networks, the developed method can be directly applied to transform Boolean networks to hardware circuits allowing to use high quality analysis tools and high-speed hardware simulation. This approach is highly scalable, so it can open new perspectives in the area of Boolean biological modeling, providing a way to handle models several orders bigger than those existing in the present.

Finally, the techniques discussed in this work are currently being generalized in order to allow efficient hardware implementations of membrane computing models [38], such as cell-like P systems [39–41], bacterial P systems [42] and spiking neural P systems [43–49], used for processing images [50], controlling mobile robots [51], robot path-planning [52,53], image processing [54] and modeling complex systems [55].

5. Materials and Methods

This section discusses the hardware implementation. In order to perform the automated translation of reaction systems to synchronous circuits a compiler `RStoVerilog` was written that allows to generate hardware description of the circuit using Verilog language. The syntax of the input file for `RStoVerilog` is the same as for `brsim/WEBRSIM` reaction systems simulator [56,57] and the description of the command-line options can be found in [36]. As a result it produces a Verilog description of the circuit, a Verilog test bench testing the circuit with the supplied input and the graph describing the corresponding Moore machine in GraphViz format [58]. The program can be found at [36] and can be freely used.

The compiler performed the following steps:

1. Parse the input file.
2. Identify input and output symbols.
3. Duplicate input symbols that are at the same time output symbols.
4. Transform the reaction system into a Boolean circuit in DNF.
5. Write Verilog output (module and the test bench).
6. Optionally, construct Moore automaton of the obtained system.

Step 1 was performed using standard compiling techniques. Now in order to apply the algorithm described in Section 3 one needs to identify input and output symbols. Since the initial concept of reaction systems does not possess this information, we used the following algorithm to automatically identify them. So during Step 2, the program identified symbols that never appeared in the products of a reaction as input symbols, while the others were identified as output symbols. This behavior can be overridden by special annotations in the source file that give the explicit list of input and output symbols (the remaining ones are considered as internal states). Since the above algorithm may lead to symbols that are at the same time input and output symbols, we introduced an additional Step 3, that, for each of these symbols (x), created a new input symbol x_{in} and copied all rules involving x in the reactants or inhibitors lists replacing x by x_{in} for all possible combinations of occurrences. We remark that this step can lead to an exponential blow-up of the number of reactions, so it is strongly suggested to manually identify input and output symbols.

Then Step 4 was performed according to the algorithm described in Section 3. Step 5 is straightforward, as a sequential Boolean (switching) function/circuit in DNF can be directly translated to Verilog. As a result, two files were obtained: a synthesizable (in FPGA) Verilog module containing the code that simulates each step of the reaction system and a (non-synthesizable) test bench that contains the timed update of the input symbols using the provided context.

Finally, the compiler can optionally construct the Moore automaton of the obtained circuit (if "-g" command-line switch is provided). This automaton is iteratively constructed by varying all the inputs starting from the initial state given by the first context. By adding the command-line switch "-ga" all possible initial states are constructed. The result was yielded using the GraphViz format [58]. Due to the combinatorial explosion, it was necessary to construct the automaton (the corresponding algorithm used standard enumeration techniques—so it was exponential), graph generation was limited to systems having less than 32 species. Another reason for this restriction was that bigger graphs were very difficult to analyze visually.

The Verilog test bench obtained by `RStoVerilog` can be directly simulated in software using a Verilog compiler and simulator. We used Icarus Verilog 10.1 compiler and simulator [59] and the corresponding result is quite competitive with respect to the other existing simulators. In order to make a real-life use case test, we used the reaction system translation of the model of ErbB receptor signal transduction in human mammary epithelial cells [24], which was performed in [22]. The corresponding reaction system model had 6720 reactions involving 246 entities. Comparing the running time for the context sequence of length 1000 also taken from [22] we obtained that the average time of running `brsim` was 4.2 s, of `HERESY` in CPU mode was 10.1 s and the average running time of our Verilog test bench was 7.98 s.

The next performed step was to run the obtained circuit in hardware. We have used a Diligent Basis 3 FPGA board, which features a Xilinx Artix 7 architecture. Since the result of our translation consists only of the circuit simulating each step of the reaction system (the test bench cannot be synthesized in hardware), we had to manually add the circuits allowing to enter the (input) context sequence as well as to save the output result. We used several test reaction systems (including those from Examples 2 and 3) as well as different input/output circuits. For small-size examples (up to 16 inputs and 16 outputs) we used switches and leds present on the board as input and output, allowing to verify the correctness of the simulation. In order to speed-up the computation we also considered input coming from the serial port at 115,200 bits/s. The tests have shown the correctness of the simulation.

Finally, the last tests were performed using an autonomous execution of the system without output and using distributed read-only memory data storage for the input (or context-less models). Under this setup the speed of 100 Mhz (corresponding to the system clock) was achieved. This means that a reaction system model can be simulated at a speed of 10^8 steps per second. Applied to the ErbB model this gives a speed-up of 2.5×10^5 with respect to the GPU-based simulator from [22].

Our tests show a low usage of FPGA resources (look-up tables (LUTs) and slices). For example, the ErbB model uses only 186 LUTs and 55 Slices, which corresponds to 0.89% and 0.67%, respectively, of available resources on Basis 3 board. Due to the simple architecture of the system these numbers suggest that a basis 3 board can handle reactions systems having up to 20 K reactants and 500 K reactions. Using a larger board, like VC707 based on Xilinx Virtex-7 architecture, it is possible to increase the size of the simulated system by 15 times.

From the speed point of view, our architecture allows to perform one computation step during one clock tick, so it runs at FPGA main clock speed, which can range from 50 Mhz to 400 Mhz. However, one needs to add the time needed for the input/output procedures as they are usually much slower (except for the input/output which are preloaded in distributed or block RAM). Depending on the size of the context it might be convenient to load it initially into the RAM, or to acquire the input data at each step, e.g., using serial port or peripheral component interconnect express (PCIe) connections.

We would like to remark that the input/output circuits are not system-specific and can be reused for different simulations. However, at the present state they need to be integrated manually in the final hardware design. It would be interesting to complete `RStoVerilog` compiler with a tool allowing to automate this process.

Author Contributions: Conceptualization, S.V.; investigation, Z.S., S.V., I.P. and G.Z.; software, Z.S. and S.V., writing—original draft preparation S.V., I.P. and G.Z.

Funding: Zeyi Shang, Sergey Verlan and Gexiang Zhang are partially supported by the National Natural Science Foundation of China, under Grant 61672437 and 61702428, the Sichuan Science and Technology Program, under Grant 2018GZ0185, 2018GZ0086 and 2018GZ0095, and the New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province China, under Grant 2018GZDZX0043. Ion Petre is partially supported by the Romanian National Authority for Scientific Research and Innovation, through the POC grant P_37_257.

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

References

1. Ehrenfeucht, A.; Rozenberg, G. Reaction systems. *Fundam. Inform.* **2007**, *75*, 263–280.
2. Ehrenfeucht, A.; Petre, I.; Rozenberg, G. Reaction systems: A model of computation inspired by the functioning of the living cell. In *The Role of Theory in Computer Science*; Konstantinidis, S., Moreira, N., Reis, R., Shallit, J., Eds.; World Scientific: Singapore, 2016; pp. 1–32. [CrossRef]
3. Brijder, R.; Ehrenfeucht, A.; Rozenberg, G. A note on causalities in reaction systems. *Electron. Commun. Easst* **2010**, *30*, 1–10. [CrossRef]
4. Bottoni, P.; Labella, A.; Rozenberg, G. Reaction systems with influence on environment. *J. Membr. Comput.* **2019**, *1*, 3–9. [CrossRef]

5. Ehrenfeucht, A.; Kleijn, J.; Koutny, M.; Rozenberg, G. Minimal reaction systems. In *Transactions on Computational Systems Biology XIV*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 102–122. [CrossRef]
6. Salomaa, A. Minimal and almost minimal reaction systems. *Nat. Comput.* **2013**, *12*, 369–376. [CrossRef]
7. Ehrenfeucht, A.; Main, M.; Rozenberg, G. Functions defined by reaction systems. *Int. J. Found. Comput. Sci.* **2011**, *22*, 167–178. [CrossRef]
8. Salomaa, A. Functions and sequences generated by reaction systems. *Theor. Comput. Sci.* **2012**, *466*, 87–96. [CrossRef]
9. Ehrenfeucht, A.; Rozenberg, G. Introducing time in reaction systems. *Theor. Comput. Sci.* **2009**, *410*, 310–322. [CrossRef]
10. Brijder, R.; Ehrenfeucht, A.; Rozenberg, G. Reaction systems with duration. In *Computation, Cooperation, and Life*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 191–202. [CrossRef]
11. Salomaa, A. Applications of the Chinese remainder theorem to reaction systems with duration. *Theor. Comput. Sci.* **2015**, *598*, 15–22. [CrossRef]
12. Ehrenfeucht, A.; Main, M.; Rozenberg, G. Combinatorics of life and death for reaction systems. *Int. J. Found. Comput. Sci.* **2010**, *21*, 345–356. [CrossRef]
13. Ehrenfeucht, A.; Rozenberg, G. Events and modules in reaction systems. *Theor. Comput. Sci.* **2007**, *376*, 3–16. [CrossRef]
14. Salomaa, A. Functional constructions between reaction systems and propositional logic. *Int. J. Found. Comput. Sci.* **2013**, *24*, 147–159. [CrossRef]
15. Genova, D.; Hoogeboom, H.J.; Jonoska, N. A graph isomorphism condition and equivalence of reaction systems. *Theor. Comput. Sci.* **2017**, *701*, 109–119. [CrossRef]
16. Kleijn, J.; Koutny, M.; Mikulski, Ł.; Rozenberg, G. Reaction systems, transition systems, and equivalences. In *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovičon the Occasion of His 60th Birthday*; Böckenhauer, H.J., Komm, D., Unger, W., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 63–84. [CrossRef]
17. Meski, A.; Penczek, W.; Rozenberg, G. Model checking temporal properties of reaction systems. *Inf. Sci.* **2015**, *313*, 24–42. [CrossRef]
18. Azimi, S.; Gratie, C.; Ivanov, S.; Manzoni, L.; Petre, I.; Porreca, A.E. Complexity of model checking for reaction systems. *Theor. Comput. Sci.* **2016**, *623*, 103–113. [CrossRef]
19. Azimi, S.; Gratie, C.; Ivanov, S.; Petre, I. Dependency graphs and mass conservation in reaction systems. *Theor. Comput. Sci.* **2015**, *598*, 23–39. [CrossRef]
20. Azimi, S.; Iancu, B.; Petre, I. Reaction system models for the heat shock response. *Fundam. Inform.* **2014**, *131*, 299–312.
21. Azimi, S.; Panchal, C.; Czeizler, E.; Petre, I. Reaction systems models for the self-assembly of intermediate filaments. *Ann. Univ. Buchar.* **2015**, *LXII*, 9–24.
22. Nobile, M.S.; Porreca, A.E.; Spolaor, S.; Manzoni, L.; Cazzaniga, P.; Mauri, G.; Besozzi, D. Efficient simulation of reaction systems on graphics processing units. *Fundam. Inform.* **2017**, *154*, 307–321. [CrossRef]
23. Azimi, S.; Panchal, C.; Mizera, A.; Petre, I. Multi-stability, limit cycles, and period-doubling bifurcation with reaction systems. *Int. J. Found. Comput. Sci.* **2017**, *28*, 1007–1020. [CrossRef]
24. Helikar, T.; Kochi, N.; Kowal, B.; Dimri, M.; Naramura, M.; Raja, S.M.; Band, V.; Band, H.; Rogers, J.A. A comprehensive, multi-scale dynamical model of ErbB receptor signal transduction in human mammary epithelial cells. *PLoS ONE* **2013**, *8*, e61757. [CrossRef]
25. Schroeder, R.L.; Stevens, C.L.; Sridhar, J. Small Molecule Tyrosine Kinase Inhibitors of ErbB2/HER2/ Neu in the Treatment of Aggressive Breast Cancer. *Molecules* **2014**, *19*, 15196–15212. [CrossRef]
26. Tatsuta, T.; Sato, S.; Sato, T.; Sugawara, S.; Suzuki, T.; Hara, A.; Hosono, M. Sialic Acid-Binding Lectin from Bullfrog Eggs Exhibits an Anti-Tumor Effect Against Breast Cancer Cells Including Triple-Negative Phenotype Cells. *Molecules* **2018**, *23*, 2714. [CrossRef]
27. Rosin, D.P.; Rontani, D.; Gauthier, D.J.; Schöll, E. Experiments on autonomous Boolean networks. *Chaos Interdiscip. J. Nonlinear Sci.* **2013**, *23*, 025102. [CrossRef]
28. Miskov-Zivanov, N.; Bresticker, A.; Krishnaswamy, D.; Venkatakrishnan, S.; Marculescu, D.; Faeder, J.R. Emulation of biological networks in reconfigurable hardware. In Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine (BCB'11), Chicago, IL, USA, 1–3 August 2011; ACM: New York, NY, USA, 2011; pp. 536–540. [CrossRef]

29. Purandare, M.; Polig, R.; Hagleitner, C. Accelerated analysis of Boolean gene regulatory networks. In Proceedings of the 27th Intnational Conference Field Programmable Logic and Applications (FPL), Ghent, Belgium, 4–8 September 2017; pp. 1–6. [CrossRef]
30. Crama, Y.; Hammer, P.L. *Boolean Functions: Theory, Algorithms, and Applications*; Encyclopedia of Mathematics and its Applications; Cambridge University Press: New York, NY, USA, 2011. [CrossRef]
31. Boole, G. *An Investigation of the Laws of Thought*; Cambridge University Press: New York, NY, USA, 1854. [CrossRef]
32. Shannon, C.E. A symbolic analysis of relay and switching circuits. *Trans. Am. Inst. Electr. Eng.* **1938**, *57*, 713–723. [CrossRef]
33. Kohavi, Z.; Jha, N.K. *Switching and Finite Automata Theory*, 3rd ed.; Cambridge University Press: New York, NY, USA, 2009. [CrossRef]
34. Mealy, G.H. A method for synthesizing sequential circuits. *Bell Syst. Tech. J.* **1955**, *34*, 1045–1079. [CrossRef]
35. Moore, M.E. Gedanken-experiments on sequential machines. *Autom. Stud.* **1956**, *23*, 129–153.
36. RsToVerilog Compiler. Available online: https://github.com/sverlan/RStoVerilog/ (accessed on 2 March 2019).
37. Vivado Design Suite. Available online: https://www.xilinx.com/products/design-tools/vivado.html (accessed on 2 March 2019).
38. Pan, L.; Păun, G.; Zhang, G. Foreword: Starting JMC. *J. Membr. Comput.* **2019**, *1*. [CrossRef]
39. Zhang, G.; Pérez-Jiménez, M.J.; Gheorghe, M. *Real-life Applications with Membrane Computing*; Springer: Berlin/Heidelberg, Germany, 2017. [CrossRef]
40. Román, G. Inference of bounded L systems with polymorphic P systems. *J. Membr. Comput.* **2019**. [CrossRef]
41. Mayne, R.; Phillips, N.; Adamatzky, A. Towards experimental P-systems using multivesicular liposomes. *J. Membr. Comput.* **2019**. [CrossRef]
42. Wang, X.; Zheng, P.; Ma, T.; Song, T. Small Universal Bacteria and Plasmid Computing Systems. *Molecules* **2018**, *23*, 1307. [CrossRef]
43. Song, T.; Zeng, X.; Zheng, P.; Jiang, M.; Rodríguez-Patón, A. A parallel workflow pattern modelling using spiking neural P systems with colored spikes. *IEEE Trans. Nanobiosci.* **2018**, *17*, 474–484. [CrossRef]
44. Cabarle, F.; de la Cruz, R.; Zhang, X.; Jiang, M.; Liu, X.; Zeng, X. On string languages generated by spiking neural P systems with structural plasticity. *IEEE Trans. Nanobiosci.* **2018**, *17*, 560–566. [CrossRef]
45. Song, T.; Rodríguez-Patón, A.; Zheng, P.; Zeng, X. Spiking neural P systems with colored spikes. *IEEE Trans. Cognit. Dev. Syst.* **2018**, *10*, 1106–1115. [CrossRef]
46. Zeng, X.; Pan, L.; Pérez-Jiménez, M.J. Small universal simple spiking neural P systems with weights. *Sci. China Inf. Sci.* **2014**, *57*, 1106–1115. [CrossRef]
47. Cabarle, F.; Adorna, H.; Jiang, M.; Zeng, X. Spiking neural P systems with scheduled synapses. *IEEE Trans. Nanobiosci.* **2017**, *16*, 792–801. [CrossRef] [PubMed]
48. Zhang, G.; Rong, H.; Neri, F.; Pérez-Jiménez, M. An optimization spiking neural P system for approximately solving combinatorial optimization problems. *Int. J. Neural Syst.* **2014**, *24*, 792–801. [CrossRef] [PubMed]
49. Wang, T.; Zhang, G.; Zhao, J.; He, Z.; Wang, J.; Pérez-Jiménez, M.J. Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Trans. Power Syst.* **2015**, *30*, 1182–1194. [CrossRef]
50. Díaz-Pernil, D.; Gutiérrez-Naranjo, M.A.; Peng, H. Membrane computing and image processing: A short survey. *J. Membr. Comput.* **2019**. [CrossRef]
51. Wang, X.; Zhang, G.; Neri, F.; Jiang, T.; Zhao, J.; Gheorghe, M.; Ipate, F.; Lefticaru, R. Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integr.-Comput.-Aided Eng.* **2016**, *23*, 15–30. [CrossRef]
52. Pérez-Hurtado, I.; Pérez-Jiménez, M.J.; Zhang, G.; Orellana-Martin, D. Simulation of rapidly-exploring random trees in membrane computing with P-lingua and automatic programming. *Int. J. Comput. Commun. Control* **2018**, *13*, 1007–1031. [CrossRef]
53. Wang, X.; Zhang, G.; Zhao, J.; Rong, H.; Ipate, F.; Lefticaru, R. A modified membrane-inspired algorithm based on particle swarm optimization for mobile robot path planning. *Int. J. Comput. Commun. Control* **2015**, *10*, 732–745. [CrossRef]
54. Yuan, J.; Guo, D.; Zhang, G.; Paul, P.; Zhu, M.; Yang, Q. A Resolution-Free Parallel Algorithm for Image Edge Detection within the Framework of Enzymatic Numerical P Systems. *Molecules* **2019**, *24*, 1235. [CrossRef]

55. Sánchez-Karhunen, E.; Valencia-Cabrera, L. Modelling complex market interactions using PDP systems. *J. Membr. Comput.* **2019**. [CrossRef]
56. Ivanov, S.; Rogojin, V.; Azimi, S.; Petre, I. WEBRSIM: A web-based reaction systems simulator. *Enjoying Natural Computing—Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday*; Lecture Notes in Computer Science; Graciani Díaz, C., Riscos-Núñez, A., Păun, G., Rozenberg, G., Salomaa, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2018; Volume 11270, pp. 170–181. [CrossRef]
57. GitHub—Scolobb/Brsim: A Basic Reaction System Simulator. Available online: https://github.com/scolobb/brsim (accessed on 2 March 2019).
58. Gansner, E.R.; North, S.C. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exp.* **2000**, *30*, 1203–1233. [CrossRef]
59. Icarus Verilog Compiler and Simulator. Available online: http://iverilog.icarus.com/ (accessed on 2 March 2019).

Sample Availability: Samples of the compounds are not available.