Chemical Reaction Network Designs for Asynchronous Logic Circuits*

Luca Cardelli[1,2], Marta Kwiatkowska[1], and Max Whitby[1]

[1] Department of Computer Science, University of Oxford
[2] Microsoft Research

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 uniform reaction rates. We model and validate the designs using Microsoft's GEC tool.

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 [10], where the fundamental computational process differs from conventional digital electronics in that it involves transformation of input chemicals into output via reaction rules. Several digital and analogue circuits [17,25] have been designed in CRNs and their computational power studied [26,7]. It has also been demonstrated in principle that any CRN can be physically realised in DNA [26,3,9]. CRNs are therefore particularly attractive as a programming language for use in nanotechnology and biomedical applications, where it is difficult to integrate traditional electronics.

While CRN designs for synchronous sequential logic circuits have been proposed, 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 [27], which instead of a clock relies on handshaking protocols to synchronise events. Asynchronous circuits are

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

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. 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 [1,5]. All components are produced with simple reactions and uniform reaction rates, and are independent of a universal clock. Moreover, any design provided in this paper could in principle be realised as a two-domain DNA strand displacement device [3].

We work with the dual-rail design methodology and employ a variant of the diagrammatic language of [4] 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. We illustrate the designs on selected components validated using Microsoft's Visual GEC tool[3], both for the deterministic and stochastic semantics, with the latter approximated using a prototype implementation of the Linear Noise Approximation of [6].

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.

2 Related Work

The computational power of CRNs, viewed as a programming language for engineering biochemical systems, has been studied by a number of authors, to mention [10,7]. Researchers have investigated their power to simulate Boolean circuits, molecular machines, or distributed algorithms [17,26,25,11,1].Assuming a small probability of error, CRNs have been shown to be Turing-universal [25]. Since the behaviour of CRNs is asynchronous, a fact evident through their equivalence with Petri net models [10], the main difficulty with programming them is the need to control the order of reactions. In [10] it is suggested that this "uncontrollability" can be handled by changing rate constants, an idea followed up in [19], 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.

In [10] we see the construction and composition of simple logic gates based upon catalytic reactions, but they do not mention control flow or systematic

[3] http://lepton.research.microsoft.com/webgec/

component design in a dual rail setting. In [23] 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. A system of actual chemical reactions is found in [12], where a precise molecular implementation is given for gates complete with a thermodynamic analysis of how the system would evolve, though only for simple gate designs. An implementation of individual dual-rail logic gates that are rate-independent is given in [8]. In contrast, our designs are composable and capable of performing non-trivial computation.

Designs for the Muller C-element, though not the remaining components of an asynchronous device, have been constructed from genetic logic gates [20] and a genetic toggle switch [21], but we are not aware of any other nanoscale designs for asynchronous circuits.

3 Preliminaries

3.1 Chemical Reaction Networks

A CRN C is a finite set R of reactions acting on a finite set S of species. A reaction, which can either be reversible or irreversible, is a triple written in the form $\langle r \in S^{\mathbb{N}}, k \in \mathbb{R}_{>0}, p \in S^{\mathbb{N}} \rangle$, where r and p are the multisets of species reactants and products, respectively, and $k > 0$ is the reaction rate [25]. We work with bi-molecular reactions with uniform rates, including catalytic reactions, and assume mass-action kinetics.

The *stochastic* semantics of a CRN [13] can be given as a continuous-time Markov chain with the state space given as discrete vectors of population counts, which can be solved through the Chemical Master Equation (CME) whose numerical solution may be infeasible for large molecular counts. The *deterministic* semantics approximates the species concentrations over time as a solution of rate equations [28], assuming a continuous state space, but is valid only for high molecular counts and cannot model stochastic fluctuations. A *stochastic approximation* of the CME is possible using the Linear Noise Approximation (LNA) [28], which provides Gaussian distributions for variance. This is a continuous approximation, which is independent of initial populations and hence scalable, and is valid in the limit of high populations. LNA was recently adapted to provide stochastic analysis of the evolution of populations of molecular species of CRNs [6] and extended to probabilistic reachability in [2].

We emphasise that we work with CRNs as an *abstract* programming language for artificial devices, as argued in [10], where we assume that molecules can be designed to carry out the required reactions. In principle, any CRN can be implemented using nucleic-acid strand displacement cascades [26,3], which has been recently experimentally demonstrated in [9].

3.2 Principles of Asynchronous Circuit design

Asynchronous computation [27] is Turing complete [18] meaning that any bounded-tape Turing machine can be implemented with an asynchronous circuit, provid-

ing 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 could be seen as the propagation of a signal down wires of exactly the same length from one component to another.

Asynchronous computation relies on 'local cooperation' in the form of handshaking protocols, rather than a governing clock. These protocols exchange completion signals (high or low, also denoted 1 or 0) in order to establish when a computation has terminated. Asynchronous circuits rely heavily on latches and rendez-vous elements. A rendez-vous element is a component which 'waits' on two or more actions to complete before a system continues. One form of a rendez-vous element is the Muller C-element [27, p.5], which has two Boolean inputs and one output, and is "stateful". When both inputs are the same, the output switches, if necessary, to be equal to the inputs, but when the inputs are different the output remains what it was last time the inputs were equal. The C-element suffices to build a gate that synchronises events, but an isochronous fork, which produces a fan-out of signals that reach the target at virtually the same time, is needed to ensure Turing completeness [18].

C-elements allow a circuit to be speed-independent by a series of local handshakes. This means that we can wait for longer computational paths to complete before advancing without additional computation occurring, negating the use of a system clock. A fundamental construct built from C-elements is a Muller pipeline, shown in Figure 1, which is used to relay handshakes and can be combined with data storage or computational components. The Muller pipeline is constructed by the composition of Muller C-elements and NOT-gates. 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 request initialized 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 1 as the signal *ReqHi*. The receiver then records this data and sets acknowledge to high (*AckHi*). Then the sender responds by setting request to low (*ReqLo*), and finally the receiver acknowledges this by setting acknowledgement to low (*AckLo*). 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.

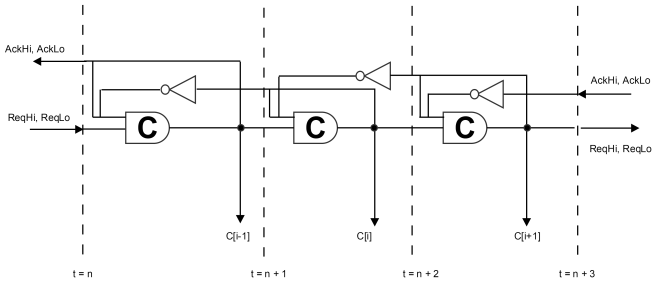


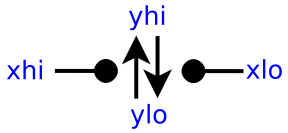
Fig. 1: Signals are propagated from left to right using a Muller pipeline. The pipeline effectively queues data, only allowing a transition to occur when a further signal has been acknowledged.

4 Circuit Construction and Design

Asynchronous computation is well suited to CRNs: they are inherently asynchronous as each reaction happens stochastically and there is no inherent clock that governs their operation.

We use a dual-rail implementation of asynchronous circuits due to the fact that we cannot detect when there are no molecules of a molecular species; we can only detect their presence. This means that there is a separate species or 'signal' representing the values high and low (or logical 1 and 0). These species are named accordingly; for instance, a signal x will be represented by two species xhi and xlo representing the high and low signals. Circuit design follows the normal rules in which components are connected by rails that transport data around the system. We assume standard knowledge of logic gates.

The designs are presented in a diagrammatic notation which allows us to view CRNs as circuits instead of a list of reactions. All our designs are built from a simple motif, seen here, which describes the two reactions:



or in standard CRN notation:

$$xhi + ylo \xrightarrow{k} xhi + yhi \qquad xlo + yhi \xrightarrow{k} xlo + ylo \tag{1}$$

where xhi, xlo are catalytic to the reaction $yhi \leftrightarrows ylo$ and k is the rate which is the same for both reactions, where we set $k = 1$. In general, a black circle represents that the species connected to it is catalytic to the reaction adjacent to the black circle. In our implementation, the presence of every signal is catalytic to another signal, and thus the total number of molecules for each signal (pair

of species) is preserved. Implementations of catalytic gates typically rely on fuel molecules to provide energy to catalyse one species into another [26].

Our motif can be used both as a basic building block for a logical element as well as a control element. For example xhi, xlo, above, can be used to control two separate sub-circuits through yhi, ylo. This demonstrates that, through just two reactions, we can create circuits that exhibit complex behaviour both in control flow and logic.

We assume well-mixed solution, which ensures that the probability of collision between molecules is independent of their position. This yields a circuit that operates correctly but with unknown delays, called delay-insensitive [27].

4.1 Latches and the Muller C-element

A latch is a device used in electronics to store a logical 0 or 1; it needs to have at least two stable states which are cycled between. We present three latch designs in Figure 2(a), each intended to interface in a specific way when used within a larger system. The first, shown in Figure 2(a-i), is almost identical to our motif except for two additional reactions which catalyse ylo to yhi, and vice versa. The latch in Figure 2(a-ii) has an input rhi used to reset the latch to a central state. The advantage of this central state, $ymid$, is that the latch can be in a state where neither yhi nor ylo are present, which is useful if these reactions are catalytic to any other component. In its electronic counterpart, a system may have a rail deciding whether the component is active or inactive; our intermediary $ymid$ fulfils this function, as the system is in a state where neither yhi nor ylo are present. In Figure 2(a-iii), we see the latter latch combined with control species chi, clo. These species are used to synchronise the latch in the pipeline. Even when the input signal xhi, xlo is present, we still need the species chi to be present in order to catalyse the reaction $s_2 \to shi$ or $s_4 \to slo$, which are the output species. This is needed since these output species cannot be read until the system has synchronised.

A C-element is conceptually similar to a latch except for having two inputs, x and y. The design for this, presented in Figure 2(b-i), was inspired by the Approximate Majority (AM) circuit in [5]. The circuit in Figure 2(b-ii) is similar to AM of [5] except for separating the inputs. Note that the C-element includes two separate AM circuits, shown schematically in Figure 2(b-iii), where the arrows in the AM box indicate the direction of switching between the two stable states. Using this mechanism it is possible to trap the gate into the state *zdn* or *zup* given both high inputs *xhi, yhi* or both low inputs *xlo, ylo*. However, when one of these inputs is changed, we see that the gate is trapped in one of these states in view of the feedback loops. The second AM mechanism corrects the weak output signal *zdn, zup* from the first AM mechanism, meaning the outputs of this gate are *zlo* and *zhi*.

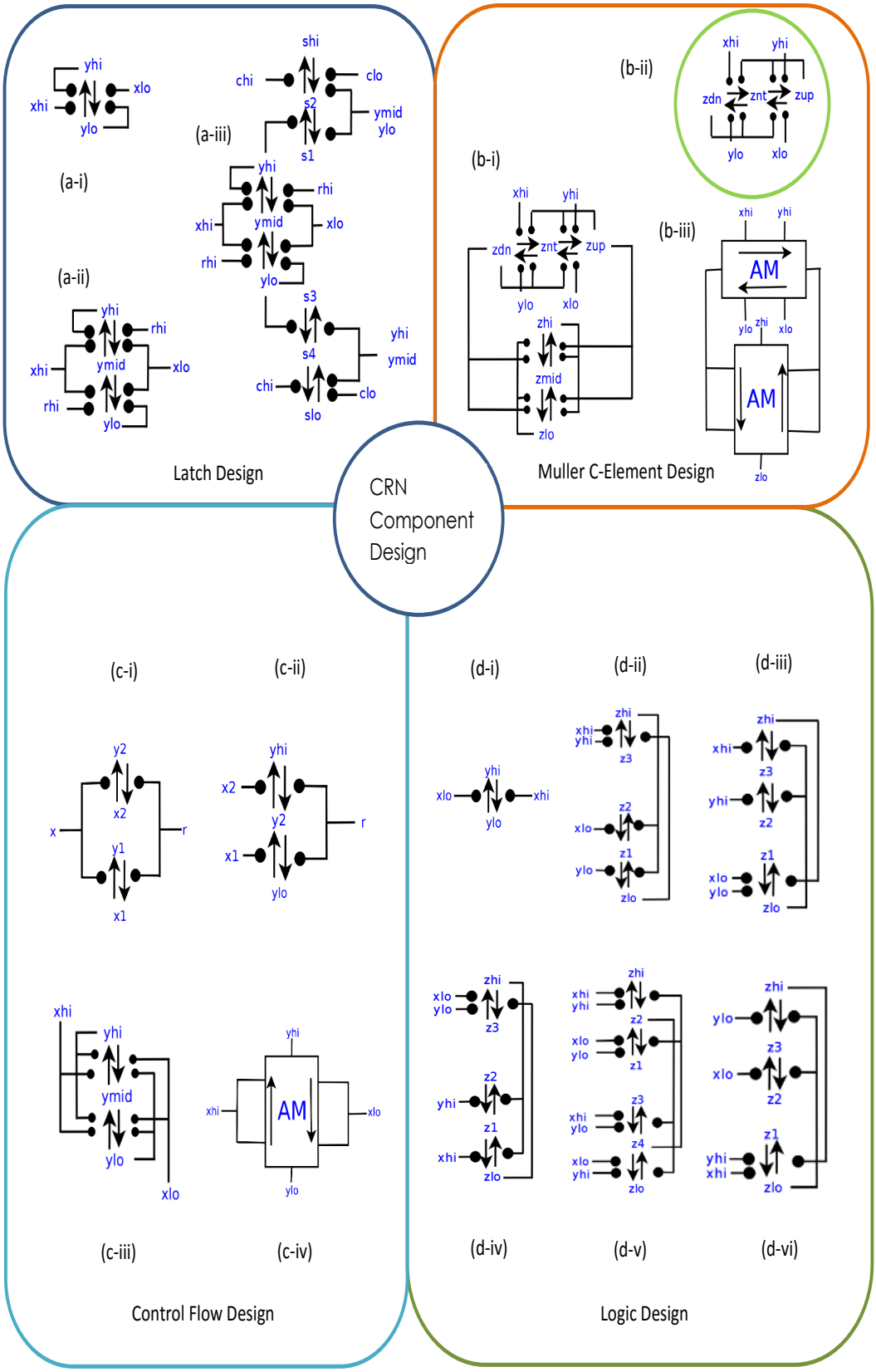


Figure 2 a) Latches – (i) Two state latch (ii) Three state latch (iii) Latch with control input
b) Muller C-Element – (i) C-Element (ii) Approximate Majority circuit based on [3]
(iii) C-Element as composition of Approximate Majority circuits
c) Control Flow Design – (i) Fork (ii) Join (iii) Arbiter (iv) Arbiter as AM circuit
d) Logic Gates – (i) NOT (ii) AND (iii) OR (iv) NAND (v) XOR (vi) NOR

4.2 Logic and Arithmetic

Although gate designs for Boolean operators have been proposed in CRNs [25], we present dual-rail implementations of logic gates in line with other designs proposed within this paper. In contrast to the gates in [25], our gates account for all inputs x and y, and also respond to change in input. The AND-gate, shown in Figure 2(d-ii), has inputs *x, y* expressed in our dual-rail implementation. With the presence of species *yhi* we can catalyse z into the state *zmid*, and with the species *xhi* we can catalyse *zmid* to *zhi*; thus both species are needed for the gate to output the signal z. The state *zhi* converts any species *zlo* back to *z1*, therefore showing that only one output signal can be present at any time. Conversely, with either *xlo, ylo* we can convert *z1* to *zlo*, which in turn can convert *zhi* back to *zmid* and *zmid* to *z2*. Using a similar trail of thought we can see how the other gates are devised, albeit 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.

Using these designs, we have also implemented a ripple carry adder, seen in Figure 7. An individual adder is composed of two XOR gates, two OR gates and an AND gate. It takes two inputs x, y and outputs the sum of the inputs z with a carry bit c. In our ripple carry implementation we compose three of these adders in series.

4.3 Control Flow

Control flow is used to mediate or propagate the flow of information throughout a system. The fork, shown Figure 2(c-i), is used to split signals; it is constructed by having one input species x catalyse the two reactions $x_1 \to y_1$ and $x_2 \to y_2$, to produce two outputs y_1, y_2. The species r acts as a reset for the fork if the process needs repeating, assuming x is no longer active. The join, see Figure 2(c-ii), is similar to the function of an AND-gate, and will only output a signal *yhi* when both inputs x_1, x_2 are present. This is a useful control mechanism since the system can stall the catalysis of further reactions via y until both input signals x_1 and x_2 are present. The species r can be used to reset the join. We also present our AM circuit as an arbiter seen in Figure 2(c-iii). An arbiter is used to decide an output signal based on which species arrived first. The AM circuit works well as an arbiter due to the fact that the output yhi, ylo starts to be converted from *ymid* as soon as either of xhi, xlo arrives, therefore automatically biasing whichever species is present first. All three of these control flow elements are used in our queue and adder implementations discussed within the next section.

5 Design Validation

We use Microsoft's Visual GEC tool to establish that the designs[4] exhibit correct behaviour, both for the deterministic and stochastic semantics of the CRNs.

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

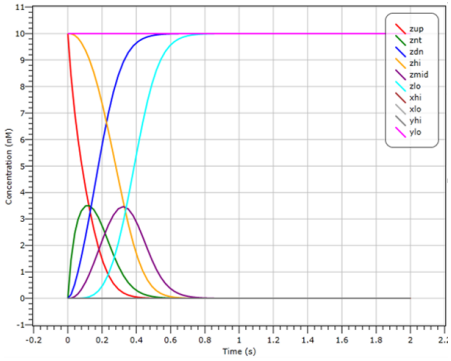
Visual GEC provides a programming language, LBS, for designing and simulating any given CRN. Using numerical simulation, we systematically tested each component in isolation by simulating its behaviour over all inputs, and then checking that those inputs yield the desired output and also suppress any unwanted outputs. 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 $xhi \xrightarrow{k} xlo$, which converts all of the signal xhi into a signal xlo while the component is operating.

Since deterministic semantics is not accurate for low molecular populations, we additionally explored its stochastic semantics. Visual GEC exports models to the probabilistic model checker PRISM [15], 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 of Figure 1, 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 used an experimental implementation of the LNA within Visual GEC, based on [6]. The LNA approximates the CME with a set of differential equations, quadratic in the number of species and independent of the initial number of molecules. The ODEs describe the time evolution of expected value and variance. As well as being capable of checking temporal logic properties [6,2], 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.

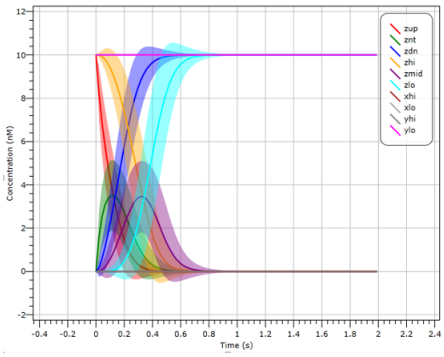
We now illustrate the results of the validation on a selection of components.

Muller C-element
Firstly, we demonstrate the robustness of the Muller C-element against changes in input signal in Figure 3(a-c). In Figure 3(d), we show that the C-element may not be robust at low molecular counts, here 10. Increasing this count to 500 greatly decreases the variance of the output species (not shown), reducing the likelihood that the wrong species will reach the threshold.

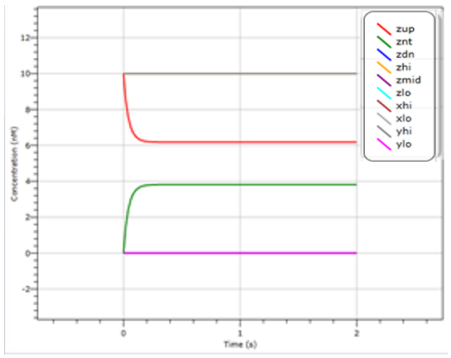
Pipeline
The pipeline, seen in Figure 1, is a mechanism that relays handshakes between components, for example latches to store data. We construct the pipeline by placing three of our C-element CRNs in sequence. At each intermediate stage



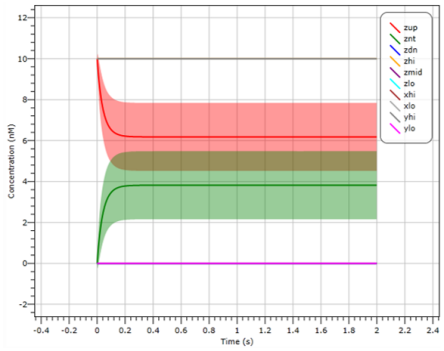
(a) An input change from xhi, yhi to xlo, ylo



(b) LNA simulation of a change in input from xhi, yhi to xlo, ylo



(c) An input change from xhi, yhi to xlo, yhi

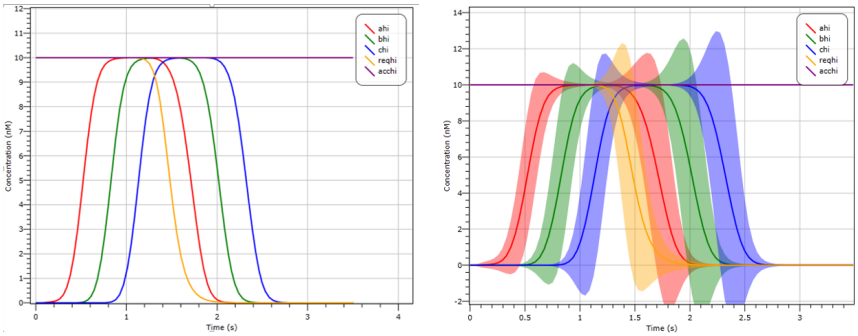


(d) LNA simulation of a change in input from xhi, yhi to xlo, yhi

Fig. 3: Validation of the Muller C-element. In these experiments we start with an input of x and y, the presence of which are represented by the species xhi and yhi. In (a) we show a change of input where both x and y change to 0 or are not present, represented by species xlo and ylo. Note how zhi responds by reaching zero molecules and zlo reaches the maximum molecular value, in this case 10. In (b) we show the LNA approximation of the same scenario, with standard deviation shown as highlighted regions, which demonstrates that the variance is low once the circuit reaches steady state. In (c) we demonstrate the change in one input value xhi across a single Approximate Majority circuit; in this case the output signal decreases, but still remains at a value greater than zlo. However, for this reason we add an additional AM circuit to further separate the output signals zhi, zlo. In (d), we show the LNA of the same scenario, demonstrating that the circuit is not robust under low molecular count.

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.

Because we have already validated the individual C-element design, we can assume that they work correctly and so we only need to observe the behaviour of the overall system. We therefore analyse the system behaviour over time, based upon a change in inputs, namely, signals *reqhi, reqlo, acchi* and *acclo*. We conducted multiple experiments in which we change these inputs, demonstrating the desired effect of them being propagated along the pipeline. This is seen as a 'wave' through the pipeline propagating a high signal and then a low signal. The results of this are shown in Figure 4. Here the presence of the species ahi, bhi, chi represents a high signal before responding and diminishing back to zero.



(a) The Muller C-pipeline responding to the input species *reqhi* being present and then transforming to the input species *reqlo*.

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

Fig. 4: Validation of the Muller C-pipeline. The input request signal, encoded by the species *reqhi*, is propagated to the end of the pipeline (represented by the species ahi, bhi, chi); we then set the request signal to low. The pipeline then responds by the presence of ahi, bhi and chi diminishing to zero. In (b) we show that the variance is low, even for low molecular counts.

Queue

We have also designed and validated a queue, shown in Figure 5, built by the addition of latches at each C-element block to the Muller pipeline. The queue uses the pipeline as a control mechanism to propagate signals between the latches. We use the complex latch in Figure 2(a-iii) for this purpose. As a high species 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. In Figure 6 we analyse the oscillatory behaviour of the queue using the LNA, demonstrating its robustness at high molecular counts.

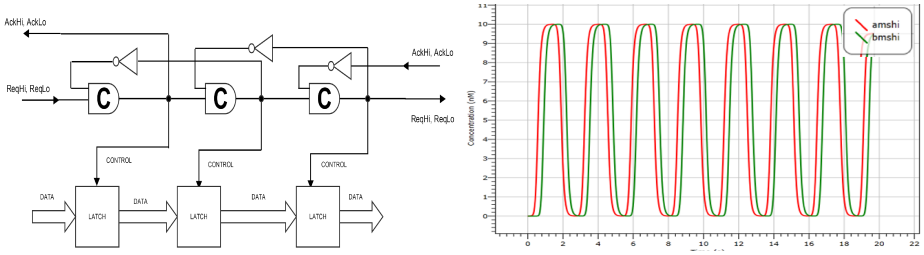


Fig. 5: Deterministic simulation of the queue pipeline. We propagate a value of 1 through the queue. The species *amshi, bmshi* represent the outputs of the first and second latches. Note that through oscillatory patterns generated by the pipeline we can mimick properties of a synchronous system.

Adder

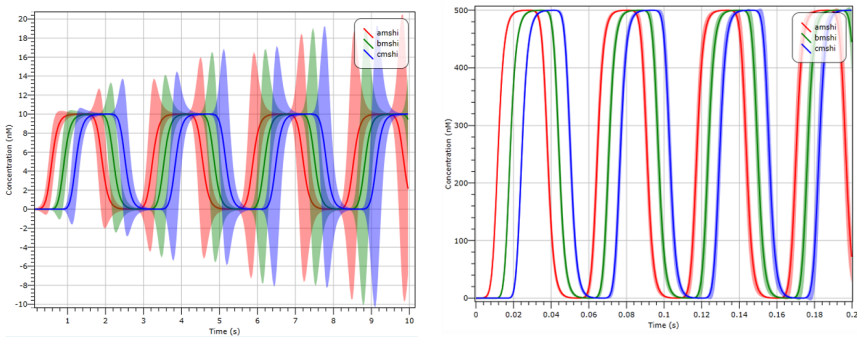
We have also designed a three bit ripple carry adder seen in Figure 7, which works in a similar fashion to the queue but instead of latches we compose adders in series. At each time step we input two bits and a carry, which outputs the sum and a carry. In this way we can add two three-bit numbers together. We show in Figure 8 that the adder exhibits correct behaviour, and each sum is calculated only in the next stage in the pipeline.

5.1 Discussion

Direct chemical implementations of CRNs have been theorised and realised, but involve complicated reaction mechanisms [24]. For instance, [14] implements chemical systems as neural networks. Most implementations need some external fuel molecules, as reactions such as $A + B \rightarrow C + B$ require some energy input in order to catalyse one species to another [26]. CRNs have been implemented in systems involving *Toehold Mediated Branch Migration and Strand Displacement* (DSD). DNA strand displacement has already been shown to be a universal substrate for chemical kinetics, specifically for bi-molecular reactions [26]. In addition to modelling the behaviours at the CRN level, we also implemented our CRN designs in two-domain DNA strand displacement devices [3] using the Visual DSD tool [16], thus providing further evidence of their experimental viability, at least for the construction of DNA-based devices.

6 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 [1,5], 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



(a) The LNA for 10 molecules. (b) The LNA for 500 molecules.

Fig. 6: LNA simulation of the queue pipeline. In these plots we show standard deviation, calculated through LNA, of an oscillatory pattern created by propagating a value of 1 and then 0. The maximum molecular count for each species in (a) is 10 while in (b) is 500. The variance decreases greatly with an increase in molecular count. We plot the values of $amshi, bmshi, cmshi$, which represent a value of 1. The troughs indicate when 0 is propagated.

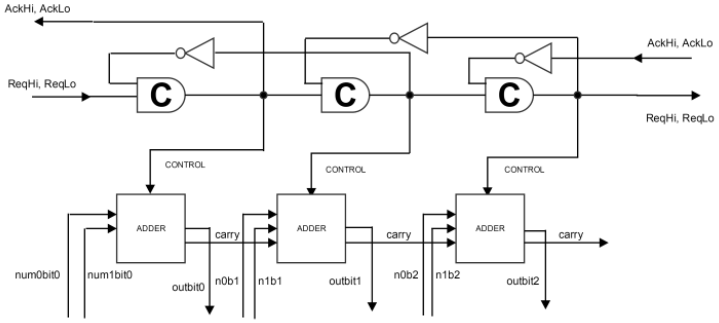
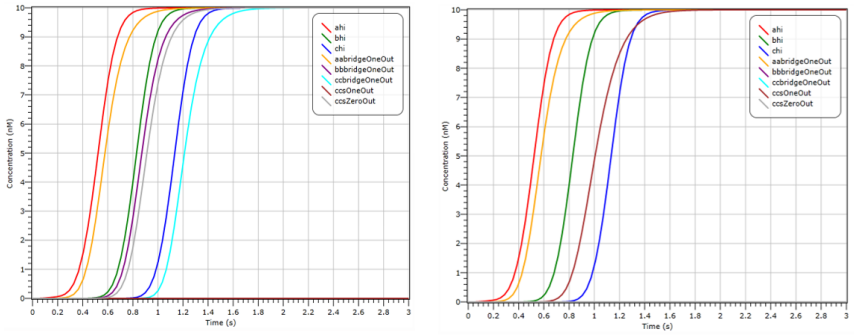


Fig. 7: Full Ripple Carry Adder used in conjunction with our Muller C-pipeline to stagger computation across the adders.

CRN can be physically realised using DNA strand displacement [26], as recently demonstrated experimentally in [9], the proposed designs are in principle implementable, and we have confirmed this in theory by modelling them in the two-domain setting [3] using Visual DSD [22,16]. 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.



(a) Adder response to value of 101010. (b) Adder response to value of 100010.

Fig. 8: Deterministic simulation of the adder circuit responding to various inputs. We overlay this with signals present in the pipeline used to coordinate the carry bit from each adder, represented by *aabridgeOneOut*, *bbbridgeOneOut* and *ccbridgeOneOut*. In (b), the final output signals cross due to pre-calculation by the adders before the carry bit arrives.

References

1. D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.
2. 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.
3. L. Cardelli. Two-domain DNA strand displacement. *Developments in Computational Models*, 26:47–61, 2010.
4. L. Cardelli. Morphisms of reaction networks that couple structure to function. *BMC systems biology*, 8(1):84, 2014.
5. L. Cardelli and A. Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific Reports*, 2, 2012.
6. 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.
7. H.-L. Chen, D. Doty, and D. Soloveichik. Deterministic function computation with chemical reaction networks. *Natural Computing*, 13(4):517–534, 2013.
8. 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.
9. 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.
10. M. Cook, D. Soloveichik, E. Winfree, and J. Bruck. Programmability of chemical reaction networks. In *Algorithmic Bioprocesses*, pages 543–584. Springer, 2009.

11. 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.
12. A. P. de Silva and N. D. McClenaghan. Molecular-scale logic gates. *Chemistry–A European Journal*, 10(3):574–586, 2004.
13. D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
14. A. Hjelmfelt, E. D. Weinberger, and J. Ross. Chemical implementation of finite-state machines. *Proceedings of the National Academy of Sciences*, 89(1):383–387, 1992.
15. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
16. 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.
17. M. O. Magnasco. Chemical kinetics is Turing universal. *Phys. Rev. Lett.*, 78:1190–1193, Feb 1997.
18. R. Manohar and A. J. Martin. Quasi-delay-insensitive circuits are Turing-complete. Technical report, DTIC Document, 1995.
19. 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.
20. 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.
21. 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.
22. A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *J R Soc Interface*, 6 Suppl 4:S419–S436, Aug 2009.
23. P. Senum and M. Riedel. Rate-independent constructs for chemical computation. *PloS One*, 6(6):e21414, 2011.
24. S. W. Shin. *Compiling and verifying DNA-based chemical reaction network implementations*. PhD thesis, California Institute of Technology, 2011.
25. D. Soloveichik, M. Cook, E. Winfree, and J. Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.
26. 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.
27. J. Spars and S. Furber. *Principles Asynchronous Circuit Design*. Springer, 2002.
28. N. G. Van Kampen. *Stochastic processes in physics and chemistry*, volume 1. Elsevier, 1992.