Noname manuscript No.
(will be inserted by the editor)

Chemical Reaction Network Designs for Asynchronous Logic Circuits

Luca Cardelli · Marta Kwiatkowska · Max Whitby

Received: date / Accepted: date

Abstract Chemical reaction networks (CRNs) are a versatile language for describing the dynamical behaviour of chemical kinetics, capable of modelling a variety of digital and analogue processes. While CRN designs for synchronous sequential logic circuits have been proposed and their implementation in DNA demonstrated, a physical realisation of these devices is difficult because of their reliance on a clock. Asynchronous sequential logic, on the other hand, does not require a clock, and instead relies on handshaking protocols to ensure the temporal ordering of different phases of the computation. This paper provides novel CRN designs for the construction of asynchronous logic, arithmetic and control flow elements based on a bi-molecular reaction motif with catalytic reactions and uniform reaction rates. We model and validate the designs for the deterministic and stochastic semantics using Microsoft's GEC tool and the probabilistic model checker PRISM, demonstrating their ability to emulate the function of asynchronous components under low molecular count.

This research is supported by a Royal Society Research Professorship and ERC AdG VERIWARE.

Luca Cardelli
Microsoft Research, Cambridge UK
Department of Computer science, University of Oxford
E-mail: luca@microsoft.com

Marta Kwiatkowska
Department of Computer science, University of Oxford
E-mail: marta.kwiatkowska@cs.ox.ac.uk

Max Whitby
Department of Computer science, University of Oxford
E-mail: max.whitby@cs.ox.ac.uk

1 Introduction

Chemical Reaction Networks (CRNs) are traditionally used to capture the behaviour of inorganic and organic chemical reactions in a well-mixed solution. Recently, a paradigm shift in the scientific community has seen the use of CRNs extend to that of a high-level programming language for molecular computing devices [13], where the fundamental computational process differs from conventional digital electronics in that it involves transformation of input chemicals into output via reaction rules, as opposed to processing discrete signals (voltage bands) interpreted as Boolean values. Several digital and analogue circuits [20,34] have been designed in CRNs and their computational power studied [35,10]. It has also been demonstrated in principle that any CRN can be physically realised in DNA [35, 5,12]. CRNs are therefore particularly attractive as a programming language for use in nanotechnology and biomedical applications, where it is difficult to integrate traditional electronics.

Chemical systems can store and process information in several ways. We focus on finite systems of molecules interacting in a well-mixed solution under mass-action kinetics and emulate Boolean circuits by encoding information through molecular concentrations reaching a particular threshold. The computation proceeds by transforming input species concentrations into outputs according to the reactions of a finite CRN. It is known that the computational power of CRNs is affected by the choice of the semantics, deterministic or stochastic. In particular, assuming a small probability of error, (finite) stochastic CRNs have been shown to be Turing universal [34]. The deterministic semantics interprets the reactions as a system of differential equations, which describe the evolution of the system as a vec-

tor of real-valued species concentrations over time [10]. The stochastic semantics, on the other hand, views the state of the system as a vector of (non-negative) integer molecular counts and state transitions as a reaction which has a non-zero probability of occurring [13]. The stochastic evolution of the system over time is obtained as a solution of the Chemical Master Equation (CME) [37]. It is well known that the deterministic semantics is not accurate for small populations. While the stochastic semantics is exact, it is infeasible for large molecular counts. One scalable alternative is the Linear Noise Approximation, which is a real-valued approximation of the CME [8]. The correctness of the behaviour of a circuit described by a finite CRN can be analysed by inspecting its stochastic and deterministic evolution over time. In addition, techniques such as model checking can be employed to analyse the temporal ordering of events.

While CRN designs for synchronous sequential logic circuits have been proposed, to mention [20,35,34], a physical realisation of these devices is challenging because of their reliance on a clock to synchronise events in order to ensure the correct temporal order of the phases of the computation. Clocks are difficult to make, since they arise from unique conditions of chemical concentrations and kinetic constants, and must control a large number of events. In electronics, an alternative circuit design technology is asynchronous sequential logic [36,23], which instead of a clock relies on handshaking protocols to synchronise events. Asynchronous circuits are widely used for low-power microprocessor designs, e.g., by ARM, though require a larger circuit area. The key component is the Muller C-element, which is used to synchronise multiple independent processes in a manner insensitive to the delays on wires and individual components. To ensure Turing completeness of asynchronous circuits, we also require an isochronous fork in addition to the Muller C-element. An isochronous fork is a component which produces a fan-out of signals that reach the target at virtually the same time. This assumption is difficult to achieve in conventional electronics, because of the need to make the wires the same length, but is straightforward in chemical kinetics because of the well-mixed assumption.

This paper provides novel CRN designs for the construction of an asynchronous computing device based on a bi-molecular reaction motif inspired by the Approximate Majority network [2,7]. The motif employs catalytic reactions to achieve bistable switching of molecular concentrations, which emulates high and low voltage signals in digital electronics. All components are produced with simple reactions and uniform reaction rates, where we assume a well-mixed solution under mass action kinetics, and are independent of a universal clock. Moreover, any design provided in this paper could in principle be realised as a DNA strand displacement device [5]. We work with the dual-rail design methodology and employ a variant of the diagrammatic language of [6] to represent the designs at the high level. Starting from the Muller C-element, we design the main components of a complete asynchronous computing device in terms of CRNs in a principled way, including logic gates, control flow and basic arithmetic, as well as more complex structures such as queues. We validate the designs by exploring their time evolution for all possible combinations of inputs using Microsoft's Visual GEC tool[1], with the latter also approximated using an experimental implementation of the Linear Noise Approximation (LNA) of [8] provided by Visual GEC that offers better scalability. We use the LNA to highlight a flaw with a key design component. Further, we demonstrate the correct behaviour of the circuits against temporal logic specifications with the probabilistic model checker PRISM[2] [17]. Our designs constitute the first feasible implementation of asynchronous computational components as CRNs, and are relevant for a multitude of applications in synthetic biology and biosensing.

This paper is an extended version of the conference paper [9].

2 Related Work

The computational power of Chemical Reaction Networks, viewed as a programming language for engineering biochemical systems, has been studied by a number of authors, to mention [13,10]. There are a number of ways in which chemical systems can encode and process information. This includes simulating Boolean circuits, where information is encoded in binary form using high and low concentrations similarly to this paper, e.g. [20, 35,34], as well as geometric arrangements, for example self-assembly [31] and molecular walkers [14] not considered here. Researchers have also investigated the power of CRNs to model distributed algorithms [2].

Regarding synchronous logic circuits, much of the work to date considered abstract CRN schemes. One exception is [15], where a system of actual chemical reactions is given, together with a precise molecular implementation for gates complete with a thermodynamic analysis of how the system would evolve, though only for simple gate designs. In [13] we see the construction and composition of simple logic gates based upon cat-

[1] `http://lepton.research.microsoft.com/webgec/` [27], both for the deterministic and stochastic semantics

[2] `www.prismmodelchecker.org`

alytic reactions, but they do not mention control flow or systematic component design in a dual rail setting. In [32] the authors propose CRNs for an inverter, an incrementer, a decrementer and a copier; their designs are based on two rate constants, "fast" and "slow", and thus are not rate-independent, in contrast to the designs presented here.

CRNs can also be viewed as computing functions over reals or Booleans. A single CRN computes a function over a finite domain, which is analogous to Boolean circuits in the sense that any given circuit computes only on inputs of a particular size [34]. An implementation of dual-rail logic gates that are rate-independent is given in [11]. In contrast, our designs are composable and capable of performing non-trivial computation.

Since the behaviour of CRNs is asynchronous, a fact evident through their equivalence with Petri net models [13], the main difficulty with programming them is the need to control the order of reactions. In [13] it is suggested that this "uncontrollability" can be handled by changing rate constants, an idea followed up in [24], where CRN designs for basic arithmetic are given based on two rate constants, "fast" and "slow". Our designs, on the other hand, exploit the asynchrony of the underlying CRN model and work with uniform rates.

Designs for the Muller C-element, though not the remaining components of an asynchronous device, have been constructed from genetic logic gates [25] and a genetic toggle switch [26], but we are not aware of any other nanoscale designs for asynchronous circuits. [35] shows that any CRN, including those presented in this paper, can theoretically be implemented as a DNA Strand Displacement device. These devices have been demonstrated in the lab [29,30,12], and thus provide an indication of experimental feasibility of our designs.

3 Preliminaries

3.1 Chemical Reaction Networks

A *Chemical Reaction Network* (CRN) $C = (\Lambda, R)$ is a pair of finite sets, where Λ is a set of chemical species and and R is a set of reactions. $|\Lambda|$ denotes the size of the set of species. Reactions in R describe how species interact. Formally, a *reaction* $\tau \in R$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^{|\Lambda|}$ is the vector of molecular counts of the reactants, $p_\tau \in \mathbb{N}^{|\Lambda|}$ is the vector of molecular counts of the products and $k_\tau \in \mathbb{R}_{>0}$ are the coefficient associated to the rate of the reaction. We assume ordering of species within vectors is alphabetical. Given a reaction $\tau_1 = ([1,1,0]^T, [0,0,2]^T, k_1)$, where $\cdot^T$ is the transpose of a vector, we often refer to it as $\tau_1 : A+B \xrightarrow{k_1} 2C$, where A, B and C are generic species.

In this paper we are only concerned with unimolecular reactions, i.e. those which have only one reactant, and bi-molecular, i.e. those with two reactants. The "reversible reaction" notation $A + B \leftrightharpoons 2C$ is a shorthand for the two reactions $A + B \xrightarrow{k_1} 2C$ and $2C \xrightarrow{k_2} A+B$, where k_1 and k_2 are not necessarily equal.

We assume that the system is *well stirred*, that is, the probability of the next reaction occurring between two molecules is independent of the location of those molecules, at fixed volume V and temperature; under these assumptions a *configuration* or *state* $x \in \mathbb{N}^{|\Lambda|}$ of the CRN is given by the number of molecules of each species. Given a configuration x we define $z = \frac{x}{N}$, where $N = V \cdot N_A$ is the volumetric factor, V is the volume and N_A Avogadro's number. We write $x(\lambda_i)$ for the number of molecules of λ_i in the configuration x and $z(\lambda_i) = \frac{x(\lambda_i)}{N}$ to denote the concentration of λ_i in the same configuration.

We will sometimes distinguish between CRNs with different initial configurations, and to this end define a *chemical reaction system* (CRS) as a tuple $S = (\Lambda, R, x_0)$ where (Λ, R) is a CRN and $x_0 \in \mathbb{N}^{|\Lambda|}$ represents its initial configuration, and we sometimes use the terms CRN and CRS interchangeably.

Diagrammatic CRN Notation To better visualize a CRN $C = (\Lambda, R)$ as a circuit, we employ a directed multi-edge graph (Λ, E) based upon a fragment of the diagrammatic notation for influence graphs [6]. Λ, the nodes, represent the species of the CRN C and edges E are derived from reactions R as follows. A reaction is represented as a directed multi-edge with sets of species as source and target. Each edge is either a pointed arrow (↑) or a rounded arrow (⥉), with the source represented by the flat edge and the target represented by the arrow head. A reaction that produces a species as a product is connected to it by a directed edge.

All reactions within our diagrams are catalytic and are bi-molecular reactions of the form $X + Y \to X + Z$, meaning that Y is transformed to Z and X is a catalyst, that is, X influences the transformation of Y to Z. The edges ⥉ represent that a source species is catalytic to a target reaction.

Example 1 (CRN Diagrams Example) We illustrate the flexibility of the diagrammatic notation with three CRN examples. Figure 1a shows the CRN with the single reaction $C + A \to C + B$. The ball (⥉) indicates that C is a catalyst to the reaction $A \to B$ represented by the arrow (↑). A species can act as both a reactant or catalyst in the same reaction, and similarly for a

product. In Figure 1b we depict the CRN with two reactions $\{A + A \to A + B, B + B \to B + A\}$, in which both A and B catalyse themselves. Figure 1c depicts the CRN with reaction set $\{A+B \to B+B, C+B \to B+B\}$, with species B catalysing multiple reactions, which is represented by a multiheaded edge with multiple ❡s. This CRN transforms species A and C into species B.

Dual-Rail Representation We represent a Boolean circuit with inputs I and outputs O, denoted $B(I, O)$, as follows. Firstly, a Boolean variable $b = \{0, 1\}$ could be encoded in a single species X, where 0 would be encoded as $E[|X|] = 0$ and 1 as $E[|X|] \geq M$, where $E[|X|]$ denotes an expectation of the number of molecules of X and M is a molecular population threshold.

The CRN computes by transforming an input concentration into an output concentration, which reaches the appropriate level upon convergence. However, since absence of molecules cannot be measured, we employ *dual-rail methodology* and represent every Boolean variable with two species, denoted X_{hi}, X_{lo}. Just like we cannot represent both 0 and 1 on an electrical wire, we restrict our CRNs such that either $E[|X_{hi}|] \geq M$ or $E[|X_{lo}|] \geq M$, but not both, can be present when a CRN has stabilised and no further reactions occur. We consider a high concentration output as correct if $E[|X_{hi}|] > 0.8max(X_{hi}) - 1SD(X_{hi})$, where 0.8 is a threshold normalised between the values $[0, 1]$, $max()$ is a function that returns the maximum molecular concentration of a species within the CRN and $1SD$ computes 1 standard deviation from the mean concentration of $E[X_{hi}]$. $1SD$ returns 0 under the deterministic semantics. Similarly, we consider a low concentration as correct if $E[|X_{lo}|] < 0.2 * max(X_{lo}) + 1SD(X_{lo})$.

For simplicity, we apply the dual rail methodology only to the variables in the input and output sets I and O. The circuits may contain additional variables, which will be considered internal and assumed not to catalyse with any species outside of the CRN circuit. We will encode these with single species and use the naming convention of referring to these internal species as $\lambda, \lambda_1, \cdots, \lambda_i$. When composing two circuits $B_1(I_1, O_1)$ and $B_2(I_2, O_2)$ in series, we define their composition as a circuit $B(I_1, O_2)$, in which all variables in $O_1 \cup I_2$ have been made internal.

Example 2 (Dual-Rail CRN 'Motif') We introduce a simple two reaction CRN which forms a 'motif' common to all our CRN circuit diagrams. The CRN is given by the set of reactions $\{X_{hi} + Y_{lo} \to X_{hi} + Y_{hi}, X_{lo} + Y_{hi} \to X_{lo}+Y_{lo}\}$ shown in Figure 2a, where the input set I contains X_{lo} and X_{hi} and the output set contains Y_{lo}, Y_{hi}. In this CRN the input X_{lo} or X_{hi} influences the reaction $Y_{lo} \leftrightharpoons Y_{hi}$ to produce as output the same Boolean value. We also include a diagrammatic CRN showing the composition of two such motifs in series. Here the inputs are X_{hi}, X_{lo} and outputs Y_{hi}, Y_{lo}.

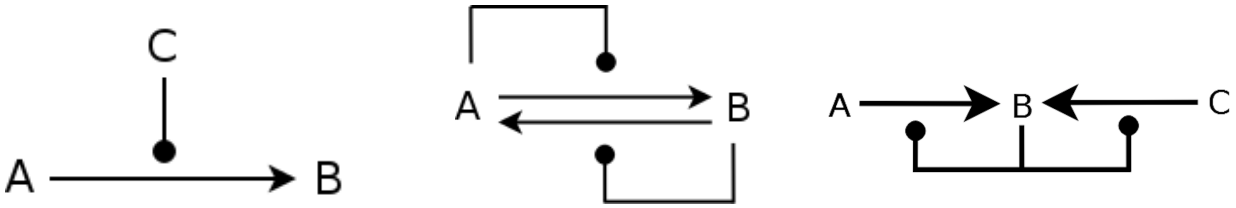
Example 3 (Dual-rail CRN with Internal (λ) Species) The CRN shown in Figure 3 represents a circuit $B(I, O)$ with internal species λ which does not belong to the set $I \cup O$. Therefore dual rail methodology is not used for λ as it is not catalytic to any species outside of this CRN circuit. I comprises X_{lo}, X_{hi} and O comprises Y_{hi}, Y_{lo}. The CRN is simplified to $\{X_{hi}+\lambda \to X_{hi}+Y_{hi}, X_{lo}+\lambda \to X_{lo} + Y_{lo}, \lambda + Y_{hi} \to \lambda + \lambda, \lambda + Y_{lo} \to \lambda + \lambda\}$. This CRN converts Y_{hi} and Y_{lo} to λ, assuming a non-zero initial concentration of molecules, unless there is either X_{lo} or X_{hi} present, in which case a deadlock occurs. We use the term conversion to mean the occurrence of a reaction where there are non-zero molecular counts of reactants present.

Deterministic Semantics Let $C = (\Lambda, R)$ be a CRN. The net change associated to a reaction $\tau \in R$ is defined by $\upsilon_\tau = p_\tau - r_\tau$. The deterministic semantics models the concentration of the species in Λ over time as a set of autonomous polynomial first order differential equations (ODEs):

$$\frac{\mathrm{d}\Phi(t)}{\mathrm{d}t} = F(\Phi(t)) = \sum_{\tau=(r_\tau,p_\tau,k_\tau)\in R} \upsilon_r \cdot (k_\tau \prod_{i=1}^{|\Lambda|} \Phi_i(t)^{r_{\tau,i}}). \quad (1)$$

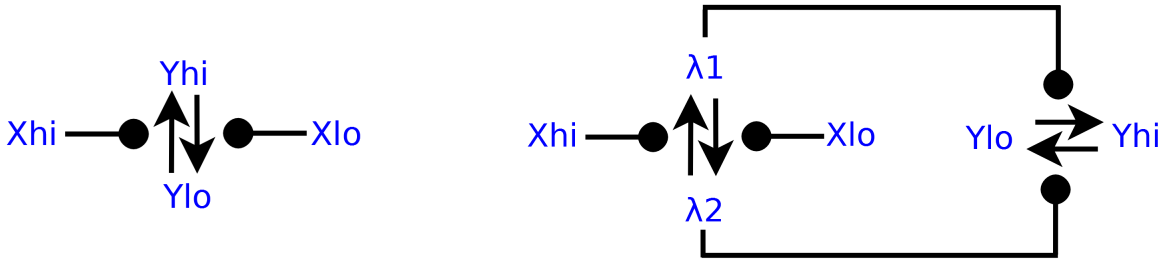
where function $\Phi : \mathbb{R}_{\geq 0} \to \mathbb{R}^\Lambda$ describes the behaviour of the system assuming a continuous state-space semantics, and therefore $\Phi(t) \in \mathbb{R}^{|\Lambda|}$ is the vector of the species concentrations at time t and F is simply the derivative of Φ with respect to time. Assuming $t_0 = 0$, the initial condition is $\Phi(0) = [x_0]$, where x_0 is the initial configuration (vector of concentrations of molecules) of the CRN. It is well known that the deterministic semantics may be imprecise for low molecular counts, but is accurate in the limit for high populations [37]. However, the deterministic semantics produces the same proportion of molecules, regardless of total concentration.

Stochastic Semantics The stochastic semantics is represented through a continuous-time Markov chain (CTMC), whose transient evolution can be given via the Chemical Master Equation (CME) [37]. Let $C = (\Lambda, R)$ be a CRN. The propensity rate α_τ of a reaction τ is a function of the current configuration



(a) (b) (c)

Fig. 1: Diagrammatic notation for CRNs. (a) CRN with the single reaction $C + A \to C + B$, in which species C catalyses the reaction $A \to B$. (b) CRN with reactions $\{A + A \to A + B, B + B \to B + A\}$, in which species A and B are both reactants, products and catalysts. (c) CRN with reactions $\{A + B \to B + B, C + B \to B + B\}$, which demonstrates that B can be catalytic to multiple reactions.



(a) (b)

Fig. 2: (a) The CRN 'motif', in which the input X_{lo} or X_{hi} influences the reaction $Y_{lo} \leftrightharpoons Y_{hi}$ to produce as output the same Boolean value. (b) CRN obtained by composing two 'motif's depicted in (a) in series. The inputs are now X_{hi}, X_{lo}, outputs Y_{hi}, Y_{lo}, and the remaining species have been made internal through renaming with λ.

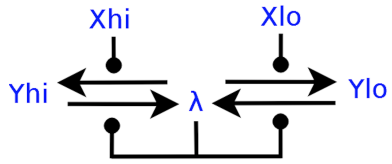


Fig. 3: Example CRN with internal λ species, which converts outputs Y_{hi} and Y_{lo} to λ, assuming a non-zero initial concentration of molecules, unless there is input X_{lo} or X_{hi} present.

of the system x such that $\alpha_\tau(x)dt$ is the probability that a reaction event occurs in the next infinitesimal interval dt. We assume mass action kinetics, therefore $\alpha_\tau(x) = k_\tau \frac{\prod_{i=1}^{|\Lambda|} r_{i,\tau}!}{N^{|r_\tau|-1}} \prod_{i=1}^{|\Lambda|} \binom{x(\lambda_i)}{r_{i,\tau}}$, where $r_{i,\tau}$ is the i-th component of the vector r_τ and $|r_\tau| = \sum_{i=1}^{|\Lambda|} r_{i,\tau}$ [1]. We define a time-homogeneous CTMC $(X^C(t), t \in \mathbb{R}_{\geq 0})$ with state space $Q \subseteq \mathbb{N}^{|\Lambda|}$ as follows. Given $x_0 \in Q$, where x_0 is the initial configuration of the system, then $P(X^C(0) = x_0) = 1$. The transition rate from state x_i to state x_j is defined as $r(x_i, x_j) = \sum_{\{\tau \in R | x_j = x_i + \upsilon_\tau\}} N\alpha_\tau(x_i)$. $X^C(t)$ describes the stochastic evolution of the molecular populations of each species in C at time t. For $x \in Q$, we define $P^{(t)}(x) = P(X(t) = x | X(0) = x_0)$, where x_0 is the initial configuration. The CME describes the time evolution of X as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(P^{(t)}(x)\right) = \sum_{\tau \in R} \{N\alpha_\tau(x - \upsilon_\tau)P^{(t)}(x - \upsilon_\tau) - N\alpha_\tau(x)P^{(t)}(x)\}. \tag{2}$$

The solution of the CME is computed through numerical simulation or discretisation techniques such as uniformization [16], and is generally feasible only for small populations. The CTMC is often represented as a $Q \times Q$ rates matrix, which can be viewed as a state transition graph and subjected to model checking against temporal logic properties [17].

Linear Noise Approximation The LNA approximates the CTMC as a continuous-state Gaussian process, given in the form of a set of ODEs that describe the time evolution of expectation and variance of the species. The error of approximation is dependent upon the volumetric factor N, the structure and the rates of the CRN. Given a CRN $\mathcal{C} = (\Lambda, \mathcal{R})$ with initial configuration $x_0 \in \mathbb{N}^{|\Lambda|}$ and in a system of volume size N, we define the stochastic process $Y = N \cdot \Phi + \sqrt{N} \cdot Z$, where

Φ is the deterministic process given in Eqn 1, and Z is a zero-mean Gaussian process (since we assume the initial condition is a fixed value), and with covariance $C[Z(t)]$ described by the solution of the following ODEs with initial condition $C[Z(0)] = 0$:

$$\frac{\mathrm{d}C[Z(t)]}{\mathrm{d}t} = F'(\Phi(t), C[Z(t)]) = J_F(\Phi(t))C[Z(t)] + C[Z(t)]J_F^T(\Phi(t)) + W(\Phi(t)) \quad (3)$$

where $J_F(\Phi(t))$ is the Jacobian of $F(\Phi(t))$, $J_F^T(\Phi(t))$ its transpose, and

$$W(\Phi(t)) = \sum_{\tau \in \mathcal{R}} \upsilon_\tau {\upsilon_\tau}^T k_\tau \prod_{S \in \Lambda} \Phi_S^{r_{S,\tau}}(t).$$

Expected value and covariance matrix of $Y(t)$ are completely characterized by $\Phi(t)$ and $C[Z(t)]$ since $E[Y(t)] = N\Phi(t)$ and $C[Y(t)] = \sqrt{N}C[Z(t)]\sqrt{N} = NC[Z(t)]$.

The LNA requires solving a number of ODEs quadratic in the number of species [8] and is therefore a scalable alternative to the solution of the CME. In contrast to the deterministic semantics, which considers average concentrations, the LNA does not compromise stochasticity.

Tool Support A number of software tools exist for examining the behaviour of CRNs. We employ Microsoft's Visual GEC, which provides a programming language, LBS, for designing and simulating a given CRN under the deterministic or stochastic semantics, including also the LNA approximation of the stochastic semantics. The tool is capable of producing plots of expected or average species concentrations over time. This functionality is used extensively within this paper to validate our circuit designs. In addition, Visual GEC exports models to the probabilistic model checker PRISM [17], which then enables verification of the induced continuous-time Markov chain against temporal logic properties. We use PRISM to verify the correctness of the temporal ordering of events occurring as the CRN circuit executes.

Example 4 (CRN Validation Under Different Semantics) We show the operation of our dual-rail CRN 'motif' given in Example 2 under the deterministic and stochastic semantics. The input configuration is $|X_{hi}| = 10$ molecules and output $|Y_{lo}| = 10$ molecules. Figure 4 demonstrates that, after 0.4 seconds, the CRN stabilises reaching the concentrations of $|Y_{hi}| = 10$ and $|Y_{lo}| = 0$ as desired. With regards to simulations provided, concentrations (given in nanomolars) are directly correlated to concentrations as we assume the volumetric factor N is fixed.

3.2 Asynchronous Hardware

Asynchronous computation is a model of computation that relies on transitions via local input signals rather than transitions via a global clock. Asynchronous computation [36], just like its synchronous counterpart, is Turing complete [21], meaning that any bounded-tape Turing machine can be implemented with an asynchronous circuit, providing that the implementation of that circuit has isochronous forks. An isochronous fork is the propagation of a signal from a single source to multiple receivers, with the important constraint that the signal must reach the receivers at precisely the same time. In classical digital circuitry this corresponds to the propagation of a signal down wires of exactly the same length from one component to another. In CRNs this could be seen as two species reaching a threshold N at precisely the same time.

Asynchronous circuits, which are designed based upon the theoretical principles of asynchronous computation, are widely used for low-power microprocessor designs, e.g., by ARM, and are increasing in popularity with the increase in distributed computing [23]. Asynchronous designs offer a number of advantages, the main one being correctness independent of timing, although they require a greater overhead in terms of silicon area.

We illustrate the principles of asynchronous circuit design by describing its key component called the Muller C-element and showing how it is used to construct a pipeline that propagates signals.

Muller C-Element The cornerstone of asynchronous computation is the Muller C-element. A Muller C-element has two Boolean inputs, X and Y, and one output Z. By definition these inputs can either be low or high (represented by 0 or 1). When both inputs are low the output is low. Similarly, when both inputs are high the output is high. The variation from a classical logic gate, however, is that if the inputs are high, or low, and one of them changes, it 'remembers' the last state. In other words, it retains the last 0 or 1 state. This is summarised in Figure 5a. An important property of the C-element is that it allows an observer to conclude on seeing output change from 0 to 1 that *both* inputs are now 1, and similarly for input change from 1 to 0.

The table specification indicates that asynchronous circuits exhibit *concurrency* and *causality*, and hence their specifications need to reflect these characteristics. A common way is as a timing diagram, seen in Figure 5d for the C-element, which represents a set of signals and their interactions over time. Each row of a timing

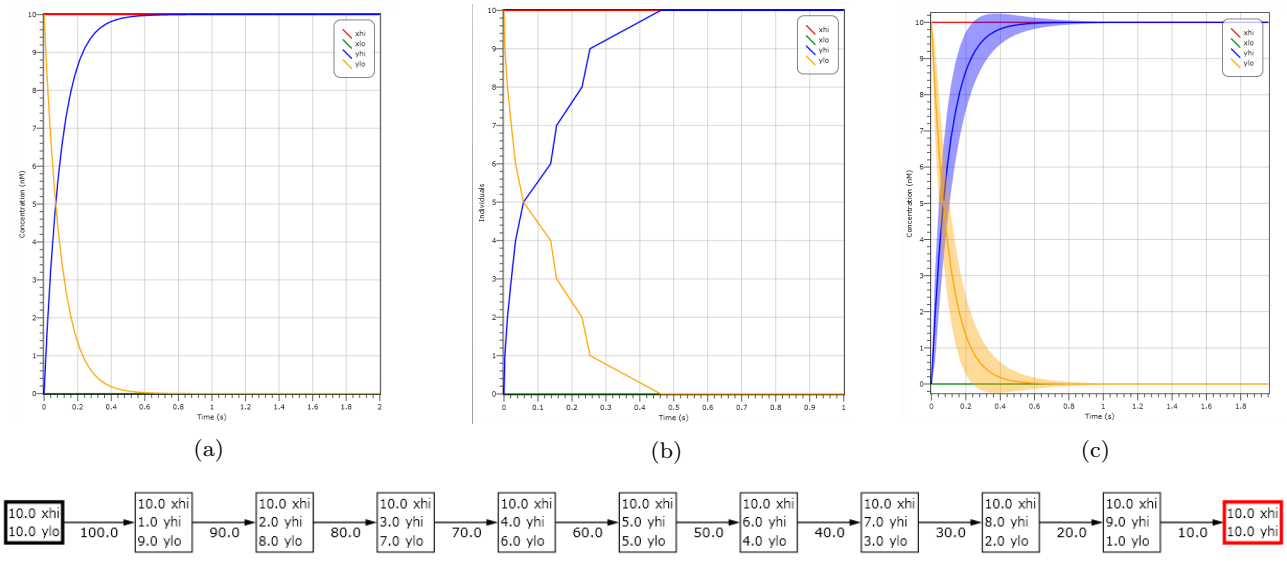


Fig. 4: Simulation of the motif given in Example 2 under different semantics. The input is initially X_{hi} and the output is Y_{lo}, both starting at a concentration of 10 molecules. In (a), (b) and (c) we respectively show the deterministic solution, stochastic simulation and LNA plot with variance, where Y_{lo} is seen in blue and Y_{hi} is seen in yellow. We can observe that in all cases Y_{hi} is present after 0.4 seconds. In (d) we show the state transitions of the induced CTMC that correspond to the output switching from Y_{lo} to Y_{hi}. The computation reaches the correct output state and stabilises, with no more transitions enabled.

diagram represents one signal and how it switches from low to high over time. If a signal displays a change before another signal on another line then this signal must precede the other. An arrow represents that one signal change triggers the change of another. In the C-element diagram, note that X and Y have to precede Z, both in the transition to 1 and down to zero. However, there is no causal dependency between X and Y.

Asynchronous diagrams, and in particular the C-element, are accurately described using Petri nets [36, p.86] or process algebras [38]. We present a (1-bounded) Petri net for the C-element in Figure 5c, in which transitions are interpreted as signal transitions and places and arcs capture the causal relations between the signal transitions. Following the usual convention, the Petri net is drawn in simpler form where most places have been omitted. We can observe that both tokens are needed in order to excite the transitions that cause the event Z_{hi}, which in turn will require the events X_{hi} and Y_{hi} to be triggered. The same is true for Z_{lo}.

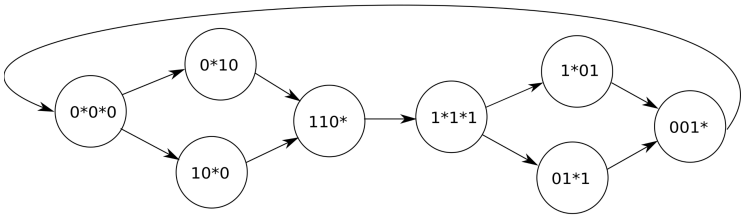
When considering circuit synthesis, we typically employ a state graph specification, which can be obtained from the Petri net representation [23] and is given for the C-element in Figure 5b. The values in each state correspond to the values of inputs X, Y and output Z, respectively. A * symbol indicates that the corresponding variable is excited by the outgoing transition (and will be changed in the following state). Observe how we can only transition to a state 1*1*1 from a state 110* requiring X as 1 and Y as 1. Because this is derived from a 1-bounded Petri net, we can assume that the transitions 0*10 → 110* and 10*0 → 110* do not conflict.

Muller C-Pipeline In order to replace the need for a global clock, asynchronous computation relies on 'local cooperation' in the form of handshaking protocols. These protocols exchange completion signals in order to establish when a computation has terminated. These handshaking protocols rely heavily on the C-element described above.

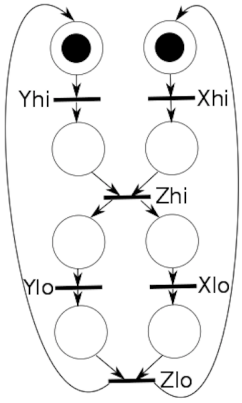
The Muller pipeline, shown in Figure 6, is constructed by the composition of Muller C-elements (depicted by the gate symbol labelled with C) and classical NOT-gates, which receive and send data to/from the environment (Left, Right in the figure). Its function is to propagate a high and low signal along the pipeline, emulating the 'wave' of high and low signals of a classical synchronous clock. Initially, all C-elements are set to a value of 0. The ith C-element $C[i]$ will propagate a 1 from its predecessor, $C[i-1]$, only if its successor, $C[i+1]$, is 0. Similarly, it will propagate a 0 from its predecessor only if its successor is 1. Eventually the first

Input X	Input Y	Output Z
0	0	0
1	0	unchanged
0	1	unchanged
1	1	1

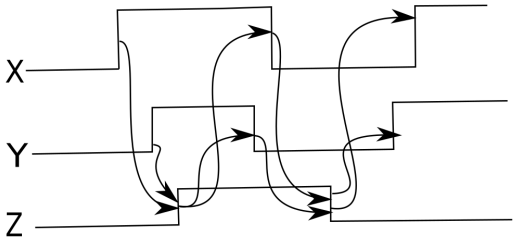
(a)



(b)



(c)



(d)

Fig. 5: Four specifications of a C-element with inputs X, Y and an output Z. (a) Conventional logic table, where 'unchanged' means that the state of the output is the last stable configuration of 1 or 0. (b) State graph, in which each state denotes a configuration and a transition is caused by the presence of a signal, where ∗ indicates that the signal is excited. (c) 1-bounded Petri net specifying the Muller C-element. (d) Timing diagram for the C-element.

request initialised on the left hand side of our pipeline is propagated to the final request on the right.

The protocol enacted upon this pipeline uses request and acknowledge rails that can be set to high or low. The Muller pipeline implements a basic four phase protocol, which is as follows. Firstly, the sender sends data and sets request to high, viewed in Figure 6 as the signal *Req* being high. The receiver then records this data and sets acknowledge to high (*Ack*). Then the sender responds by setting request to low (*Req*), and finally the receiver acknowledges this by setting acknowledgement to low (*Ack*). If at any point a handshake along the pipeline is slower than another, the pipeline will behave like a FIFO queue with data preserved. Herein lies the important purpose of the pipeline: it allows for the delay-insensitive transfer of information from one place to another. In combination with a latch we can create the propagation of information across latches using the pipeline as a control structure.

The construction of data storage and control structures such as queues and adders is similar to the Muller pipeline.

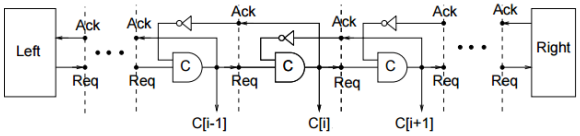


Fig. 6: Muller pipeline. Signals are propagated from left to right using request and acknowledge signals. The pipeline effectively queues data, only allowing a transition to occur when a further signal has been acknowledged.

Timing Properties of Asynchronous Circuits
Asynchronous circuits can be classified as being self-timed, speed independent or delay-insensitive, depending upon delay assumptions that are made. Assume a circuit is composed of gates and wires. A self-timed circuit operates correctly if both gates and wires experience measurable and fixed delays. A speed independent circuit operates correctly if gates exhibit some unknown time delay within gates but exhibits no time delay on wires. A delay-insensitive circuit operates correctly if there is both unknown delay within gates but also unknown delay within wires. The set of delay insensitive circuits is small, essentially those built from the Muller C-element and NOT gates, and so a broader class of quasi-delay insensitive circuits are identified. Quasi-delay insensitive circuits, which can be composed

of purely C-elements, NOT-gates and forks, are Turing complete [21]. They are not possible to achieve without an isochronous fork.

4 Asynchronous Circuit Designs as Chemical Reaction Networks

In this section we present our dual-rail designs for an asynchronous computing device in CRNs. The key component is the Muller C-element, whose design is inspired by the well known *Approximate Majority* (AM) CRN [2]. We begin by providing a detailed justification for our C-element design, and then describe the remaining simple components, including latches, logic gates and control flow. Finally, we present complex circuits such as the pipeline, queue and adder.

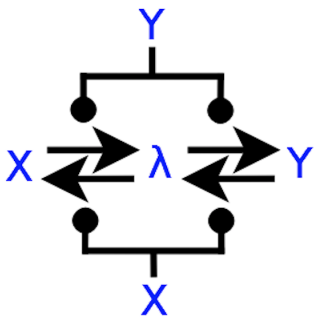
To justify the designs, we demonstrate that each component we design exhibits correct behaviour. Considering as an example the C-element, this amounts to working with an informal specification of the C-element in terms of high/low signals as in Figure 5, and then showing that our (continuous) CRN empirically satisfies that specification according to appropriate thresholding for high/low signals, as normally done in electronics for transistor logic. To this end, we explore the time evolution of the components under the deterministic and stochastic semantics, including also LNA for scalability. We additionally employ probabilistic model checking with PRISM, where temporal logic is used to express the temporal ordering of events. We remark that, although we validate the components for all possible input configurations, this does not amount to full verification of the correctness of the designs. We discuss the challenges of achieving full verification in Section 6.

4.1 Muller C-element as a CRN

The C-element design is based on the AM CRN, which computes the majority of two finite populations by converting the minority population into the majority population, so that a single population emerges as output. It uses a third 'undecided' state of the population, from where catalysis can drive the individuals into either of the final states. Interestingly, since approximate majority cannot be exactly computed by a bi-molecular CRN with less than 4 reactions [22], below we present the bi-molecular AM CRN with exactly four reactions:

$$X + Y \to X + \lambda$$
$$Y + X \to Y + \lambda$$
$$X + \lambda \to X + X$$
$$Y + \lambda \to Y + Y$$

where X, Y are both the input and output species and λ is the aforementioned catalytic driver. The intuition behind this reaction network is that we have two competing initial populations of X and Y, both of which try to eliminate the other by transforming their counterpart into the intermediary λ. If λ then interacts with X, it transforms itself into X, else into Y. Presented below in Figure 7a is the same AM CRN in our diagrammatic notation.



(a)

Fig. 7: Two diagrammatic CRNs which are capable of computing Approximate Majority [2]. In (a) we present the original in which the inputs X, Y, dependent on which species has the majority, influence outputs X, Y. (b) shows a similar AM circuit, but now the input species are catalysed to arbitrary output species W, Z.

Deconstructed, if we consider the left-hand side of the diagram, the reaction $X \to \lambda$ is catalysed by the species Y and so yields the bi-molecular reaction $Y + X \to Y + \lambda$. Similarly, X catalyses the reaction $\lambda \to X$ in the other direction and yields the reaction $X + \lambda \to X + X$. On the right-hand side of the diagram, λ is again catalysed by X and Y to produce Y, yielding the other two reactions from the AM CRN. The CRN in Figure 7b, with inputs X and Y and outputs W, Z, is similar to the AM CRN, except that we produce new arbitrary outputs W, Z instead of producing greater quantities of X and Y.

Since we wish to apply the dual-rail methodology, we represent each signal as a pair of species and encode the value 1 (0) as a molecular population of at least M for some population threshold M (population 0). We thus present in Figure 8a a dual-rail CRN which computes approximate majority with four inputs $X_{hi}, X_{lo}, Y_{lo}, Y_{hi}$ and outputs Z_{hi}, Z_{lo}. Like before, our input catalyses an intermediary species λ, but this is split over two reactions. In addition, Z_{lo} and Z_{hi} catalyse with reactions $Z_{lo} \leftrightharpoons \lambda$ and $Z_{hi} \leftrightharpoons \lambda$. This has the effect that, if there are larger numbers of either

Z_{lo} or Z_{hi}, the network enlarges its majority by converting the other into λ. Two rounded arrows ($\twoheadrightarrow$) over a reaction ($\uparrow$) indicates that either species can act as a catalyst to that reaction. We demonstrate this AM effect in Figure 8b, where, given the initial configuration of inputs X_{lo}, Y_{lo} of 10 molecules and output Z_{hi}, we observe under deterministic semantics that, because $|X_{lo}|, |Y_{lo}| > |X_{hi}|, |Y_{hi}|$, an output of Z_{lo} where $|Z_{lo}| = 10$ molecules and $|Z_{hi}| = 0$ is produced after 0.5 seconds.

We now need to justify the correctness of the design against the C-element specification in Figure 8a. Given a starting configuration of input X, Y both 1, and any starting output, the C-element should eventually output 1. Similarly, if X, Y are both 0, on any initial output our final output should be 0. For any other configuration of X, Y, the output signal should remain the same. Thus, given an input X_{hi}, Y_{hi}, and crucially any starting output configuration Z, we wish to reach a state where Z_{hi} has at least M molecules where M is a population threshold, and Z_{lo} has 0 molecules. Similarly, for X_{lo}, Y_{lo} and any Z we should to see a presence of Z_{lo} after some time t. For all other configurations of the inputs we wish the output species to remain the same.

When validated against this informal specification with a starting configuration of X_{lo}, Y_{hi} at 10 molecules and Z_{hi} at 10 molecules, our CRN unfortunately fails, seen in Figure 9. More specifically, we observe that species Z_{hi} is at a concentration of 6 molecules and λ has a concentration of 4 molecules after 0.3 seconds. This simulation was produced using LNA, see Section 3.1, which outputs standard deviation of the mean concentrations of species, seen in the shaded regions. This output configuration is incorrect since Z_{hi} is below the required threshold, see Section 3.1, of 8 molecules to represent an output of $Z = 1$.

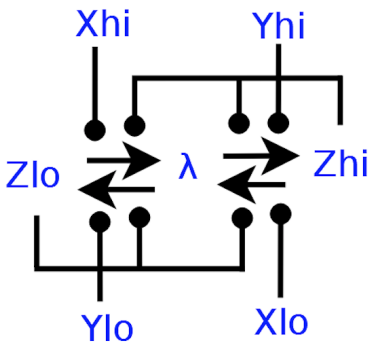
We present an amended CRN that resolves this issue in Figure 11a. This CRN is composed of two approximate majority circuits connected to each other, with the outcome of the first AM amplifying the outcome of the second. As we can see from the plot in Figure 9, we need to amplify the Z_{hi} species and suppress the λ species. The second AM corrects exactly this issue. Figure 11b is a simulation with inputs X_{lo}, Y_{hi} at 10 molecules and output Z_{hi} at 10 molecules. Here we can see that all three species are now at 10 molecules throughout the duration of the simulation.

To strengthen the validation of the final C-element design, we provide two further plots for selected initial configurations. We include in Figure 11c a deterministic plot with starting configuration of input X_{lo}, Y_{lo} at 10 molecules and output Z_{hi} at 10 molecules. After one second the system converges to output Z_{lo} at 10 molecules. Figure 11d shows an LNA simulation with starting configuration of input X_{hi}, Y_{hi} at 10 molecules and output Z_{lo} at 10 molecules. After one second we reach output Z_{hi} with probability ≈ 1. Both these simulations show that our output changes if both inputs change. From Figure 11b we observe that the output does not change if both inputs are different.

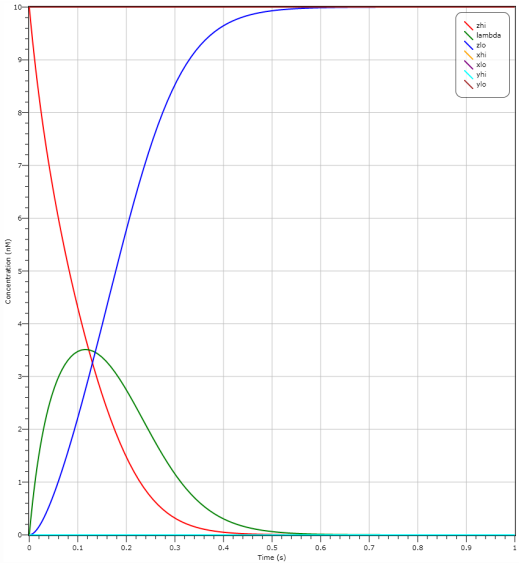
An issue to address is the use of dual-rail systems in which our chemical output is precisely a value of 0 molecules or K molecules, where K is the largest number of molecules achievable by a population in the system. In reality, a species may not reach its maximum population and, due to variance, we may have a situation, as seen in Figure 9, where one species has moderate probability of being higher than the output species we wish to present. Fortunately, we can use our approximate majority circuit to boost species. Figure 10 shows an example where species X_{hi} and Y_{hi} are weak, but the output is boosted by the C-element back to the maximum output of 10 molecules. The gates are also reusable: specifically, in Figure 11d we can observe the inherent reusability of the C-element because the output reacts to the change in input.

4.2 Latch Design

A latch is a device used in electronics to store a logical 0 or 1, which therefore needs to have at least two stable states that are cycled between. Latches are used in asynchronous computing both for storage and for synchronisation purposes. When an input of 1 is received a latch will ideally display an output of 1, and likewise for an input of 0. We present two latch designs in Figure 12, each intended to interface in a specific way when used within a larger system. The first simple latch, shown in Figure 12a(i), has input X_{lo}, X_{hi} and output species Y_{lo}, Y_{hi}, the intuition being that either X_{lo} catalyses Y_{hi} to Y_{lo} or X_{hi} catalyses Y_{lo} to Y_{hi}. There are also two additional reactions that catalyse Y_{lo} to itself and Y_{hi} to itself, creating a feedback loop. These additional reactions ensure that, if there is a drop in the molecular concentrations of input species, the latch retains its state. For some larger systems we may need the output state of a latch to be neither Y_{lo} nor Y_{hi}, to signify that no value is stored within the latch, known as a neutral state in electronics. A reset wire, to reset a latch to neutral state, is also commonly used in circuits. To this end, the latch in Figure 12a(ii) has an input R_{hi}, R_{lo} used to reset the latch to a central state λ, as well as the standard inputs X_{hi}, X_{lo} and outputs Y_{hi}, Y_{lo}. The advantage of this central state, λ, is that the latch can be in a state where neither Y_{hi} nor Y_{lo} are present, which is useful if these reactions are catalytic to any



(a)



(b)

Fig. 8: A dual-rail Approximate Majority CRN. (a) Circuit diagram. (b) Deterministic simulation of the CRN in (a). Given inputs X_{lo}, Y_{lo} at 10 molecules and output Z_{hi}, we observe that, because $|X_{lo}|, |Y_{lo}| > |X_{hi}|, |Y_{hi}|$, an output of Z_{lo} where $|Z_{lo}| = 10$ molecules (seen in blue) and $|Z_{hi}| = 0$ (seen in red) is expected to be produced after 0.5 seconds.

other component, in which case they will not be triggered directly. A comparison of the behaviour of the two latches is displayed in Figures 12b and 12c. With the same initial conditions X_{hi} and output Y_{lo} at 10 molecules, the latch in Figure 12a(ii) produces an output Y_{hi} at 10 molecules in 0.2 seconds. We contrast this with the latch in Figure 12c, which outputs Y_{hi} at 10 molecules in the slower time of 0.5 seconds.

4.3 AM as a Control Flow Element

An arbiter is used to decide an output signal based on which signal arrived first or if one signal is dominant over another. They are used in error correction where a signal may have degraded. Essentially, an arbiter computes the well known function $max(|X_1|, |X_2|)$ for two inputs. In terms of CRNs, this can be seen as either one species arriving before another or having higher molecular concentration. Since the AM circuit computes the $max(|X_1|, |X_2|)$ function, as one population is biased over another depending upon which has the majority, it therefore serves as an appropriate candidate for an arbiter. The proposed arbiter design, seen in Figure 13a, is the same as our AM CRN presented in Figure 8a, except that there are two inputs, X_{hi} and X_{lo}, and two outputs, Y_{hi} and Y_{lo}, instead of four inputs. This works as desired since the output Y_{hi}, Y_{lo} begins to be converted from λ as soon as either of X_{hi}, X_{lo} arrives, therefore automatically biasing whichever species is present first. The ability for approximate majority to reach a consensus means that this circuit can deal with stochastic fluctuations in input. Although, within electronic circuits, an arbiter outputs which signal arrived first, we assume that this information is revealed through the promotion of an output species linked to an input species.

In Figure 13b we demonstrate the operation of this arbiter by LNA simulation on inputs of X_{hi} at a concentration of 5 molecules and X_{lo} with a concentration of 0 molecules. After 0.4 seconds we see Y_{hi} (in red) at a concentration of 10 molecules, representing the majority.

4.4 Other Control Flow Circuits

Control flow is used to mediate or propagate the flow of information throughout the computing device. In digital circuitry forks and joins, both control flow elements, are naturally implemented using wires. Unfortunately, there is no natural fork or join within CRNs and consequently we present designs for them. The fork, shown Figure 14(a), is used to split signals. It has two input species X_{hi}, X_{lo} and four output species, $Y_{hi}(1), Y_{hi}(2)$ to represent the splitting of X_{hi} and $Y_{lo}(1), Y_{lo}(2)$ to represent the splitting of X_{lo}. The join, see Figure 14(b), joins two input signals to create one output sig-

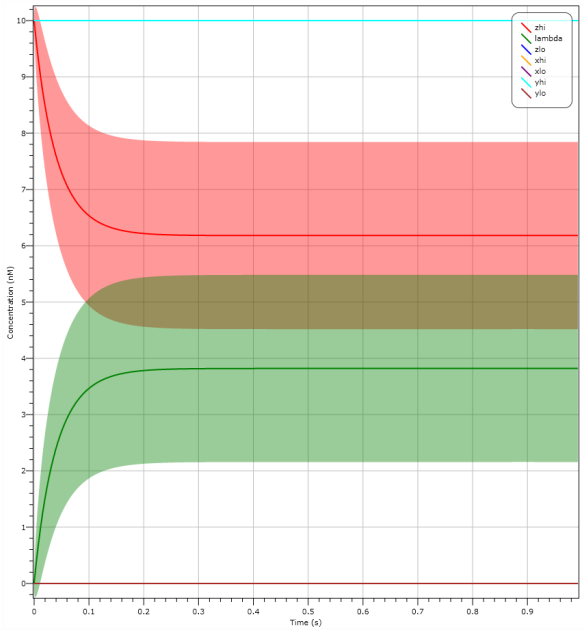


Fig. 9: LNA plot for the candidate CRN for the dual-rail Muller C-element design in Figure 8. With a starting configuration of inputs X_{lo}, Y_{hi} at a concentration of 10 molecules and output Z_{hi} at a concentration of 10 molecules, we can observe that after 0.3 seconds Z_{hi} (seen in red) is at a concentration of 6 molecules and λ is at a concentration of 4 molecules (seen in green). The shaded regions represent standard deviation. As we can see, with a non-zero probability we cannot distinguish the signals.

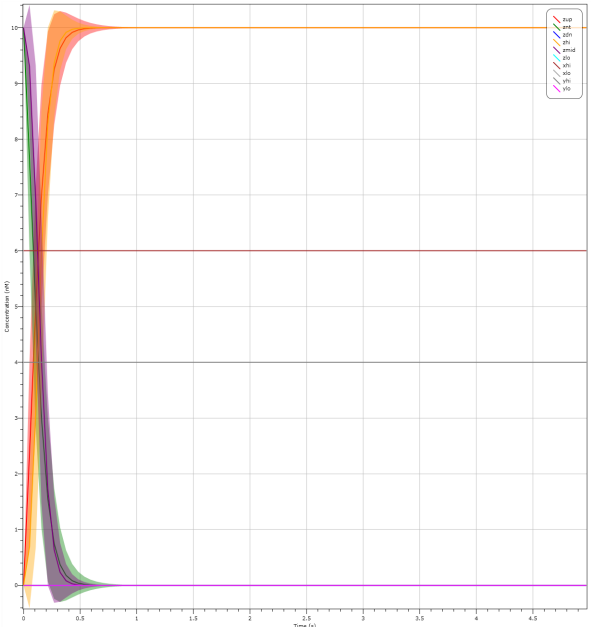


Fig. 10: LNA simulation demonstrating two weak input signals, X_{hi} and Y_{hi} at 6 and 4 molecules respectively, boosted by the C-element with an output Z_{hi} at 10 molecules.

nal. There are 4 inputs $X_{hi}(1), X_{hi}(2)$ with output Y_{hi} to represent the merging of the two input signals and $X_{lo}(1), X_{lo}(2)$ to merge to an output Y_{lo}.

4.5 Dual Rail Asynchronous Logic Gate Designs

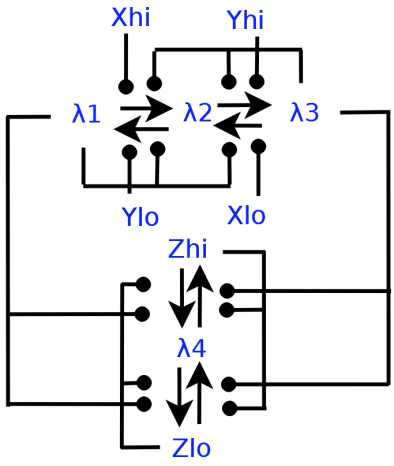
Although gate designs for Boolean operators have been proposed in CRNs [34], we present dual-rail implementations of logic gates in line with other designs proposed within this paper. In contrast to the gates in [34], our gates account for all inputs $X_{hi}, X_{lo}, Y_{hi}, Y_{lo}$ and outputs Z_{hi}, Z_{lo}. They are also reusable and respond to changes in input.

The simplest gate, NOT, in Figure 15a(i), inverts the inputs X_{hi}, X_{lo} to outputs Y_{lo}, Y_{hi}. The AND-gate, shown in Figure 15a(iii), has initial concentrations of λ_1, λ_2 as well as input concentrations. With a presence of species Y_{hi} we can catalyse λ_2 into the state λ_1, and with the species X_{hi} we can catalyse λ_1 to Z_{hi}; thus both species are needed for the gate to output the signal Z_{hi}. The state Z_{hi} catalyses a reaction between species Z_{lo} and λ_3, therefore showing that only one output signal can be present at any time. Conversely, with either X_{lo}, Y_{lo} we can convert λ_3 to Z_{lo}, which in turn can convert Z_{hi} back to λ_2 and λ_2 to λ_1. Using a similar trail of thought we can see how the other gates are devised, with OR (Figure 15a(ii)) having initial concentrations of λ_2, λ_3, NOR (Figure 15a(vi)) having initial concentrations of λ_2, λ_3 and NAND (Figure 15a(iv)) having initial concentrations of λ_1, λ_2. We provide an example, showing a deterministic simulation of the AND gate, seen in Figure 15b, in which inputs X_{hi}, Y_{hi} at 10 molecules and initial output Z_{lo} is converted to Z_{hi} after 0.8 seconds.

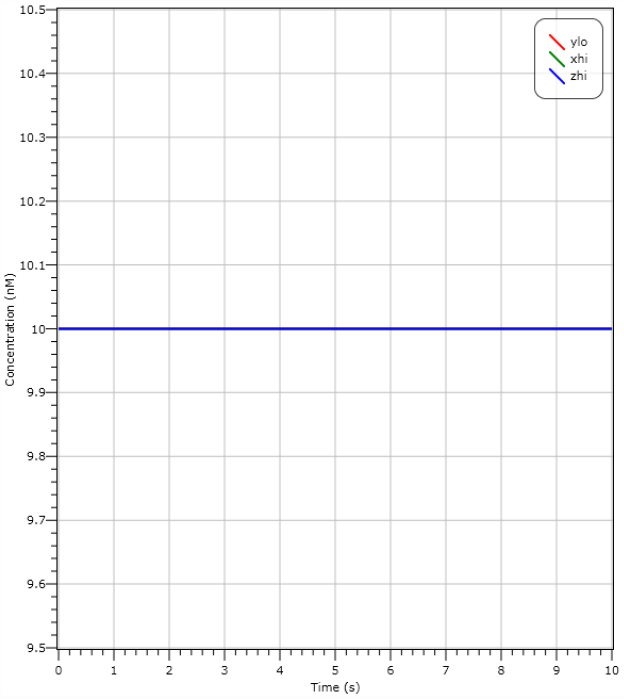
XOR is slightly different. XOR, traditionally a gate that requires a composition of many other logic gates, has to be constructed with all combinations of inputs considered. The XOR gate (Figure 15a(v)) has initial concentrations of $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. In Figure 16 we show an example validation of the XOR gate for all four input configurations using LNA.

4.6 Muller C-Pipeline

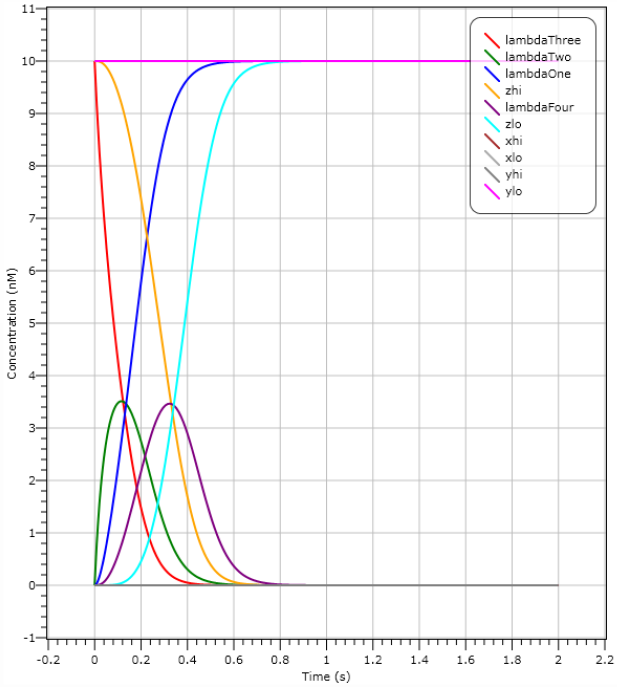
We construct a CRN to emulate the behaviour of the C-pipeline outlined in Section 3.2. The pipeline is a mechanism to relay handshakes between components, for example latches to store data. We construct the pipeline



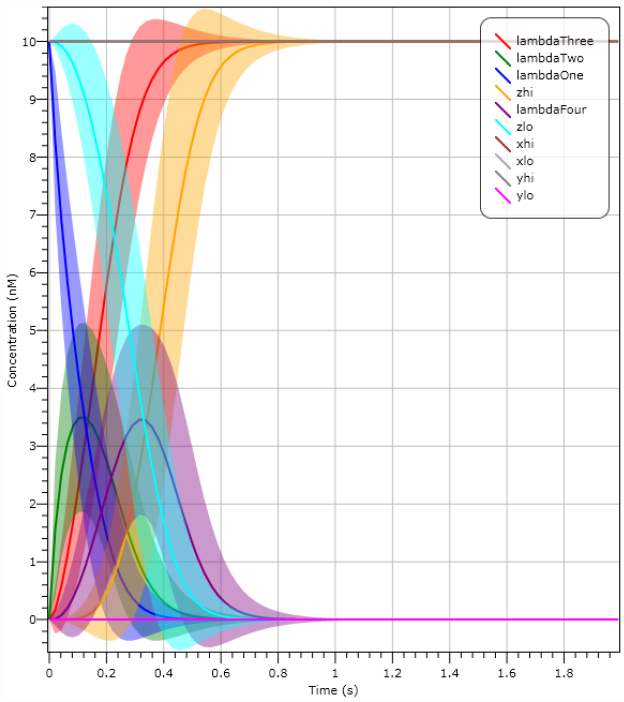
(a) Our final C-element design.



(b) Inputs X_{hi}, Y_{lo} maintain current output.

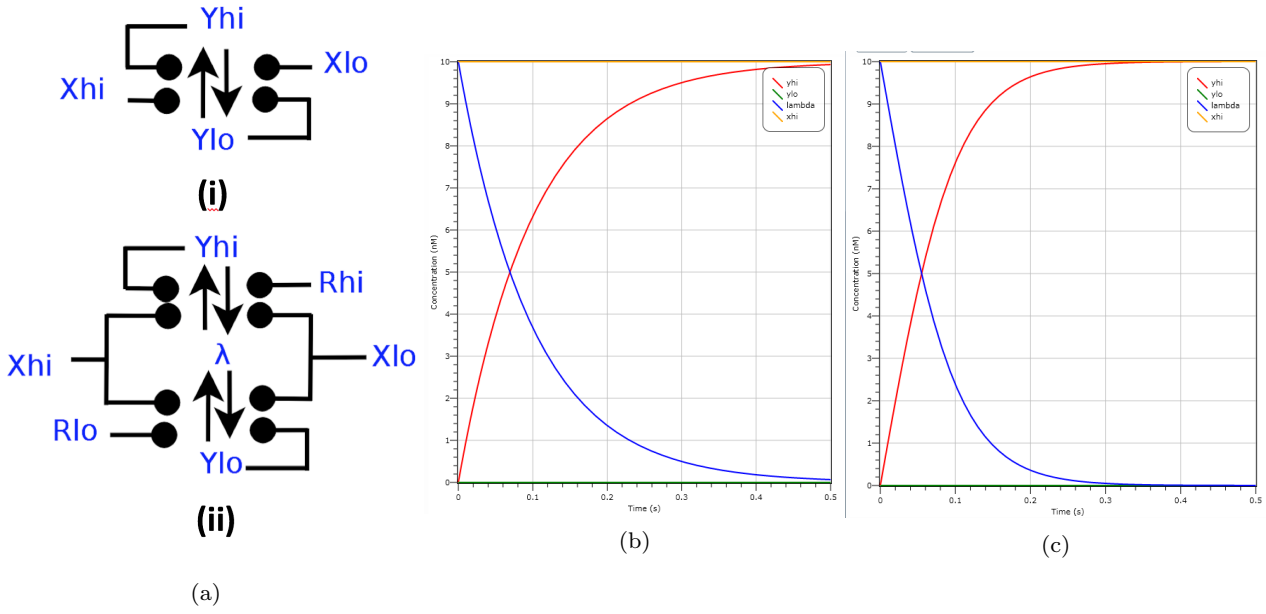


(c) Inputs X_{lo}, Y_{lo} lead to output Z_{lo}.



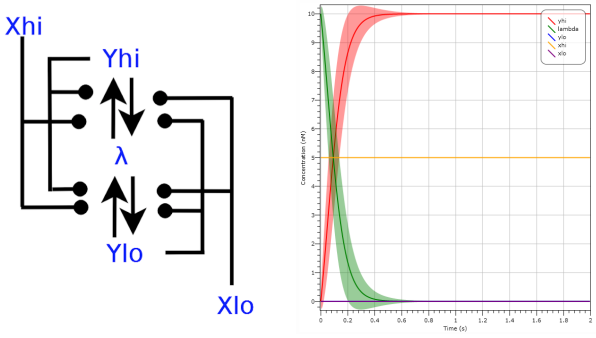
(d) Inputs X_{hi}, Y_{hi} lead to output Z_{hi}.

Fig. 11: Simulation of the final dual-rail Muller C-element design on selected input configurations. (a) Dual-rail AM circuit which is our final C-element design. (b) Deterministic simulation with inputs X_{lo}, Y_{hi} at 10 molecules and Z_{hi} at 10 molecules, which does not exhibit any change over time. (c) Deterministic simulation resulting in Z_{lo} (seen in light blue) based on the initial configuration X_{lo}, Y_{lo} at 10 molecules and initial output Z_{hi} at 10 molecules. (d) LNA simulation resulting in Z_{hi} (seen in yellow) based on the initial configuration of input X_{hi}, Y_{hi} at 10 molecules and initial output Z_{lo} at 10 molecules. Note that some plots are overlayed but are either set to 0 or 10 molecules.



(a) (b) (c)

Fig. 12: Two latch designs and their comparison. a(i) Simple latch with two feedback loops. a(ii) Latch with reset to a neutral state. (b) Deterministic plot of simple latch in a(i). (c) Deterministic plot of the latch in a(ii) which shows faster convergence.



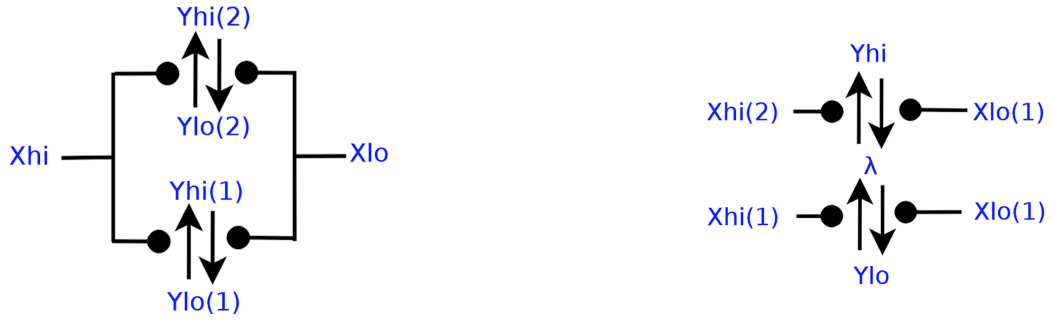
(a) (b)

Fig. 13: Arbiter circuit design and its simulation. (a) The arbiter CRN with inputs X_{hi} and X_{lo} and outputs Y_{hi} and Y_{lo}. The output reflects the input with the higher concentration of molecules (or which ever species appeared first). (b) LNA simulation of the arbiter, demonstrated with an input of X_{hi} at a concentration of 5 molecules and X_{lo} with a concentration of 0 molecules. After 0.4 seconds we see Y_{hi} (in red) at a concentration of 10 molecules, representing the majority input.

by placing three of our C-element CRNs, shown Figure 11a, in series. At each intermediate stage between the C-elements we add a fork. One path of the fork is negated and fed back into the previous C-element, and the other path is fed into the new C-element. The key interaction between the components of the C-pipeline is that no C-element can output a positive species or negative species without the previous displaying a positive or negative one.

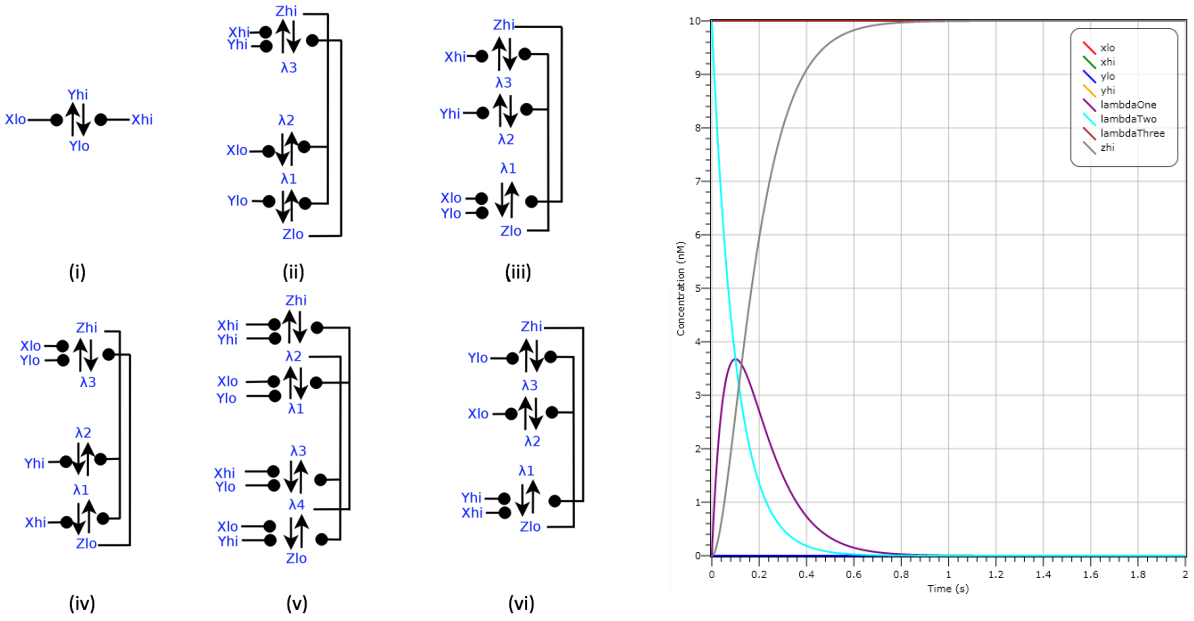
The inputs to the C-pipeline CRN are $Req_{hi}, Req_{lo}, Acc_{hi}$ and Acc_{lo}. The only output is C_{hi}, C_{lo}, which is the output species of the third C-element. However, for the sake of clarity, we also include four other species A_{hi}, A_{lo} corresponding to the output of the first C-element and B_{hi}, B_{lo} corresponding to the second. On an input of Req_{hi} at 10 molecules we would expect to see that A_{hi}, B_{hi} and C_{hi} are all at 10 molecules after a staggered amount of time. If we then changed the input to Req_{lo} we would expect to see A_{hi}, B_{hi}, C_{hi} diminish with A_{lo}, B_{lo}, C_{lo}, reaching 10 molecules to reflect the change in input. This is seen as a 'wave' through the pipeline propagating a high signal and then a low signal.

We design an experiment, see Figure 17, where we initialise the pipeline with the input species Req_{hi} at 10 molecules, and all C-elements are initialized with the intermediary species λ at 10 molecules such that no C-element yet outputs a species. From both the deterministic and LNA simulations of this we can observe how the species A_{hi}, B_{hi}, C_{hi} approach 10 molecules one after another, showing that indeed the value of Req_{hi} is being propagated along the pipeline. In order to show that our pipeline design is continuously reactive, we convert all of the species Req_{hi} to Req_{lo} via the introduction of a reaction $Req_{hi} \rightarrow Req_{lo}$. This effect oc-



(a) (b)

Fig. 14: CRNs for control flow components. (a) The fork used to split a signal into two. (b) The join used to merge signals.

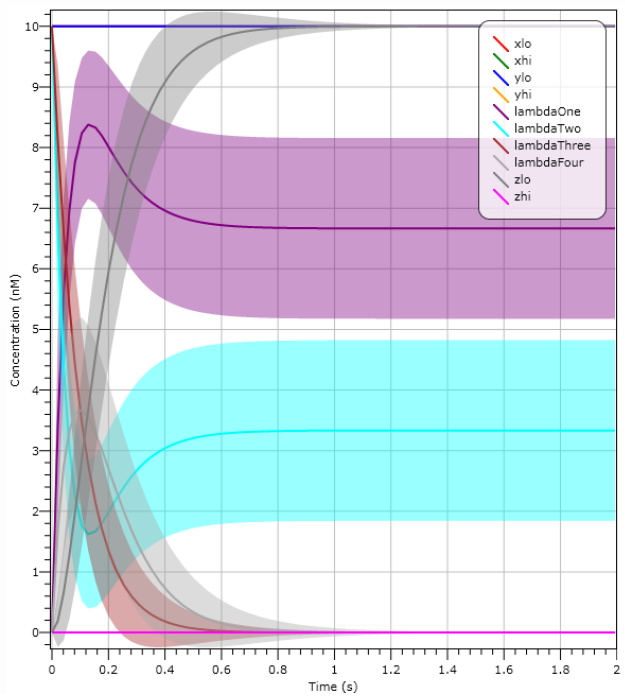


(a) (b)

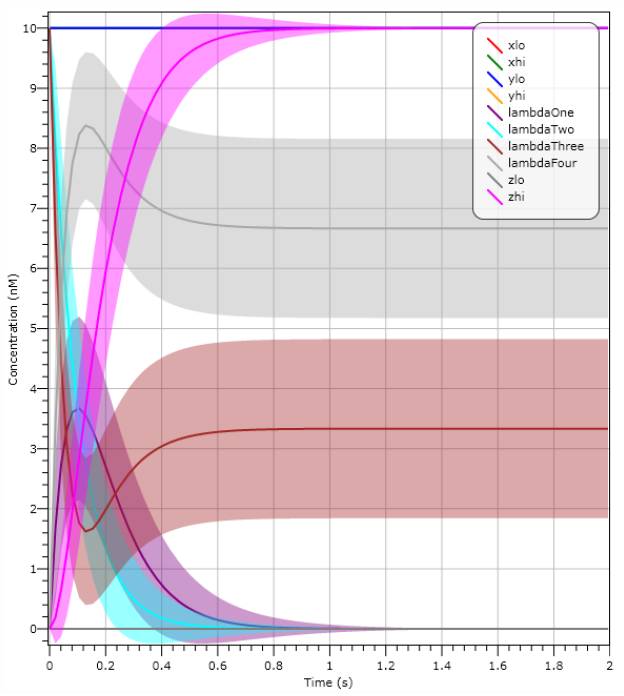
Fig. 15: (a) Dual-rail logic gate designs: we present a NOT (i), OR (ii), AND (iii), NAND (iv), XOR (v) and NOR (vi) over inputs X, Y and output Z. (b) Deterministic simulation for AND on inputs X_{hi}, Y_{hi} at 10 molecules and initial output Z_{lo}, which is converted to Z_{hi} after 0.8 seconds (seen in grey).

curs at around 1 second and we can observe that the pipeline responds by reducing the species A_{hi}, B_{hi}, C_{hi} to a molecular count of 0. We also observe (not shown on the simplified plot) that the species A_{lo}, B_{lo}, C_{lo} reach 10 molecules at the same time as A_{hi}, B_{hi}, C_{hi} reach 0 molecules (2 seconds). The LNA plot reveals that all output species are separated by a significant time difference such that no two species can be conflated.

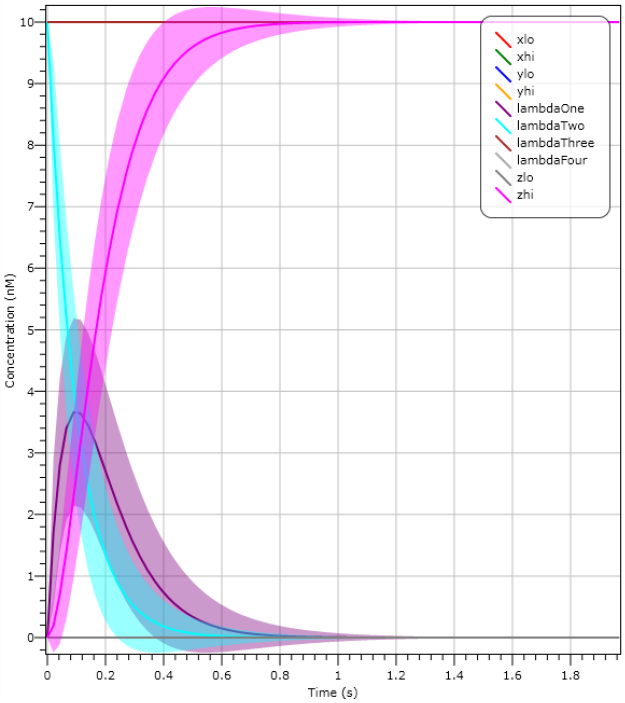
In addition to simulation experiments which plot expected concentrations of species over time, we also check temporal properties concerning the interactions between species and components. Using the PRISM model checker we interrogate the CTMC models of the pipeline with specific queries. We give some important examples of such queries in Table 1, which are verified by PRISM as being true with very high probability by checking 20 paths against the property. "Th" refers to the required population threshold which can be set by the user. PRISM also has the ability to track and plot concentrations of species over specific time intervals and number of samples. For example, for the C-pipeline we



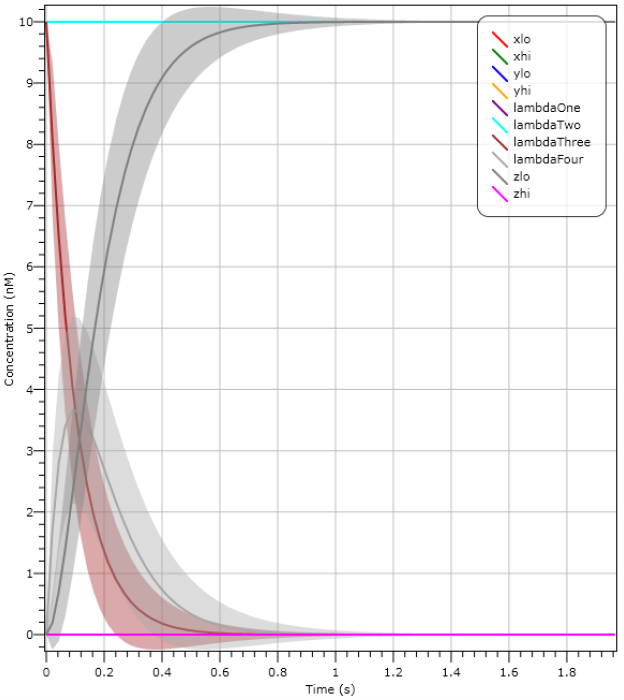
(a) Inputs X_{lo}, Y_{lo}



(b) Inputs X_{hi}, Y_{lo}

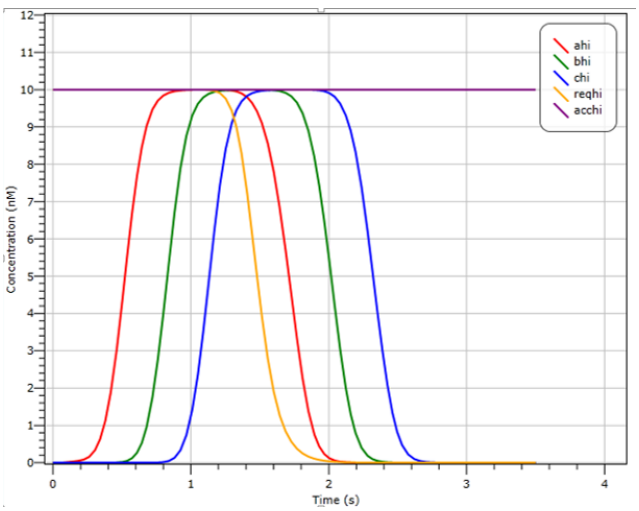


(c) Inputs X_{lo}, Y_{hi}

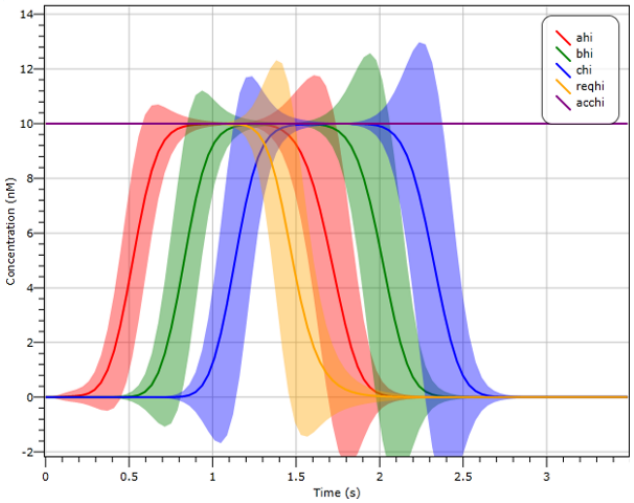


(d) Inputs X_{hi}, Y_{hi}

Fig. 16: XOR gate validation demonstrated using LNA for all input combinations and $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ having initial concentrations of 10 molecules. In (a), we demonstrate that XOR on an input configuration X_{lo}, Y_{lo} at 10 molecules produces an output Z_{lo} (seen in grey) at 10 molecules after 0.4 seconds. (d) shows a similar plot on input X_{hi}, Y_{hi}, which results in output Z_{lo} (seen in grey). In (b) and (c) we show that both X_{hi}, Y_{lo} and X_{lo}, Y_{hi} demonstrate the correct output of Z_{hi} (seen in pink).



(a) The Muller C-pipeline responding to the input species Req_{hi} being present and then transforming to the input species Req_{lo}.



(b) The same experiment calculated with the LNA. The standard deviation is shown with highlighted regions.

Fig. 17: Validation of the Muller C-pipeline. The input request signal, encoded by the species Req_{hi}, is propagated to the end of the pipeline (represented by the species A_{hi}, B_{hi}, C_{hi}); we then set the request signal to low. The pipeline then responds by the presence of A_{hi}, B_{hi} and C_{hi} diminishing to zero. In (b) we show that the variance is low, even for low molecular counts.

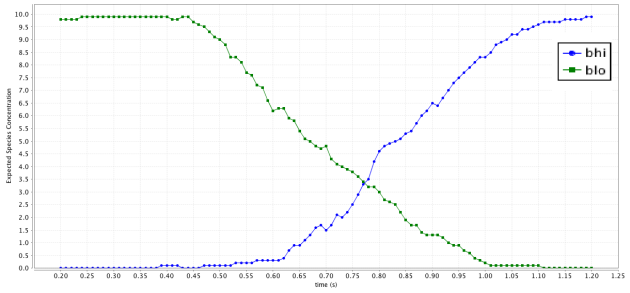


Fig. 18: The expected concentration of species B_{hi},B_{lo} and intermediary λ plotted over time for the C-pipeline under the stochastic semantics using reward structures within PRISM. Each data point is the expectation over 20 samples. We start at t=0.2. With an initial condition of Req_{hi}, we can see (in blue) that species B_{hi} is output from the second C-element at $t > 1.25$.

may wish to focus specifically on species concentrations of the second C-element whilst ignoring the others. We can isolate the species in question starting at a time > 0 and simulating only the species of the second C-element. We demonstrate this property on the pipeline with initial input species Req_{hi} at 10 molecules in Figure 18.

4.7 Queue

We have also designed and validated a queue, the schematic for which is shown in Figure 19, built by the addition of latches at each C-element block to the Muller pipeline. The queue is used in electronics to regulate and store the flow of information. The asynchronous queue uses the pipeline as a control mechanism to propagate signals between the latches. We use the latch with reset seen in 12 for this purpose. As the species Req_{hi} is propagated along the pipeline, it sends a signal to the queue to read and store the value in the next latch along. Each latch represents some computation that could be completed within each time interval.

For the latches of our queue we have input species Am_{hi} and Am_{lo} and output species $Ams_{hi}, Bms_{hi}, Cms_{hi}$ representing the output of each latch. Deterministic simulation the queue pipeline is shown in Figure 19. In this experiment we propagate a 1 (represented as Am_{hi} at 10 molecules) followed by 0 (Am_{lo} at 10 molecules). We can observe the species Ams_{hi}, Bms_{hi}, which represent the outputs of the first and second latches, noting an oscillatory pattern of cycling between 1 and 0.

Property in English	Initial Condition	PRISM Query	Prob. of Success
"Probability that the first C-element always outputs a high signal before the second wihin 3s"	$Req_{hi} > Th$	P =?[$(B_{hi} < Th) \mathcal{U}^{[0,3]}$ $(A_{hi} > Th)$]	0.97
"Probability that the C-element only changes when *both* inputs change within 10s"	1) $Req_{hi} Acc_{lo}$ 2) $Req_{lo} Acc_{hi}$ 3) $Req_{hi} Acc_{hi}$	P =?[true $\mathcal{U}^{[0,10]} Z_{hi} > Th$]	0.96
"Probability that the species A_{hi} reach their maximum population within 10s"	$A_{hi} < Th$	P =?[true $\mathcal{U}^{[0,10]} A_{hi} >= Th$]	1
"Probability that request signal is propagated to the end of the pipeline within 10s"	$Req_{hi} > Th$, $A_{hi} > Th$	P =?[$F^{[0,10]} C_{hi} > Th$]	1

Table 1: Temporal properties for the C-pipeline verified by the PRISM model checker. Each property was checked on 20 paths for the pipeline with inputs at 10 molecules.

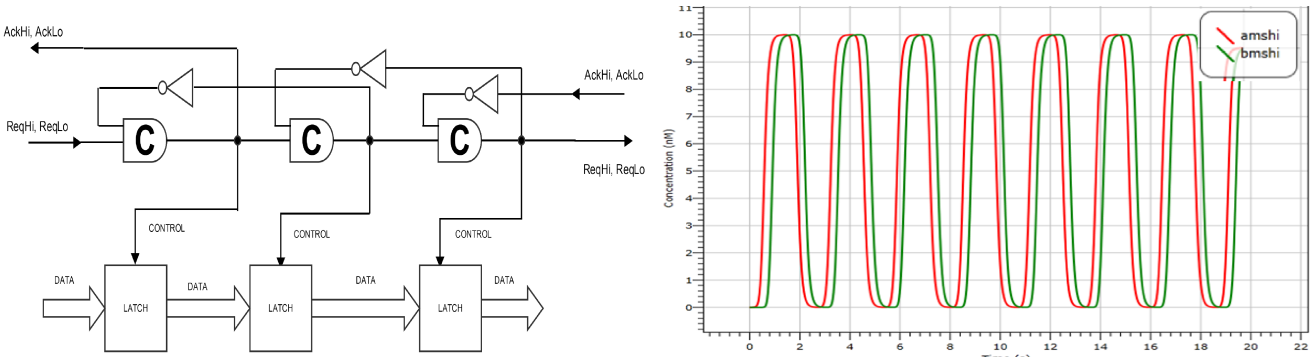


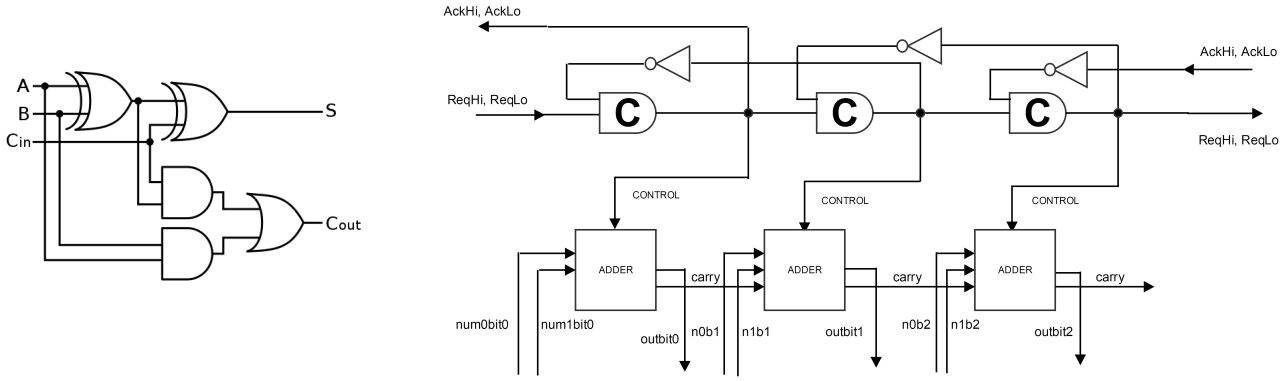
Fig. 19: Deterministic simulation of the queue pipeline. We propagate a value of 1 through the queue. The species Ams_{hi}, Bms_{hi} represent the outputs of the first and second latches. Note that through oscillatory patterns generated by the pipeline we can mimick properties of a synchronous system.

4.8 Adder

We have also designed a three bit ripple-carry adder, which operates in a similar fashion to the queue but instead of latches we compose adders in series, seen in Figure 20b. An adder, the circuit design for which is seen in Figure 20a, is a composition of two XOR gates, two OR gates and an AND gate. The adder produces two outputs, the value of the summation and the carry. Within the ripple-carry adder, the carry output of each adder is fed into the next adder, which outputs the sum and a carry. In this way, with three adders, we can add three two-bit numbers together.

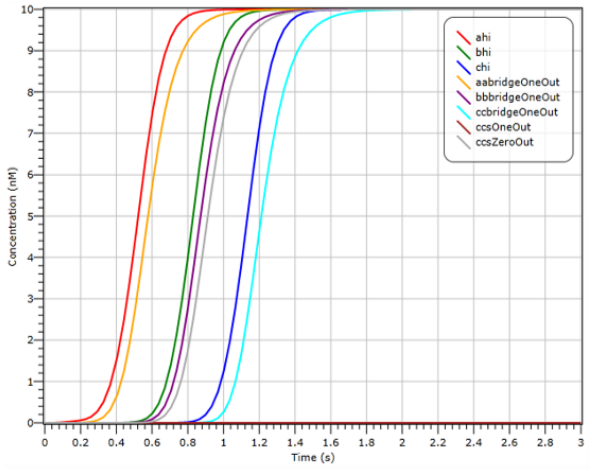
Our ripple-carry CRN has four input species per adder representing the two inputs, and two output species representing the output. In Figure 21 we show that the adder exhibits correct behaviour by producing the desired output species for a specific input, where each sum is calculated only in the next stage in the pipeline. If we view each C-element and adder as one stage in the pipeline, labelled A,B and C, then we can view the output species of each adder as a bridge to the next adder. The six output species, representing the carry-bit output, are denoted by $A_{abridgeOneOut}$, $B_{bbridgeOneOut}$, $C_{cbridgeOneOut}$ and $A_{abridgeZeroOut}$, $B_{bbridgeZeroOut}$, $C_{cbridgeZeroOut}$. In order to show correct operation the output of the adders (represented in this case by 10 molecules) should be interleaved with the control species of the C-pipeline (A_{hi},B_{hi},C_{hi}), allowing time for the carry species to catalyse with the input of the following adder.

Whilst a simulation provides the intuition behind the ripple-carry adder, using the PRISM model checker we can query the output of all three adders after 10 seconds to confirm if the correct output is present. We have four input species per adder, excluding the carry, which represent two numbers. We expect one species from each adder as output, representing the addition of two inputs, plus the carry. The third adder relies on the

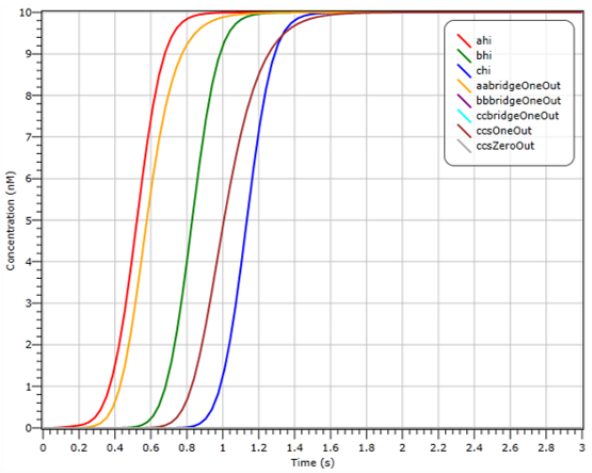


(a) (b)

Fig. 20: (a) Circuit diagram for a ripple-carry adder. (b) Three adders in series controlled by the C-pipeline. A carry-bit output from one adder is fed into the next as part of the input.



(a) Ripple-carry adder response to input values of 1: 10 2: 10 3: 10 to each adder.



(b) Ripple-carry adder response to input values of 1: 10 2: 00 3: 10 to each adder.

Fig. 21: Deterministic simulation of the ripple-carry adder circuit responding to various inputs. We plot the output species from each section of the pipeline used to coordinate the output from each adder. The carry-bit output from each adder is represented by $A_{abridgeOneOut}$, $B_{bbridgeOneOut}$ and $C_{cbridgeOneOut}$. The output of the C-element (A_{hi}, B_{hi}, C_{hi}) arrives strictly before the output from the adder. The logical input for (a) is 1 and 0 for the first adder, 1 and 0 for the second adder, and 1 and 0 for the third adder. In (b), we show the computation on different inputs, namely 1 and 0 for the first adder, 0 and 0 for the second adder, and 1 and 0 for the third adder. The crossover in the concentrations of output species of the C-element and the logical output of 1 (resulting from inputs 0, 1 and carry of 1) in the third adder (plots C_{hi} and $C_{csOneOut}$) indicates faster convergence but does not affect the results in further stages of the pipeline.

previous adder's carry being correct. We therefore only need to look at the desired output of each adder plus the carry of the final adder. We summarise this with the following example PRISM property:

P =?[true U <= 10(($A_{abridgeOneOut}$ > Th) and ($B_{bbridgeZeroOut}$ > Th) and ($C_{cbridgeOneOut}$ > Th) and ($C_{cbridgeCarryOneOut}$ > Th))]

With this example we can only satisfy these three outputs and the carry, by seeing each of their molecular concentrations rise above the threshold Th, based on a specific input configuration. This particular input are the species representing 0 and 1 for the first adder (for example $A_{aOneZeroIn}$ and $A_{aTwoOneIn}$), the species representing inputs 1,1 for the second adder and 1,1 for the third adder. 0 + 1 in the first adder

should give us an outcome of 1 carry 0 and so satisfies $A_{abridgeOneOut} > Th$. 1 + 1 plus the 0 carry from the first adder gives an output of 0 carry 1, and so satisfies $B_{bbridgeZeroOut} > Th$. An input of 1 + 1 plus 1 carry from the second adder means that our output should be 1 as well as the final carry should be 1, represented by $C_{cbridgeOneOut} > Th$ and $C_{cbridgeCarryOneOut} > Th$. Our adder satisfies this property based upon the inputs given and therefore shows correct operation for an adder.

5 Experimentation

All designs[3] presented in this paper have been validated using both Microsoft's Visual GEC tool [27] and the PRISM model checker [17], both for the deterministic and stochastic semantics of the CRNs. Visual GEC provides a programming language, LBS, for designing and simulating any given CRN. We systematically tested each component in isolation by simulating its behaviour against all input and output configurations. Next, we examined how a component might behave in a larger system, where it will be exposed to a change in input. To this end, we introduced new reactions to emulate a signal change. For instance, if we wished to change a carrier signal from high to low, we would introduce an additional reaction $X_{hi} \xrightarrow{k} X_{lo}$, which converts all of the signal X_{hi} into a signal X_{lo} while the component is operating.

Since deterministic semantics is not accurate for low molecular populations, we additionally explored stochastic semantics. Visual GEC exports models to the probabilistic model checker PRISM, which then enables verification of the induced continuous-time Markov chain against temporal logic properties. This allows one to check that the circuits ensure the correct temporal ordering of the events, for example, for the Muller pipeline seen in Figure 6, that the species in the first stage of the pipeline is present before the species in the second, i.e. with probability 1, and that the signal is eventually propagated to the end of the pipeline. PRISM implements numerical solution of the CME, which is exponential in the initial number of molecules and hence not scalable, and analysis based on stochastic simulation, which is time consuming. We thus additionally used an experimental implementation of the LNA within Visual GEC, based on [8]. As well as being capable of checking temporal logic properties [8,4], the LNA can plot the species concentration over time together with standard deviation, and is fast and reasonably accurate even for low molecule counts. Moreover, compared to the deterministic semantics, LNA provides important information about stochasticity that may affect the robustness of the circuits, and which can be explored further with CME, stochastic simulation, or verifying that the circuit converges with probability 1 to a single value.

[3] Available from `https://github.com/max1s/CRNcode`

6 Discussion

When modelling asynchronous circuits as a chemical system, the wires are chemical species from the output set of one gate component to the input set of another. We cannot bound the time for a gate to transform an input species to an output species. This excludes the class of self-timed circuits. Under deterministic semantics, we could guarantee an isochronous fork since two chemical species, either high or low, could theoretically reach the threshold M at precisely the same time given equal rates and initial concentrations, and therefore under deterministic semantics we have a Turing-complete method of computation. We cannot guarantee this under stochastic semantics [13]. This is because there is a non-zero probability that one species could reach M before the other. We thus conclude that our circuits, at worst, can be classified as speed independent. We can calculate approximately the delay on wires based upon rates and concentrations for each semantics.

Direct chemical implementations of CRNs have been theorised and realised, but involve complicated reaction mechanisms [33]. The most common substrate for chemical kinetics is DNA strand displacement (DSD), which involves the displacement of DNA strands in solution. These strands are labelled with the chemical species and, once the reaction has taken place, an outputting strand represents an output from the CRN that the strand displacement system is trying to emulate. DNA strand displacement has been shown to be a universal substrate for chemical kinetics, specifically for bi-molecular reactions used here [35]. Most importantly, the AM circuit seen in Figure 8b has been implemented as a strand displacement device [18]. However, a potential difficulty with this approach is scalability: as the number of components increases, the number of chemical species representing them also increases. Large numbers of chemical species result in large numbers of DSD complexes in solution, and consequently crosstalk needs to be accounted for. Fortunately, a recent experiment with the implementation of a square-root circuit in solution provided a new experimental ceiling on the number of species that can be used [29].

Another challenge is to provide formal verification of the correctness of the designs. We remark that, although we have validated the behaviour of the compo-

nents for all possible input configurations and verified that correct outputs are produced and in the correct order using simulation and simulation-based probabilistic model checking with PRISM, this does not amount to full verification. Asynchronous diagrams are represented using a variety of notations, see Figure 5, and correspond to certain classes of (safe) Petri nets [23] known as Signal Transition Graphs (STGs), representing the rise and fall of signals. Our designs are systems containing many molecules which exhibit stochastic behaviour. Firstly, one would need to show that our CRN designs meet the specification given as a Petri net, which would involve relating the two formally via a refinement relation, where one needs to relate structures with many molecules of a given a species to structures with at most one. This presents us with two major challenges: scalability and stochasticity. Scalability can be addressed using compositional verification, which has been developed for (non-probabilistic) process algebraic specifications of asynchronous circuits [38] (equivalent to STGs) and it would be interesting to see if they can be applied in this setting. However, no probabilistic extension of this approach is known. Another possibility is to capture stochasticity by employing stochastic Petri nets to model the designs as done for molecular walkers in [3], and then perform temporal logic verification using the tool Cosmos. Cosmos relies on an implementation of statistical model checking that exploits parallelism of the Petri net specifications and achieves greater scalability than PRISM.

7 Conclusion

We have proposed a novel design for an asynchronous computing device based on Chemical Reaction Networks. CRNs are inherently asynchronous, and thus particularly well suited to this computational paradigm. Our designs are based on a simple, bi-molecular reaction motif inspired by Approximate Majority [2,7], employ catalytic reactions and assume well-mixed solution and constant, uniform rates. Moreover, they do not rely on the universal clock which is difficult to realise. Since an arbitrary CRN can be physically realised using DNA strand displacement [35], as recently demonstrated experimentally in [12], the proposed designs are in principle implementable, and we have confirmed this in theory by modelling them in the two-domain setting [5] using Visual DSD [28,19]. Our designs are the first feasible implementation of an asynchronous computing device in chemical kinetics and are relevant for a multitude of applications in nanotechnology and synthetic biology. As future work we would like to investigate alternative experimental settings.

References

1. D. F. Anderson and T. G. Kurtz. Continuous time Markov chain models for chemical reaction networks. In *Design and analysis of biomolecular circuits*, pages 3–42. Springer, 2011.
2. D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.
3. B. Barbot and M. Kwiatkowska. On quantitative modelling and verification of DNA walker circuits using stochastic Petri nets. In R. Devillers and A. Valmari, editors, *Application and Theory of Petri Nets and Concurrency*, volume 9115 of *Lecture Notes in Computer Science*, pages 1–32. Springer International Publishing, 2015.
4. L. Bortolussi, L. Cardelli, M. Kwiatkowska, and L. Laurenti. Approximation of probabilistic reachability for chemical reaction networks using the linear noise approximation. In *Proc. 13th International Conference on Quantitative Evaluation of SysTems (QEST 2016)*, LNCS. Springer, 2016. To appear.
5. L. Cardelli. Two-domain DNA strand displacement. *Developments in Computational Models*, 26:47–61, 2010.
6. L. Cardelli. Morphisms of reaction networks that couple structure to function. *BMC systems biology*, 8(1):84, 2014.
7. L. Cardelli and A. Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific reports*, 2, 2012.
8. L. Cardelli, M. Kwiatkowska, and L. Laurenti. Stochastic analysis of chemical reaction networks using linear noise approximation. In *Computational Methods in Systems Biology*, pages 64–76. Springer, 2015.
9. L. Cardelli, M. Kwiatkowska, and M. Whitby. Chemical reaction network designs for asynchronous logic circuits. In *Proc. 22nd International Conference on DNA Computing and Molecular Programming (DNA22)*, LNCS. Springer, 2016. To appear.
10. H.-L. Chen, D. Doty, and D. Soloveichik. Deterministic function computation with chemical reaction networks. *Natural Computing*, 13(4):517–534, 2013.
11. H.-L. Chen, D. Doty, and D. Soloveichik. Rate-independent computation in continuous chemical reaction networks. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 313–326. ACM, 2014.
12. Y.-J. Chen, N. Dalchau, N. Srinivas, A. Phillips, L. Cardelli, D. Soloveichik, and G. Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8(10):755–762, 2013.
13. M. Cook, D. Soloveichik, E. Winfree, and J. Bruck. Programmability of chemical reaction networks. In *Algorithmic Bioprocesses*, pages 543–584. Springer, 2009.
14. F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. J. Turberfield. DNA walker circuits: Computational potential, design and verification. *Natural Computing*, 14(2):195–211, 2015.
15. A. P. de Silva and N. D. McClenaghan. Molecular-scale logic gates. *Chemistry–A European Journal*, 10(3):574–586, 2004.
16. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Formal methods for performance evaluation*, pages 220–270. Springer, 2007.
17. M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer aided verification*, pages 585–591. Springer, 2011.

18. M. R. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, page rsif20110800, 2012.
19. M. R. Lakin, S. Youssef, F. Polo, S. Emmott, and A. Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.
20. M. O. Magnasco. Chemical kinetics is Turing universal. *Phys. Rev. Lett.*, 78:1190–1193, Feb 1997.
21. R. Manohar and A. J. Martin. Quasi-delay insensitive circuits are Turing complete. In *ASYNC '96: Proceedings of the 2nd International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society, 1996.
22. G. B. Mertzios, S. E. Nikoletseas, C. L. Raptopoulos, and P. G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *International Colloquium on Automata, Languages, and Programming*, pages 871–882. Springer, 2014.
23. C. J. Myers. *Asynchronous circuit design*. John Wiley & Sons, 2004.
24. N. E. Napp and R. P. Adams. Message passing inference with chemical reaction networks. In *Advances in Neural Information Processing Systems*, pages 2247–2255, 2013.
25. N.-p. Nguyen, C. Myers, H. Kuwahara, C. Winstead, and J. Keener. Design and analysis of a robust genetic Muller C-element. *Journal of theoretical biology*, 264(2):174–187, 2010.
26. N.-P. D. Nguyen, H. Kuwahara, C. J. Myers, and J. P. Keener. The design of a genetic muller c-element. In *Asynchronous Circuits and Systems, 2007. ASYNC 2007. 13th IEEE International Symposium on*, pages 95–104. IEEE, 2007.
27. M. Pedersen and A. Phillips. Towards programming languages for genetic engineering of living cells. *Journal of the Royal Society Interface*, April 2009.
28. A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *J R Soc Interface*, 6 Suppl 4:S419–S436, Aug 2009.
29. L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.
30. L. Qian and E. Winfree. A simple DNA gate motif for synthesizing large-scale circuits. *Journal of the Royal Society Interface*, page rsif20100729, 2011.
31. P. W. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA sierpinski triangles. *PLoS biology*, 2(12):e424, 2004.
32. P. Senum and M. Riedel. Rate-independent constructs for chemical computation. *PloS one*, 6(6):e21414, 2011.
33. S. W. Shin. *Compiling and verifying DNA-based Chemical Reaction Network implementations*. PhD thesis, California Institute of Technology, 2011.
34. D. Soloveichik, M. Cook, E. Winfree, and J. Bruck. Computation with finite stochastic chemical reaction networks. *natural computing*, 7(4):615–633, 2008.
35. D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.
36. J. Spars and S. Furber. *Principles Asynchronous Circuit Design*. Springer, 2002.
37. N. G. Van Kampen. *Stochastic processes in physics and chemistry*, volume 1. Elsevier, 1992.
38. X. Wang and M. Kwiatkowska. On process-algebraic verification of asynchronous circuits. *Fundamenta Informaticae*, 80(1-3):283–310, 2007.