# SCIENTIFIC REPORTS

**OPEN**

# Computing Mathematical Functions using DNA via Fractional Coding

Sayed Ahmad Salehi, Xingyi Liu, Marc D. Riedel & Keshab K. Parhi [ID]

This paper discusses the implementation of mathematical functions such as exponentials, trigonometric functions, the sigmoid function and the perceptron function with molecular reactions in general, and DNA strand displacement reactions in particular. The molecular constructs for these functions are predicated on a novel representation for input and output values: a fractional *encoding*, in which values are represented by the relative concentrations of two molecular types, denoted as type-1 and type-0. This representation is inspired by a technique from digital electronic design, termed *stochastic* logic, in which values are represented by the probability of 1's in a stream of randomly generated 0's and 1's. Research in the electronic realm has shown that a variety of complex functions can be computed with remarkably simple circuitry with this stochastic approach. This paper demonstrates how stochastic electronic designs can be translated to molecular circuits. It presents molecular implementations of mathematical functions that are considerably more complex than any shown to date. All designs are validated using mass-action simulations of the chemical kinetics of DNA strand displacement reactions.

Molecular computing holds the promise for transforming research in areas such as disease monitoring and drug delivery. Since early, pioneering work by Adleman[1], the field has evolved significantly. A particularly promising strategy for molecular computation is based on the mechanism of DNA strand displacement[2–6].

Various computational structures have been proposed for DNA-based systems, as well as in other contexts. Simple logic primitives, such as AND, OR, NAND, NOR, and XOR have been demonstrated[7–14]. These circuits have been used as building blocks for both digital signal processing[15–20], and mixed-signal (i.e., analog and digital) computation[17,21]. Using these simple circuits, complex genetic circuits have been constructed to perform computation in cells[22]. To automate the design of genetic and DNA circuits, computer-aided design (CAD) systems have been presented[22,23]. There has been interest in activating and inhibiting pathways by "filtering" concentrations of molecular types in different frequency bands[24–26].

The theory of computing with abstract chemical reactions, termed chemical reaction networks (CRNs), has evolved into a bona fide *computer programming framework*. Work on CRNs includes programs for computing different sorts of mathematical functions such as polynomials[27,28], and logarithms[29,30].

In prior work, different approaches to compute complex functions such as exponentials and sigmoids have been presented. For example, it has been shown that CRNs describing covalent modification cycle can realize exponential, logarithm and sigmoid functions[29,31]. The hyperbolic regime can be used to realize exponential and logarithm while ultrasensitive regime can be used to realize sigmoid function. The CRNs in[29,31] describe analog behavior of the system while the CRNs described by the proposed approach describe digital behavior. In[29,31], each region of operation is described by a specific input-output characteristic. This implies that the exponential and sigmoid functions are realized for specific ranges of input concentrations. Furthermore, the sigmoid function in[29] describes a hard-limit response. On the other hand, the proposed approach realizes digital circuits and the function behavior is not limited to a specific range of input concentrations (only the ratio of two concentrations used to represent a variable is important not their concentrations).

This paper presents a method for designing CRNs that compute a wide range of mathematical functions, ranging from simple to complex. The building blocks in the proposed methodology are units composed of four chemical reactions. All chemical reactions in the proposed system have exactly two reactants. Such bimolecular chemical reactions can be implemented as DNA strand-displacement reactions in a robust way[32]. Thus, our

Department of Electrical and Computer Engineering, University of Minnesota, 200 Union St. S.E., Minneapolis, MN, 55455, USA. Correspondence and requests for materials should be addressed to K.K.P. (email: parhi@umn.edu)
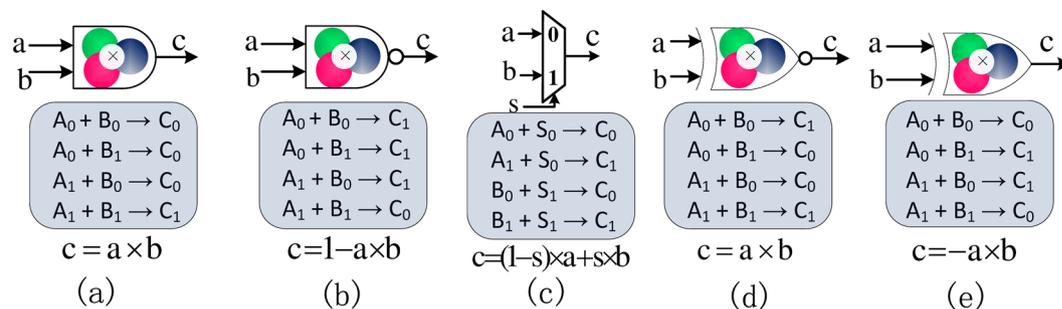
**Figure 1.** Basic molecular units. (**a**) The Multiplication unit, Mult. This unit calculates $c = a \times b$, the multiplication of two input variables $a$ and $b$ in the unipolar fractional representation. (**b**) The NMult unit. This unit computes $c = 1 - a \times b$ in the unipolar fractional representation. (**c**) The MUX unit. This unit performs scaled addition. Here $a,b$ and $c$ can be in the unipolar or the bipolar representation, while $s$ must be in in unipolar representation. (**d**) The bipolar Mult unit. This unit performs multiplication in the bipolar fractional representation. (**e**) The bipolar NMult unit. This unit computes $c = -a \times b$ in the bipolar fractional representation. Note the symbols that we will use to represent the different units are shown above the CRNs.
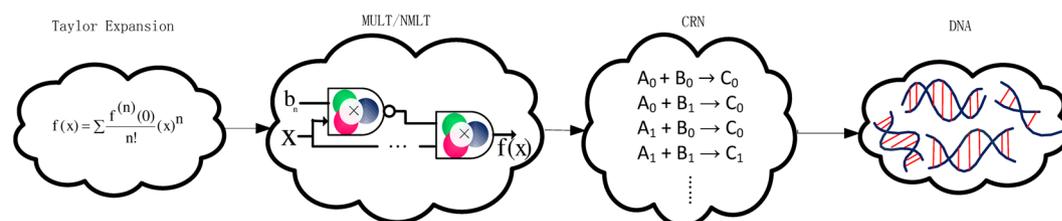


**Figure 2.** The proposed methodology. This figure shows the required steps for computing functions based on the proposed methodology. It starts with the approximation of the desired function as a polynomial using a series expansion method. The polynomial is then expressed in an equivalent form that only contains Mult and NMult units. The Mult and NMult units are mapped to their equivalent chemical reactions and finally the CRN is implemented by DNA strand-displacement reactions.

method provides a systematic way to design DNA reactions that compute mathematical functions. These computational constructs are central to the topic of perceptrons, that represent simple machine learning algorithm.

Machine learning classifiers have become ubiquitous in the computational sciences. Their physical realization using different technologies has been considered[33,34]. Molecular implementations of machine learning classifiers could have important applications. One can imagine instances where *inference* and *learning* might be an integral part of tasks such as biochemical sensing. For example, genetic logic circuits for cell classification can sense features of mRNAs; they can detect their expression patterns and selectively respond to specific cell types[35–38]. Such circuits could enable the production of personalized "smart" drugs that target specific diseases for specific patients[39].

Past work on neural computation with molecular reactions includes[40–44]. As early theoretical research on the topic, Hjelmfelt *et al.* presented chemical reactions that, based on the ordinary differential equations of mass action kinetics model, can emulate so-called McClulloch-Pitts neurons[44]. These chemical neurons can be coupled together to build chemical neural networks or finite-state machine[45]. Also in a theoretical vein, Mills *et al.* described a DNA implementation of a Hopfield neural network as well as a DNA implementation of a multi-layer perceptron[46,47]. The authors speculated that networks containing as many as $10^9$ neurons might be feasible.

Laplante *et al.* performed pattern recognition with chemical (as opposed to biomolecular) reactions, in a continuous flow stirred tank reactors[48]. Lim *et al.* implemented pattern recognition with differentially-labeled probe DNA molecules that competitively hybridized to compute the decisions[49]. Zhang and Seelig described an implementation of a linear classifier using DNA strand displacement[50]. Design of DNA circuits for supervised learning of a class of linear functions using buffered strand displacement reactions has been presented in[51]. Finally, Qian *et al.* demonstrated a complete artificial neural network, implemented experimentally using DNA strand displacement[52].

In general, an artificial neural network consists of one or more layers where, in each layer, a neuron computes a weighted sum followed by a nonlinear activation (transfer) function. Typically the activation function corresponds to a sigmoid function. Prior work on molecular implementations of ANNs has considered either a hard-threshold[52] or linear transfer function[50] as the activation function.

This paper discusses the implementation of mathematical functions such as exponentials, trigonometric functions, the sigmoid function and a perceptron function with the limitation that the weighted sum of the inputs is scaled down by the dimension of the input vector.
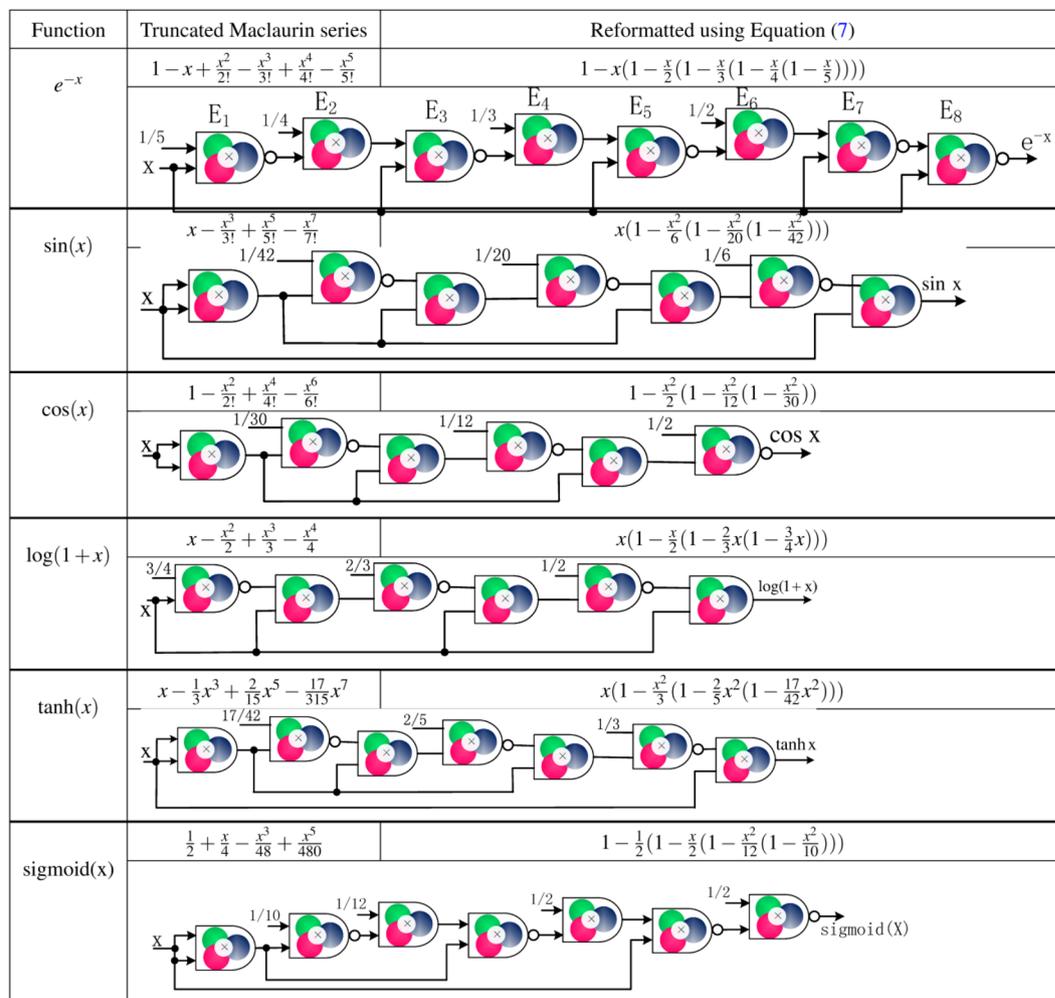
**Figure 3.** Examples of molecular circuits for mathematical functions. Truncated Maclaurin series, reformatted Maclaurin series using Horner's rule, and Mult/NMult structure for functions in equations (41)–(46) of the Supplementary Information.

In prior work on molecular computing, two types of *representation* for the input and output variables of chemical reaction networks (CRNs) have been considered:

1. The value of each variable corresponds to the concentration of a specific molecular type; this is referred to as a direct representation.
2. The value of each variable is represented by the difference between the concentrations of a pair of molecular types; this is referred to as a dual-rail representation[53].

In recent work, we have proposed a new type of representation, referred to as a *fractional representation*[28]. Here a pair of molecular types is assigned to each variable, e.g., $(X_0, X_1)$ for a variable $x$. The value of the variable is determined by the ratio of the concentrations for the assigned pair,
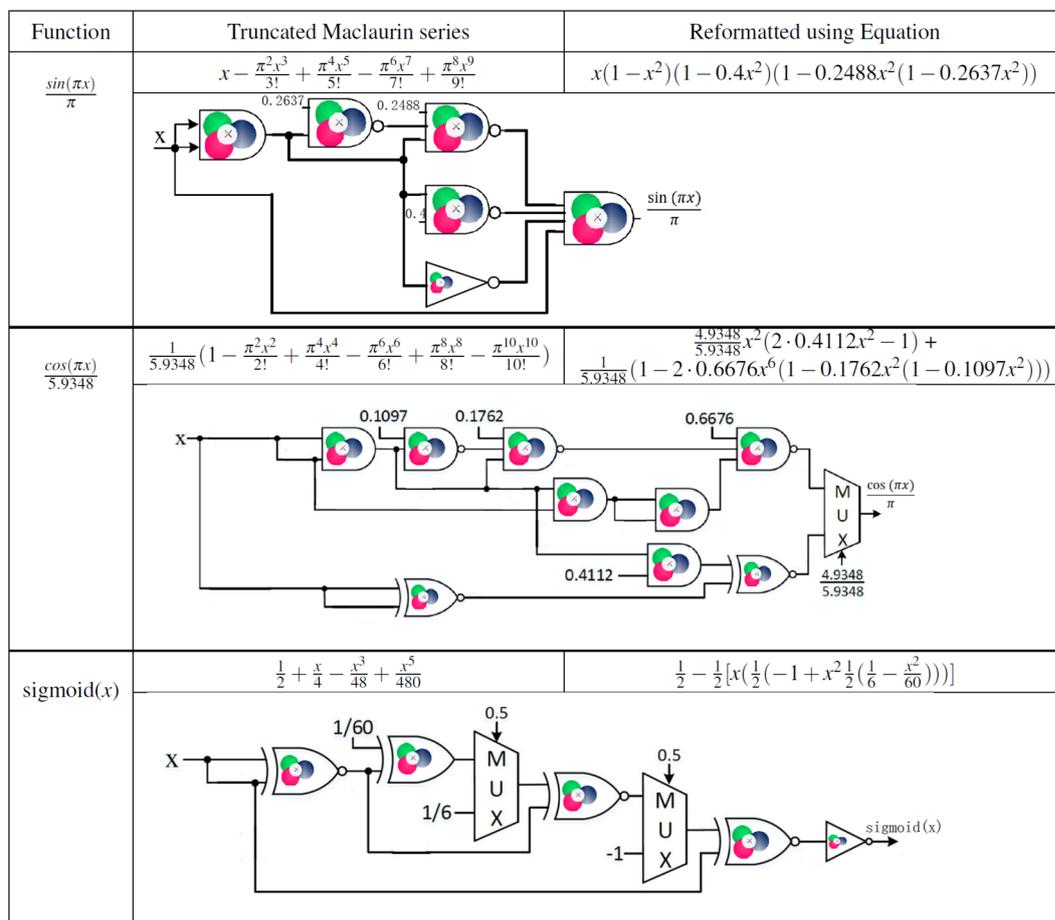
$$x = \frac{[X_1]}{[X_0] + [X_1]} \tag{1}$$

where $[X_1]$ and $[X_0]$ represent concentrations of molecules $X_1$ and $X_0$, respectively. Note that the value of $x$ is confined to the unit interval, [0, 1]. With the values confined to the unit interval, we refer to the representations as a *unipolar* fractional encoding.

Variables with values in the range $[-1, 1]$ can be represented by a slightly different encoding on the assigned pair $(X_0, X_1)$, given by:

$$x = \frac{[X_1] - [X_0]}{[X_0] + [X_1]}. \tag{2}$$

We refer to this representation as a *bipolar* fractional encoding.

| Function | Truncated Maclaurin series | Reformatted using Equation |
|---|---|---|
| $\frac{sin(\pi x)}{\pi}$ | $x - \frac{\pi^2 x^3}{3!} + \frac{\pi^4 x^5}{5!} - \frac{\pi^6 x^7}{7!} + \frac{\pi^8 x^9}{9!}$ | $x(1-x^2)(1-0.4x^2)(1-0.2488x^2(1-0.2637x^2))$ |
| |  | |
| $\frac{cos(\pi x)}{5.9348}$ | $\frac{1}{5.9348}\left(1 - \frac{\pi^2 x^2}{2!} + \frac{\pi^4 x^4}{4!} - \frac{\pi^6 x^6}{6!} + \frac{\pi^8 x^8}{8!} - \frac{\pi^{10} x^{10}}{10!}\right)$ | $\frac{4.9348}{5.9348}x^2(2 \cdot 0.4112x^2 - 1) +$ $\frac{1}{5.9348}(1 - 2 \cdot 0.6676x^6(1 - 0.1762x^2(1 - 0.1097x^2)))$ |
| |  | |
| sigmoid($x$) | $\frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + \frac{x^5}{480}$ | $\frac{1}{2} - \frac{1}{2}[x(\frac{1}{2}(-1 + x^2\frac{1}{2}(\frac{1}{6} - \frac{x^2}{60})))]$ |
| |  | |

**Figure 4.** Examples of molecular circuits for mathematical functions with inputs covering entire range. Truncated Maclaurin series, reformatted Maclaurin series using Horner's rule, Mult/NMult and MUX structure for functions in equations (47), (48) and (46) of the Supplementary Information. The output of the cosine function and the input of the bipolar sigmoid are in bipolar representation.
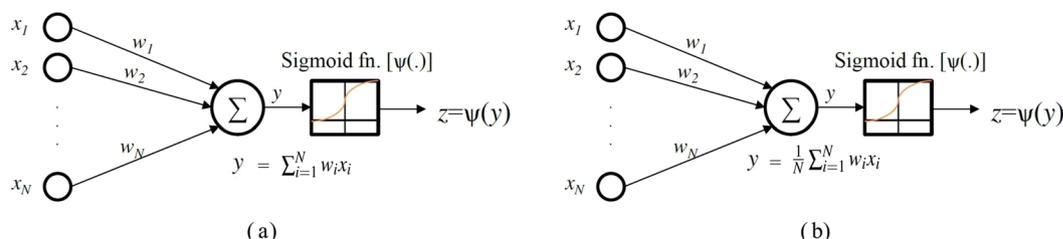


**Figure 5.** Molecular Perceptron. (**a**) A standard perceptron that computes sigmoid ($\sum_{i=1}^{N} w_i x_i$), (**b**) A molecular perceptron that computes sigmoid $\left(\frac{1}{N}\sum_{i=1}^{N} w_i x_i\right)$.

The unipolar fractional coding and the connection that it makes between molecular computation and electronic stochastic logic design have been introduced in[28]. However, the extension of the idea for bipolar fractional coding and a systematic method for molecular implementation of complex functions using fractional coding has not been reported in prior work. The contributions of this paper are twofold. Firstly, molecular reactions are proposed to compute operations such as $ab$, $1-ab$, and $sa + (1-s)b$ using both the unipolar and bipolar fractional representations. These molecular circuits are, respectively, referred to as Mult, NMult, and MuX. Secondly, this paper demonstrates that unipolar and bipolar fractional coding approaches can be used to design CRNs for computing complex mathematical functions such as $e^{-x}$, $\sin(x)$, and sigmoid ($x$). The proposed CRNs can readily be implemented using DNA strand displacement.

The fractional representation is inspired by a technique from digital electronic design, termed *stochastic logic*, in which values are represented by the probability of seeing 1's in a stream of randomly generated 0's and 1's[54–59]. Research in the electronic realm has shown that a variety of complex functions can be computed with remarkably simple circuitry with this stochastic approach.
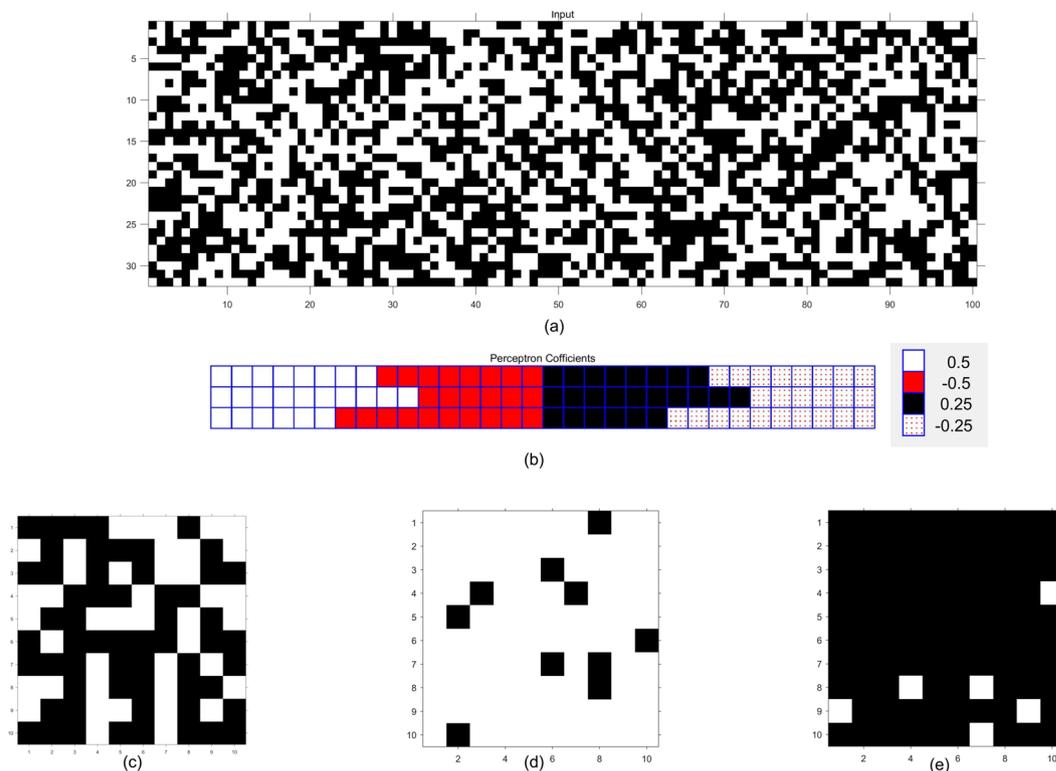
**Figure 6.** Inputs, weights and outputs of three perceptrons, denoted A, B and C. (**a**) Inputs to the perceptron: each column of the $32 \times 100$ matrix illustrates an input vector containing 32 binary inputs. Each white square corresponds to a 1 and each black square corresponds to a 0. (**b**) Weights: The weights for the three perceptrons are illustrated. These weights are divided into 4 parts and correspond to $1/2, -1/2, 1/4$ and $-1/4$ from left to right. (**c**) Binary outputs of Perceptron A containing 58 1's and 42 0's. (**d**) Binary outputs of Perceptron B containing 90 1's and 10 0's. (**e**) Binary outputs of Perceptron C containing 6 1's and 94 0's.

The main difference between[28] and this paper lies in the approach proposed to design and synthesize computing CRNs. The approach in[28] uses so-called control generating reactions and the transferring reactions that lead to reactions with $m$ reactants for a polynomial of degree $m$. In contrast, this paper uses simple molecular units such as `Mult` and `NMult` described in the next section. Regardless of the complexity of the target functions, the molecular reactions designed by the new approach are only composed of simple reactions with two reactants and one product. These reactions are more suitable for DNA implementation. The molecular implementations presented in this paper are inspired by the stochastic implementations of functions presented in[60].

The fractional encoding discussed in this paper is analogous to the stochastic representation. The concentrations of the $X_0$ and $X_1$ molecular types, correspond to the probability of seeing 0's and 1's, respectively, in the random streams. This paper demonstrates how stochastic electronic designs can be translated to molecular circuits.

One should notice that the bipolar fractional coding is just a representation of the value of a variable using two molecular types. This means that it is not required to actually calculate Equation (2). In other words, Equation (2) is our interpretation for the value of a variable and molecular reactions do not calculate this equation.

Section 1 introduces molecular reactions for the `Mult` and `NMult` units; these perform multiplication in the unipolar fractional representation. Section 2 presents an approach for mapping specific target functions to a cascade of `Mult/NMult` units. Section 3 introduces a molecular MUX unit that performs scaled addition, as well as `Mult/NMult` units for multiplication using the bipolar representation. Section 3 also presents an application: CRNs for implementing a single-layered neural network (also referred to as a perceptron). Section 4 discusses the DNA implementations of the proposed CRNs.

## CRNs for Multiplication Units

Based on the fractional coding discussed in the previous section, we propose two simple sets of CRNs for computing multiplication. We refer to these as `Mult` and `NMult`. These sets will serve as fundamental units in the construction of other desired functions in Section 2. `Mult` computes $c = a \times b$, and `NMult` computes $c = 1 - a \times b$ where $a,b$, and $c$ are in the unipolar fractional representation. The units are described below.

**Mult unit.** Consider the four reactions shown in Fig. 1(a). These compute $c$ as the multiplication of two inputs $a$ and $b$, all in unipolar fractional representation. So if $a = \frac{[A_1]}{[A_0] + [A_1]}$ and $b = \frac{[B_1]}{[B_0] + [B_1]}$ then $c = \frac{[C_1]}{[C_0] + [C_1]} = a \times b$.

We prove this in Supplementary Section S.1, on the basis of both stochastic and ordinary differential equations.
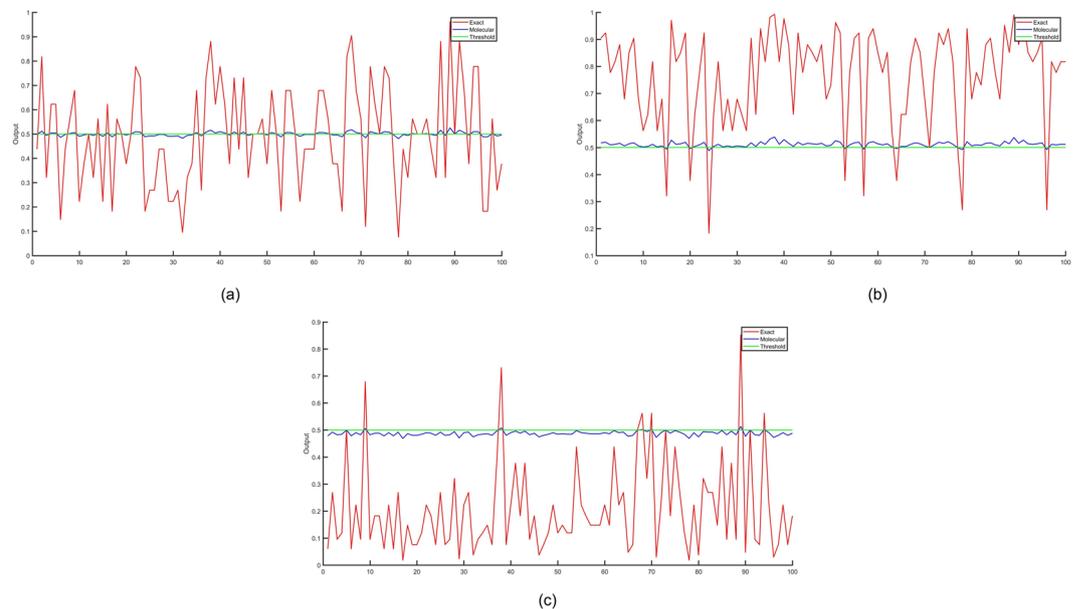
**Figure 7.** Exact perceptron outputs that represent sigmoid of the weighted sum of the inputs and the molecular perceptron outputs that compute sigmoid of the weighted sum scaled by a factor 1/32 for the 100 input vectors for: (**a**) Perceptron A, (**b**) Perceptron B, (**c**) Perceptron C. The *x* axis corresponds to input vector number.
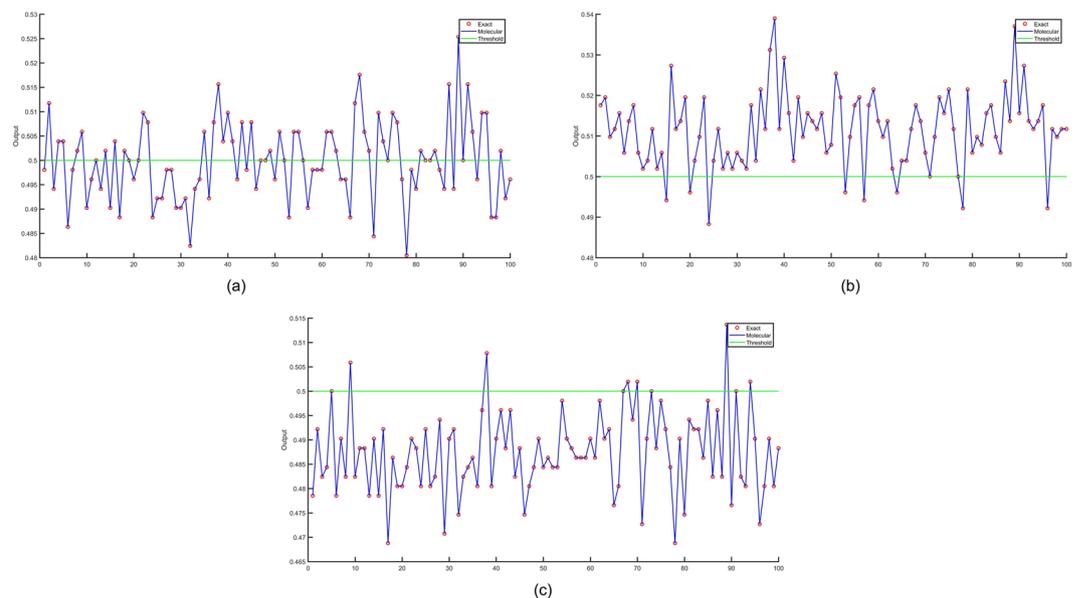


**Figure 8.** Exact and molecular perceptron outputs with weighted sum of the inputs scaled by 1/32 for 100 input vectors for: (**a**) Perceptron A, (**b**) Perceptron B, (**c**) Perceptron C. The *x* axis corresponds to input vector number.

**NMult unit.** If we switch $C_0$ and $C_1$ in the molecular reactions of the `Mult` unit, we obtain what we call an `NMult` unit which computes $1 - a \times b$ in the unipolar fractional coding. Figure 1(b) shows the corresponding set of reactions. The proof that the `NMult` unit computes $1 - a \times b$ is very similar to the proof for `Mult` unit. It can be obtained by switching $C_0$ and $C_1$ in the proof presented for `Mult` unit.

Note that the CRNs in Fig. 1 do not preserve the initial values of the input molecular types. The reactions can be modified such that the initial concentrations of either one or both of the input pairs, $(A_0, A_1)$ and $(B_0, B_1)$, are preserved. The details are presented in Section S.2 of the Supplementary Information.

Figure 1 shows three additional units. For some functions we use a CRN unit called `MUX`, shown in Fig. 1(c). To perform multiplication on the bipolar fractional coding, we use the CRN units shown in Fig. 1(d) and (e). All three CRN units are described in detail in Section 3 where we use them to compute the bipolar sigmoid function.
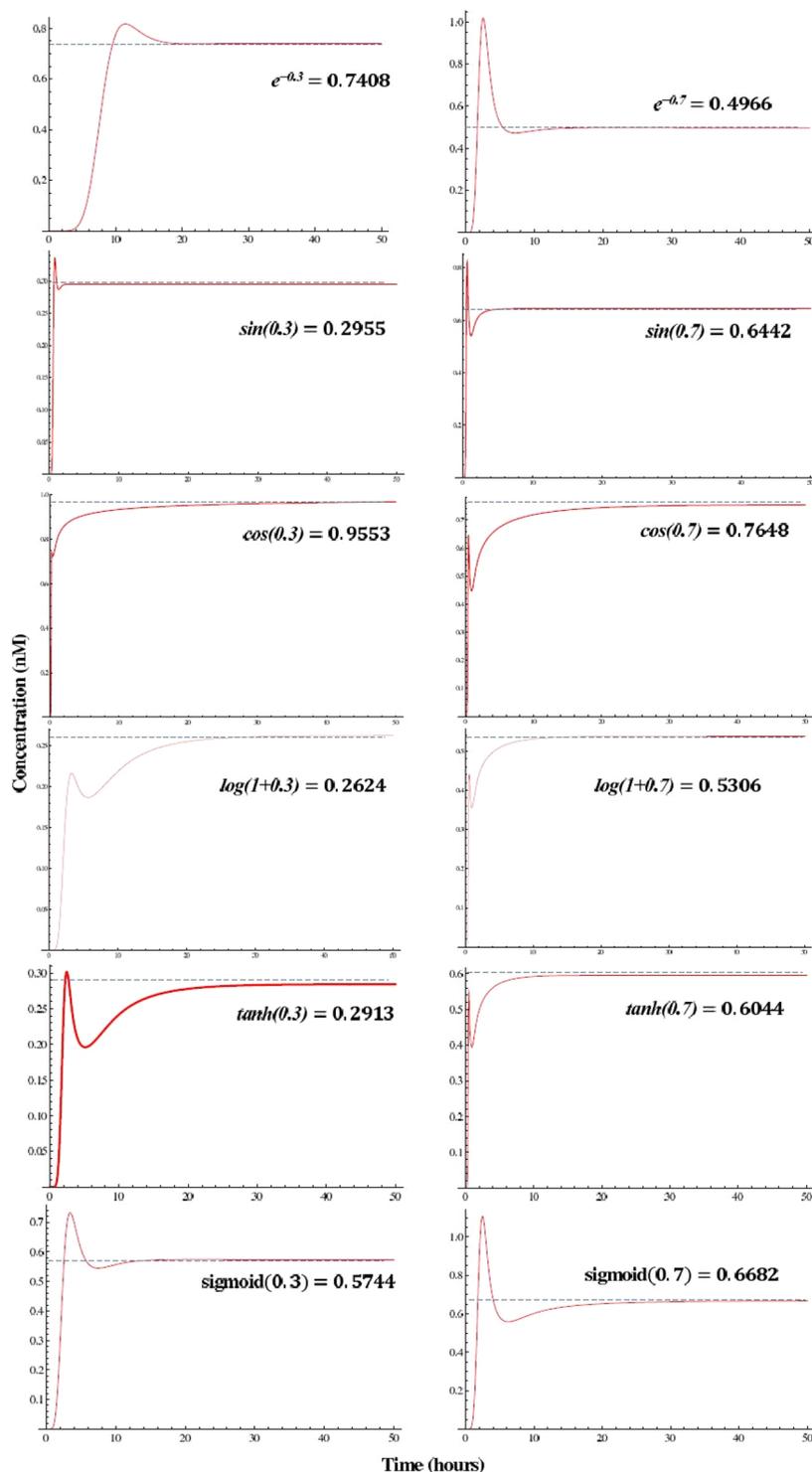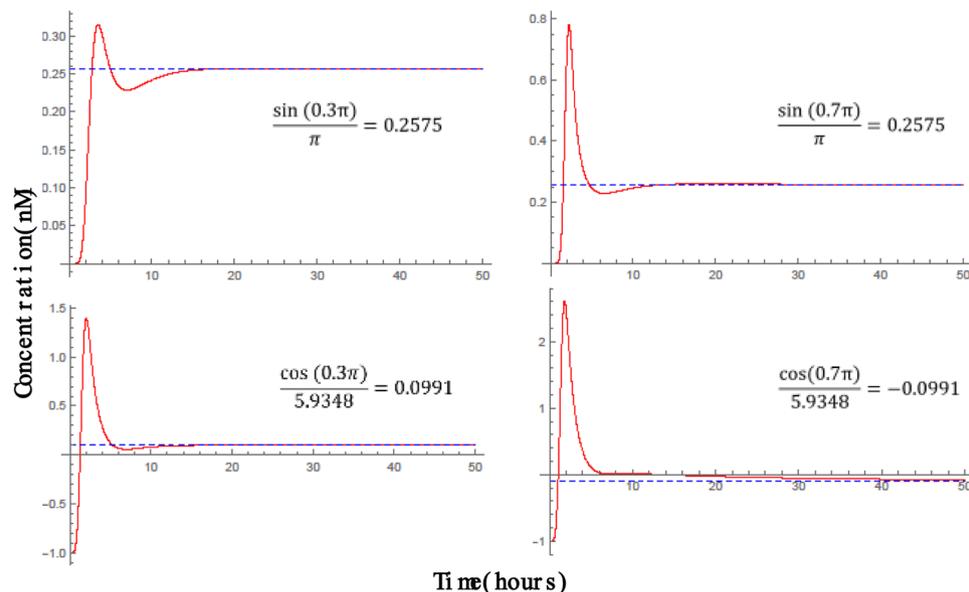
**Figure 9.** DNA simulation results. The DNA reaction kinetics for the computation of $e^{-x}$, $\sin(x)$, $\cos(x)$, $\log(1+x)$, $\tanh(x)$, and sigmoid$(x)$ for $x = 0.3$, and $x = 0.7$. Each row pertains to one function. The details for the DNA implementation are listed in Supplementary Information Section S.8.

## Designing CRNs for Computing Functions

In this section we propose a framework for designing CRNs to compute different functions. Our method is illustrated in Fig. 2.

**Methodology.**     In the proposed methodology, the functions are approximated by truncating their Maclaurin series expansions. Note that other expansion methods such as Taylor series could also be used. The approximated polynomials are then mapped into equivalent forms that can be readily implemented using Mult and `NMult` units.

**Figure 10.** DNA simulation results. The DNA reaction kinetics for the computation of $\frac{sin(\pi x)}{\pi}$, and $\frac{cos(\pi x)}{5.9348}$ for $x = 0.3$, and $x = 0.7$. For the cosine function, the simulation shows $\frac{[Y_1] - [Y_0]}{[Y_0] + [Y_1]}$, where $Y_0$ and $Y_1$ represent the output in bipolar fractional coding.

The `Mult`/`NMult` units are then mapped to CRNs. These are implemented by DNA strand-displacement reactions. We describe these steps using $f(x) = e^{-x}$ as an example.

**Step 1- Approximate the function**

The Taylor series of any function $f(x)$ that is infinitely differentiable at a point $a$ corresponds to the power series

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x - a)^n.$$

(3)

If the Taylor series is centered at zero, i.e., $a = 0$, then the series is called a Maclaurin series. As an example for $f(x) = e^{-x}$ the Maclaurin expansion is given by:

$$e^{-x} = \sum_{n=0}^{\infty} \frac{(-x)^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots$$

(4)

The series is truncated to a polynomial of degree $n$, in order to approximate the desired function. As an example if $n = 5$, i.e., the first six terms are retained, for $f(x) = e^{-x}$ we obtain

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!}.$$

(5)

**Step 2- Reformat the approximation and map it to Mult/NMult units**

As the second step, the approximating polynomials obtained in the first step are mapped into equivalent forms can be implemented using `Mult` and `NMult` units. The `Mult` and `NMult` units are analogous to AND and NAND gates in electronic design paradigm called stochastic processing. First developed by Poppelbaum[55] and Gaines[56] in the late 1960's, stochastic processing implements logical computation on random bit streams. Numbers are encoded by the probability of obtaining a one versus a zero in stream of random bits.

In this work, the `Mult` and `NMult` units perform the same operation on molecular concentrations in the unipolar fractional encoding as AND and NAND gates do, respectively, in stochastic logic. Recent work in stochastic logic[60] has shown that the form of polynomials that we generate in this step can be changed in a way that they can be mapped to a cascade of AND and NAND logic gates. The approach presented by Parhi and Liu uses the well known Horner's rule in order to map polynomials with alternating positive and negative coefficients and decreasing magnitudes to AND and NAND gates[60]. This approach can be used for Maclaurin series of the functions $e^{-x}$, $sin(x)$, $cos(x)$, $log(1 + x)$, $tanh(x)$, and sigmoid $(x)$. Note that for the trigonometric functions, the operand $x$ is in radians. We use the approach of Parhi and Liu[60] to change the form of the desired approximating polynomials and then map them to a cascade of `Mult` and `NMult` units. We briefly describe this approach.

| Function | | x = 0 | x = 0.1 | x = 0.2 | x = 0.3 | x = 0.4 | x = 0.5 | x = 0.6 | x = 0.7 | x = 0.8 | x = 0.9 | x = 1 | MSE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $e^{-x}$ | computed | 0.9568 | 0.8770 | 0.7975 | 0.7228 | 0.6609 | 0.5951 | 0.5295 | 0.4772 | 0.4300 | 0.3872 | 0.3482 | $5.02 \times 10^{-4}$ |
| | exact | 1 | 0.9048 | 0.8187 | 0.7408 | 0.6703 | 0.6065 | 0.5488 | 0.4966 | 0.4493 | 0.4066 | 0.3679 | |
| $\sin(x)$ | computed | 0 | 0.1045 | 0.2062 | 0.3043 | 0.3970 | 0.4833 | 0.5570 | 0.6261 | 0.6844 | 0.7460 | 0.7967 | $4.63 \times 10^{-4}$ |
| | exact | 0 | 0.0998 | 0.1986 | 0.2955 | 0.3894 | 0.4794 | 0.5646 | 0.64421 | 0.7173 | 0.7833 | 0.8414 | |
| $\cos(x)$ | computed | 0.9728 | 0.9757 | 0.9641 | 0.9407 | 0.9129 | 0.8671 | 0.8071 | 0.7461 | 0.6778 | 0.6029 | 0.5221 | $3.16 \times 10^{-4}$ |
| | exact | 1 | 0.9950 | 0.9800 | 0.9553 | 0.9210 | 0.8775 | 0.8253 | 0.7648 | 0.6967 | 0.6216 | 0.5403 | |
| $\log(1+x)$ | computed | 0.0090 | 0.0985 | 0.1868 | 0.2675 | 0.3410 | 0.4075 | 0.4660 | 0.5212 | 0.5707 | 0.6217 | 0.6699 | $1.8 \times 10^{-4}$ |
| | exact | 0 | 0.0953 | 0.1823 | 0.2623 | 0.3364 | 0.4054 | 0.4700 | 0.5306 | 0.5877 | 0.6418 | 0.6931 | |
| $\tanh(x)$ | computed | 0 | 0.0935 | 0.1883 | 0.2823 | 0.3701 | 0.4574 | 0.5277 | 0.5826 | 0.6246 | 0.6682 | 0.7038 | $7.35 \times 10^{-4}$ |
| | exact | 0 | 0.0996 | 0.1973 | 0.2913 | 0.3799 | 0.4621 | 0.5370 | 0.6043 | 0.6640 | 0.7162 | 0.7615 | |
| sigmoid(x) | computed | 0.5196 | 0.5453 | 0.5657 | 0.5878 | 0.6068 | 0.6212 | 0.6366 | 0.6570 | 0.6721 | 0.6906 | 0.7084 | $2.5 \times 10^{-4}$ |
| | exact | 0.5000 | 0.5250 | 0.5498 | 0.5744 | 0.5987 | 0.6225 | 0.6457 | 0.6682 | 0.6900 | 0.7109 | 0.7311 | |
| $\frac{\sin(\pi x)}{\pi}$ | computed | 0 | 0.0984 | 0.1871 | 0.2574 | 0.3023 | 0.3176 | 0.3016 | 0.2565 | 0.1931 | 0.1329 | 0.0899 | $8.48 \times 10^{-4}$ |
| | exact | 0 | 0.0984 | 0.1871 | 0.2576 | 0.3027 | 0.3183 | 0.3027 | 0.2575 | 0.1871 | 0.0984 | 0 | |
| $\frac{\cos(\pi x)}{5.9348}$ | computed | 0.1685 | 0.1602 | 0.1363 | 0.0991 | 0.0527 | 0.0030 | −0.0429 | −0.07729 | −0.0921 | −0.0852 | −0.0673 | $1.67 \times 10^{-3}$ |
| | exact | 0.1685 | 0.1603 | 0.1363 | 0.0990 | 0.0521 | 0 | −0.0521 | −0.0990 | −0.1363 | −0.1603 | −0.1685 | |

**Table 1.** Computed values of functions with the proposed CRNs compared to their exact values.

| Perceptron | TP | TN | FP | FN | Mean Square Error | |
|---|---|---|---|---|---|---|
| | | | | | Molecular | DNA |
| A | 58 | 42 | 0 | 0 | $2.0198 \times 10^{-11}$ | $3.7670 \times 10^{-7}$ |
| B | 90 | 10 | 0 | 0 | $1.2301 \times 10^{-10}$ | $1.0357 \times 10^{-6}$ |
| C | 6 | 94 | 0 | 0 | $3.9050 \times 10^{-12}$ | $9.1999 \times 10^{-8}$ |

**Table 2.** Classification accuracy and mean sqaure error values for the three perceptrons with weighted input values scaled by factor 1/32 for molecular reactions and DNA strand displacement reactions.

*Horner's rule.* Consider a polynomial $P(x)$ of degree $n$ given in its power form as

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n. \tag{6}$$

As described by Parhi and Liu[60], Eq. (6) can be rewritten as

$$P(x) = b_0(1 - b_1 x(1 - b_2 x(1 - b_3 x \ldots (1 - b_{n-1} x(1 - b_n x))))) \ldots) \tag{7}$$

where $b_0 = a_0$ and $b_i = -\frac{a_i}{a_{i-1}}$ for $i = 1, 2, \ldots, n$. Provided $0 \leq b_i \leq 1$ for $i = 0, 1, \ldots, n$, this representation can be easily mapped to a regular cascade of molecular `Mult` and `NMult` units as described by Parhi and Liu[60].

In order to guarantee $0 \leq b_i \leq 1$ the following requirements must be satisfied. Firstly, the coefficients of the original polynomial, i.e., the $a_i$'s, should be alternatively positive and negative. Secondly, the absolute values for all the coefficients, i.e., the $a_i$'s, should be less than one and decrease as the terms' orders increase. There exist several polynomials that satisfy these requirements. For example Maclaurin series expansion of $e^{-x}$, $\sin(x)$, $\cos(x)$, $\log(1+x)$, $\tanh(x)$, and sigmoid $(x)$, listed in equations (41) to (46) of the Supplementary Information, meet these requirements and can be represented using Equation (7).

Consider the following example. If we apply the Horner's rule for the fifth order Maclaurin series of $f(x) = e^{-x}$, shown in (5), we obtain

$$e^{-x} = 1 - x\left(1 - \frac{x}{2}\left(1 - \frac{x}{3}\left(1 - \frac{x}{4}\left(1 - \frac{x}{5}\right)\right)\right)\right). \tag{8}$$

Equation (8) can be implemented using `Mult` and `NMult` units as shown in Fig. 3.

Elements $E_i$ of the structure shown in Fig. 3 compute intermediate outputs $t_i$ in order to progressively compute the $e^{-x}$ function using Equation (8). What follows is the computation for each element:

$$E_1: \quad t_1 = \left(1 - \frac{x}{5}\right) \qquad E_2: \quad t_2 = \frac{1}{4}t_1 \qquad E_3: \quad t_3 = \left(1 - \frac{x}{4}t_1\right)$$

$$E_4: \quad t_4 = \frac{1}{3}t_3 \qquad E_5: \quad t_5 = \left(1 - \frac{x}{3}t_3\right) \qquad E_6: \quad t_6 = \frac{1}{2}t_5$$

$$E_7: \quad t_7 = \left(1 - \frac{x}{2}t_5\right) \qquad E_8: \quad f(x) = 1 - x t_7 = e^{-x}.$$

Figure 3 summarizes the truncated Maclaurin series, reformatted Maclaurin series using Horner's rule, and `Mult/NMult` structure for several other desired functions where the input and output are in unipolar representation. Figure 4 presents `Mult/NMult` structure for stochastic logic implementations of half-period of $\frac{sin(\pi x)}{\pi}$ and $\frac{cos(\pi x)}{5.9348}$ as presented in Parhi and Liu[60] and described by equations (9) and (10). Note that in the scaled cosine computation, the input is in unipolar representation while the output is in bipolar representation and can represent negative values. This is referred to as hybrid representation[61].

$$\frac{sin(\pi x)}{\pi} = x(1 - x^2)(1 - 0.4x^2)(1 - 0.2488x^2(1 - 0.2637x^2)). \tag{9}$$

$$\frac{cos(\pi x)}{5.9348} = \frac{4.9348}{5.9348}x^2(2 \cdot 0.4112x^2 - 1)$$
$$+ \frac{1}{5.9348}(1 - 2 \cdot 0.6676x^6(1 - 0.1762x^2(1 - 0.1097x^2))). \tag{10}$$

**Step 3- Synthesize the Chemical Reactions**

To build the CRN for computing the desired function, the next step is to synthesize the related chemical reactions for each element used in the `Mult/NMult` structure. Depending on the unit type, either the set of reactions presented in Fig. 1(a–c) is used. After designing the CRNs, the final step is to map them to DNA reactions as described in Section 4. Note that `Mult/NMult` units with more than two inputs are built by cascading two-input `Mult/NMult` units.

## Molecular Perceptron

This section describes implementation of a single-layered neural network, also called a perceptron, by molecular reactions. As shown in Fig. 5(a), the system first computes the inner product of a binary input vector and a coefficient vector as $y = \sum_{i=1}^{N} w_i x_i + w_0$; then it uses the sigmoid function to compute the final output $z$ as $z =$ sigmoid $(y)$. The stochastic sigmoid circuit shown in Fig. 5(b) was presented in[60] the reader is referred to[60] for details of the derivation. This performs a soft decision of whether the output should be close to 0 or 1. For the perceptron system that we implement, the inputs are binary, that is to say either $x_i = 0$ or $x_i = 1$, and the coefficients, i.e., $w_i$'s, are between $-1$ and 1. All multiply-add operations are implemented using bipolar `Mult` units. Since the input of the sigmoid function is between $-1$ and 1, we implement the sigmoid function using a bipolar fractional coding.

Note that prior biomolecular implementations of artificial neural networks (ANNs) have considered either hard limit or linear activation functions[50,52]. No prior publication has considered molecular ANNs using a sigmoid activation function. In this section we describe the implementation of the bipolar `MUX` unit and the bipolar `Mult` and `NMult` units.

**MUX unit.** The `MUX` unit, shown in Fig. 1(c), computes $c$ as the weighted addition of two inputs $a$ and $b$ as $c = a \times (1 - s) + b \times s$, where $0 \leq s \leq 1$. Here $a$, $b$, and $c$ can be in either the unipolar or the bipolar fractional representation while the weight $s$ must be in the unipolar representation. The set of four reactions in Fig. 1(c) describes the CRN for a `MUX` unit for both unipolar and bipolar fractional codings. Mass-action kinetic equations for both unipolar and bipolar fractional coding are discussed in Supplementary Information Section S.4.

**Bipolar Mult unit.** The bipolar `Mult` unit, shown in Fig. 1(d), computes $c$ as the multiplication of two inputs $a$ and $b$, where $a$, $b$ and $c$ are represented in bipolar fractional representation. So if $a = \frac{[A_1] - [A_0]}{[A_0] + [A_1]}$ and $b = \frac{[B_1] - [B_0]}{[B_0] + [B_1]}$ then $c = \frac{[C_1] - [C_0]}{[C_0] + [C_1]} = a \times b$. The set of four reactions in Fig. 1(d) represents the CRN for a multiplication unit in the bipolar fractional coding. In Supplementary Information Section S.3 we prove that these reactions compute $c = a \times b$.

**Bipolar NMult unit.** Analogous to the way that we obtained `NMult` from `Mult` unit in the unipolar fractional coding, if we switch $C_0$ and $C_1$ in the reactions of the bipolar Mult unit, we obtain the bipolar `NMult` unit which computes $-a \times b$. Figure 1(e) gives the corresponding set of reactions. Similar to the method we used for `Mult` unit, it is easy to show that the reactions listed in Fig. 1(e) compute $c = -a \times b$ in the bipolar fractional coding.

The proof is very similar to the bipolar `Mult` unit. Indeed, for bipolar `NMult` we just switch $C_0$ and $C_1$ meaning that in the proof for bipolar `Mult` instead of $C_1 - C_0$ in the numerator we have $C_0 - C_1$. This leads to having $c = -ab$ instead of $c = ab$.

**Hybrid sigmoid function and Perceptron with Binary Inputs.** The bipolar fractional representation can be used to implement the sigmoid function, presented in Section 2.1.1 for the unipolar fractional representation. Therefore, the function can be computed for inputs between $-1$ and 1, i.e., $-1 \leq x \leq 1$. The output of this function, however, is still in the unit interval [0, 1] and can be represented by a unipolar fractional representation. In fact, for $x \in [-1, 1]$ the corresponding output range is [0.2689, 0.7311]. In Parhi and Liu[60], it is shown that the sigmoid function using hybrid format, i.e., for bipolar input and unipolar output can be implemented by electronic stochastic logic circuits, namely, XOR and XNOR gates and multiplexers. These electronic circuits perform multiplication and weighted addition for stochastic bit streams analogous to the same operations that bipolar `Mult`, `NMult`, and `MUX` units in Fig. 1 perform for CRNs. Accordingly, we map the circuit to the cascade of proposed molecular units as shown in Fig. 5(b). The inner product can be implemented by $N$ bipolar `Mult` units

having the same output. Details for the molecular implementation of the inner product are described in Section S.5 of the Supplementary Information.

By cascading the inner product part and the sigmoid function, we can implement molecular perceptrons with binary inputs as shown in Fig. 5. Although the inner product in the standard perceptron shown in Fig. 5(a) computes $\sum_{i=1}^{N} w_i x_i$, the molecular inner product in Fig. 5(b) computes $\frac{1}{N}\sum_{i=1}^{N} w_i x_i$. We map this molecular circuit to DNA strand-displacement reactions and simulate it for $N=32$ using 32 coefficients. Three perceptrons are simulated. The 32 binary inputs are selected at random such that each bit is equally likely to be 1 or 0. It is important to note that the inputs are not constrained to be binary in the proposed methodology, but are constrained to lie between $-1$ and 1. For each perceptron, the same 100 input vectors are simulated. The input vectors are illustrated in Fig. 6(a) where the 100 columns correspond to 100 input vectors, and each column contains 32 binary values chosen at random with equal probability. The corresponding binary matrix representing the 100 input vectors is also shown in Figure S.7.1 in the Supplementary Information Section S.7. The weights of perceptrons are chosen from the set 1/2, $-1/2$, 1/4, and $-1/4$. These weights for the 3 perceptrons, denoted A, B and C, are illustrated in Fig. 6(b), and are also listed in Supplementary Section S.7. In Perceptron A, each weight occurs 8 times. In Perceptron B, the weights 1/2, $-1/2$, 1/4 and $-1/4$, occur with frequencies 10, 6, 10 and 6, respectively, In Perceptron C, the weights 1/2, $-1/2$, 1/4, and $-1/4$ occur with frequencies 6, 10, 6, and 10, respectively. In a perceptron, let the presence or absence of the input molecules be denoted by 1 or 0, and the coefficients describe the weights associated with each input, and each weighted molecule either activates or inhibits the perceptron *state* depending on whether it is positive or negative. Then Perceptron B has more molecules that activate the state whereas Perceptron C has more molecules that inhibit the state, whereas Perceptron A has equal number of molecules that either activate or inhibit the state. For equally likely binary inputs, the probabilities of the weighted sum for the Perceptrons A, B, and C, respectively, correspond to 0, 1.5 and $-1.5$. The expected sigmoid values for the three perceptrons correspond to 0.5, 0.8175, and 0.1825, respectively. Each perceptron output is classified as 1 or 0 using a threshold of 0.5. If very large number of random input vectors are simulated, we would expect the percent of input vectors classified as 1 in these three perceptrons to be 50%, 81.75% and 18.25%, respectively. For the 100 input vectors, the classification results for the three perceptrons are illustrated in Fig. 6(c–e). The number of 1's in these perceptrons correspond to 58, 90 and 6, respectively. All three molecular perceptrons achieve classification accuracy of 100%.

The simulation results in Fig. 7(a–c) illustrate the exact sigmoid values of the weighted sum of the inputs and the outputs of the molecular perceptrons that compute sigmoid of the weighted sum of the inputs scaled down by the dimension of the input vector, i.e., 32, for the Perceptrons A, B, and C, respectively. The horizontal axis in Fig. 7 represents the index of the input vector and the vertical axis shows the exact sigmoid value and the molecular sigmoid value. Although the molecular CRN outputs do not perfectly match with actual values, if we consider 0.5 as the threshold for a binary decision, the molecular perceptron classification results and the actual perceptron classifier results are the same for all 100 input vectors. Since the molecular inner product computes $y = \frac{1}{N}\sum_{i=1}^{N} w_i x_i$ instead of $y = \sum_{i=1}^{N} w_i x_i$, the amplitude for the computed output is not same as the exact value. Note that $x_i$ and $w_i$, respectively, represent the binary value of the $i^{\text{th}}$ input and its associated weight. Figure 8 shows the exact and molecular outputs of the three perceptrons that compute sigmoid of the scaled versions of the weighted inputs for the 100 input vectors. The next section describes DNA implementations of the proposed CRNs.

## DNA Implementation

Constructs in the previous sections were presented in terms of abstract CRNs. In this section, we translate our Mult/NMult circuits to DNA strand displacement (DSD) reactions. The idea of DSD reactions based on toehold mediation was first introduced by Yurke *et al.* for the construction of DNA tweezers[2]. A general method for translating CRNs to DSD reactions was proposed by Soloveichik *et al.*[6] and is illustrated in Supplementary Information Section S.8 and Figure S.8.1. That work proved that DSD reactions can closely emulate the mass-action kinetics of any CRN.

Recently Chen *et al.* showed that bimolecular reactions, such as $A + B \rightarrow C$, can be implemented by linear, double-stranded DNA complexes that are compatible with natural DNA[32]. We note that our computational units are all constructed from bimolecular reactions and so these could be implemented using the framework proposed by Chen *et al.*[32].

Using the software tool provided by Erik Winfree's group in Caltech[6] we simulate the reactions using DSD. Figures 9 and 10 show the simulation results for the functions at $x=0.3$ and $x=0.7$. Table 1 presents simulation data highlighting the accuracy of the proposed method. It lists computed values for functions at eleven equally separated points in the interval [0,1]. For each function, the computed result is reported 50 hours after the simulation starts. The table also lists the *mean square error* (MSE) at the eleven points. The error may be due to several factors: the approximation of the function with a truncated series expansion; the emulation of the related CRNs by DSD reactions; and the limited simulation time (of 50 hours for DSD reactions). As the results show, the error is less than $1 \times 10^{-3}$. For a visual comparison, Figure S.8.2 of the Supplementary Information illustrates the exact values of the functions together with their computed values.

Table 2 lists the classification accuracy of the three perceptrons simulated using DSD with results collected after 50 hours of simulations. The table also lists the mean square error values for the three perceptrons for both molecular reactions and DNA strand displacement reactions. The mean square error, MSE, is defined as:

$$MSE = \frac{1}{100}\sum_{j=1}^{100} |y(j) - \hat{y}(j)|^2$$

where $y(j) = sigmoid\left(\frac{1}{N}\sum_{i=1}^{N}w_i x_i[j]\right)$ and $\widehat{y}(j)$ is the computed value of $y(j)$ from molecular or DNA simulation, $x_i[j]$ represents the $i^{th}$ bit position of input vector $j$, and $w_i$ represents the $i^{th}$ weight. The mean square error values for molecular and DNA simulations are small as the dynamic range of the sigmoid function with scaled weighted sum of binary inputs is small. For example, $sigmoid(1.5/32)$ and $sigmoid(-1.5/32)$, respectively, correspond to 0.5117 and 0.4882. Although the DNA implementation of the perceptron achieves 100% classification accuracy in simulation, we caution that in an actual experiment the DNA perceptron may not achieve perfect classification accuracy.

## Conclusion

Although there have been numerous examples of CRNs for computing specific functions presented in the literature, as yet there has been no *systematic* way to design molecular systems to compute mathematical functions. This paper presents a systematic methodology for designing CRNs to implement complex mathematical functions robustly. The proposed method is unique in that it relies exclusively on bimolecular reactions, with no requirements on the reaction rates. According to the work of Chen *et al.*, bimolecular reactions are compatible with natural DNA[32]. This means that, the computational elements we propose here could potentially be used for *in vivo* applications. A key contribution of this paper is the ability to map any stochastic logic circuit to a molecular circuit based on fractional coding. Numerous prior papers have demonstrated stochastic logic implementations of digital filters, error control coders such as low-density parity check codes and polar codes. The proposed molecular logic gates can be used to design molecular digital filters and molecular error control coders in a straightforward manner.

This paper builds on our prior work. The computation of polynomials was presented in Salehi *et al.*[28]. In that paper we showed how arbitrary polynomials can be mapped to a CRN. Although that method could be used to compute truncated Maclaurin series of desired functions, it uses a rather complex set of chemical reactions with $m$ reactants and at least $m+1$ products, with $m \geq 2$, for polynomials of degree $m$. Implementing reactions with more than two reactants may be biologically infeasible, since this entails large complexes. In contrast, the methodology proposed in this paper requires only bimolecular reactions and so is readily implementable.

Although molecular and DNA implementations of several mathematical functions using fractional coding have been demonstrated, the proposed method suffers from numerous limitations. Use of fractional coding, inspired by stochastic logic[62–66], requires molecules to be bounded between $-1$ and 1. Thus, complete dynamic range of a function cannot be computed by the proposed method. For example, the proposed method can only compute scaled sine and cosine values. The molecular perceptron cannot compute the sigmoid value of the weighted sum of the binary inputs. This is an inherent limitation of the proposed method as the sigmoid function processes a scaled version of the weighted inputs (scaled down by the dimension of the input vector). Furthermore, the weight values are constrained to lie between $-1$ and 1. Molecular implementations of general perceptrons with arbitrary weights remains a topic for future research. In addition, further research needs to be directed towards molecular implementations of perceptrons used in inference applications as opposed to binary classification applications.

## References

1. Adelman, L. Molecular Computation of Solutions to Combinatorial Problems. *Science* **266**, 1021–1024 (1994).
2. Yurke, B., Turberfield, A. J., Mills, A. P., Simmel, F. C. & Neumann, J. L. A DNA-fuelled Molecular Machine Made of DNA. *Nature* **406**, 605–608 (2000).
3. Turberfield, A. J. *et al.* DNA Fuel for Free-running Nanomachines. *Phys. Rev. Lett.* **90**, 118102 (2003).
4. Yurke, B., Mills, B. P. & Using, D. N. A. to Power Nanostructures. Genet. Program. *Evolvable Mach.* **4**, 111 (2003).
5. Zhang, D. Y. & Winfree, E. Control of DNA Strand Displacement Kinetics Using Toehold Exchange. *J. Am. Chem. Soc.* **131**, 17303 (2009).
6. Soloveichik, D., Seelig, G. & Winfree, E. DNA as a Universal Substrate for Chemical Kinetics. Proceedings of the National Academy of Sciences, 5393–5398 (2010).
7. Gardner, T. S., Cantor, C. R. & Collins, J. J. Construction of a Genetic Toggle Switch in Escherichia Coli. *Nature* **403**, 339–342 (2000).
8. Weiss, R. *et al.* Genetic Circuit Building Blocks for Cellular Computation, Communications, and Signal Processing. *Nat. Comput.* **2**, 47–84 (2003).
9. Jiang, H., Riedel, M. D. & Parhi, K. K. Digital Logic with Molecular Reactions. *IEEE/ACM International Conference on Computer-Aided Design.* **29**, 21–31 (2013).
10. Jiang, H., Riedel, M.D., & Parhi K.K. Synchronous Sequential Computation with Molecular Reactions. ACM Design Automation Conference, (2011).
11. Beneson, Y., Gil, B., Ben-Dor, U., Adar, R. & Shapiro, E. An Autonomous Molecular Computer for Logical Control of Gene Expression. *Nature* **429**, 423–429 (2004).
12. Endy, D. Foundations for Engineering Biology. *Nature* **438**, 449–453 (2005).
13. Ramalingam, K., Tomshine, J. R., Maynard, J. A. & Kaznessis, Y. N. Forward Engineering of Synthetic Bio-logical AND Gates. *Biochem. Eng. J.* **47**, 38–47 (2009).
14. Tamsir, A., Tabor, J. J. & Voigt, C. A. Robust Multicellular Computing Using Genetically Encoded NOR Gates and Chemical 'Wires'. *Nature* **469**, 212–215 (2011).
15. Jiang, H., Riedel, M. D. & Parhi, K. K. Digital Signal processing with Molecular Reactions. *IEEE Design & Test Magazine, (Special Issue on Bio-Design Automation in SyntheticBiology)* **29**, 21–31 (2012).
16. Jiang, H., Salehi, S. A., Riedel, M. D. & Parhi, K. K. Discrete-Time Signal Processing with DNA. *American Chemical Society (ACS) SyntheticBiology* **2**, 245–254 (2013).
17. Salehi, S. A., Jiang, H., Riedel, M. D. & Parhi, K. K. Molecular Sensing and Computing Systems (Invited Paper). *IEEE Transactions on Molecular, Biological, and Multi-ScaleCommunications* **1**(3), 249–264 (2015).
18. Salehi, S. A., Riedel, M. D. & Parhi, K. K. Markov Chain Computations using Molecular Reactions. *IEEE International Conference on Digital Signal Processing* **1**(3), 249–264 (2015).
19. Salehi, S. A., Riedel, M. D. & Parhi, K. K. Asynchronous Discrete-Time Signal Processing with Molecular Reactions. *Asilomar Conference on Signals, Systems, and Computers* **1**(3), 249–264 (2014).
20. Senum, P. & Riedel, M. D. Rate-Independent Constructs for Chemical Computation. *PLoS ONE* **6**, 6 (2011).

21. Rubens, J. R., Selvaggio, G. & Lu, T. K. Synthetic Mixed-Signal Computation in Living Cells. *Nat. Commun.* **7**, 11658 (2016).
22. Nielsen, A. A. K. *et al*. Genetic Circuit Design Automation. *Science*. https://doi.org/10.1126/science.aac7341 (2016).
23. Thubagere, A. J. *et al*. Compiler-aided Systematic Construction of Large-Scale DNA Strand Displacement Circuits Using Unpurified Components. Nature. *Communications* **8**, 1038 (2017).
24. Samoilov, M., Arkin, A. & Ross., J. Signal Processing by Simple Chemical Systems. *The Journal of Physical Chemistry A* **106**, 10205–10221 (2002).
25. Thurley, K. *et al*. Reliable Encoding of Stimulus Intensities Within Random Sequences of Intracellular $Ca^{2+}$ Spikes. *Science Signaling* **7**(331), ra59, https://doi.org/10.1126/scisignal.2005237 (2014).
26. Sumit, M., Neubig, R. R., Takayama, S. & Linderman, J. J. Band-Pass Processing in a GPCR Signaling Pathway Selects for NFAT Transcription Factor Activation. *Integr. Biol.* **7**, 1378–1386 (2015).
27. Buisman, H. J., ten Eikelder, H. M. M., Hilbers, P. A. J. & Liekens, A. M. L. Computing Algebraic Functions with Biochemical Reaction Networks. *Artif. Life.* **15**(1), 5–19 (2009).
28. Salehi, S. A., Parhi, K. K. & Riedel, M. D. Chemical Reaction Networks for Computing Polynomials. *ACS Synthetic Biology Journal* **6**(1), 76–83 (2017).
29. Foo, M., Sawlekar, R. & Bates, D. G. Exploiting the Dynamic Properties of Covalent Modification Cycle for the Design of Synthetic Analog Biomolecular Circuitry. *Journal of Biological Engineering* **10**, 15 (2016).
30. Chou, C. T. Chemical Reaction Networks for Computing Logarithm. Synthetic Biology, **2**(1) (2017).
31. Gomez-Uribe, C., Verghese, G. C. & Mirny, L. A. Operating Regimes of Signaling Cycles: Statics, Dynamics, and Noise Filtering. *PLoS Comput Biol* **3**(12), e246 (2007).
32. Chen, Y. J. *et al*. Programmable Chemical Controllers Made from DNA. *Nature Nanotechnology* **8**, 755–762 (2013).
33. Bishop, C. M. Pattern Recognition and Machine Learning. Springer ISBN 8132209060, 9788132209065 (2013).
34. Alpaydin, E. Introduction to Machine Learning. 3rd Edition, MIT press (2014).
35. Xie, Z., Wroblewska, L., Prochazka, L., Weiss, R. & Benenson, Y. Multi-input RNAi-based Logic Circuit for Identification of Specific Cancer Cells. *Science* **333**, 1307–1311 (2011).
36. Li, Y. *et al*. Modular Construction of Mammalian Gene Circuits Using TALE Transcriptional Repressors. *Nat. Chem. Biol.* **11**, 207–213 (2015).
37. Miki, K. *et al*. Efficient Detection and Purification of Cell Populations Using Synthetic MicroRNA Switches. *Cell Stem Cell* **16**, 699–711 (2015).
38. Sayeg, M. K. *et al*. Rationally Designed MicroRNA-based Genetic Classifiers Target Specific Neurons in the Brain. *ACS Synth. Biol.* **4**, 788–795 (2015).
39. Mohammadi, P., Beerenwinkel, N. & Benenson, Y. Automated Design of Synthetic Cell Classifier Circuits Using a Two-Step Optimization Strategy. *Cell Systems* **4**(2), 207–218 (2017).
40. Bandyopadhyay, A., Sahu, S. & Fujita, D. Smallest Artificial Molecular Neural-net for Collective and Emergent Information Processing. *Applied physics letters* **95**(11), 113702 (2009).
41. Baum, E. B. Building an Associative Memory Vastly Larger than the Brain. *Science* **268**, 583–585 (1995).
42. Haronian, D. & Lewis, A. Elements of a Unique Bacteriorhodopsin Neural Network Architecture. *Applied optics* **30**(5), 597 (1991).
43. Huang, W. T., Chen, L. X., Lei, J. L., Luo, H. Q. & Li, N. B. Molecular Neuron: From Sensing to Logic Computation, Information Encoding, and Encryption. *Sensors and Actuators: B. Chemical* **239**, 704–710 (2017).
44. Hjelmfelt, A., Weinberger, E. D. & Ross, J. Chemical Implementation of Neural Networks and Turing Machines. *Proc. Natl Acad. Sci. USA* **88**, 10983–10987 (1991).
45. Hjelmfelt, A., Weinberger, E. D. & Ross, J. Chemical Implementation of Finite-State Machines. *Proc. Natl. Acad. Sci. USA* **89**, 383 (1992).
46. Mills, A. P. Jr, Turberfield, M., Turberfield, A. J., Yurke, B. & Platzman, P. M. Experimental Aspects of DNA Neural Network Computation. *Soft Comput.* **5**, 10–18 (2001).
47. Mills, A. P., Yurke, B. & Platzman, P. M. Article for Analog Vector Algebra Computation. *Biosystems* **52**, 175–180 (1999).
48. Laplante, J. P., Pemberton, M., Hjelmfelt, A. & Ross, J. Experiments on Pattern Recognition by Chemical Kinetics. *J. Phys. Chem.* **99**, 10063–10065 (1995).
49. Lim, H. W. *et al*. *In Vitro* Molecular Pattern Classification via DNA-Based Weighted-Sum Operation. *Biosystems* **100**, 1–7 (2010).
50. Zhang, D. Y. & Seelig, G. In DNA Computing and Molecular Programming. *Lecture Notes in Computer Science, Springer* **6518**, 176–186 (2011).
51. Lakin, M. R. & Stefanovic, D. Supervised learning in adaptive DNA strand displacement networks. *ACS Synthetic Biology* **5**(8), 885–897 (2016).
52. Qian, L. & Winfree, E. Neural Network Computation with DNA Strand Displacement Cascades. *Nature* **475**, 368–372 (2011).
53. Chen, H., Doty, D.& Soloveichik, D. Rate-Independent Computation in Continuous Chemical Reaction Networks. Conference on Innovations in Theoretical Computer Science, 313–326 (2014).
54. Gaines, B. R. Stochastic Computing. Proceedings of AFIPS spring joint computer conference, ACM, 149–156 (1967).
55. Poppelbaum, W. J., Afuso, C. and Esch. J. W. Stochastic Computing Elements and Systems. In *Proceedings of the Joint Computer Conference*, AFIPS '67 (Fall), pages 635–644, New York, NY, USA, ACM (1967).
56. Gaines, B. R. Stochastic Computing Systems. in Advances in information systems science, Springer, 37–172 (1969).
57. Qian, W. & Riedel, M. D. The Synthesis of Robust Polynomial Arithmetic with Stochastic Logic. Design Automation Conference, 648–653 (2008).
58. Qian, W., Li, X., Riedel, M. D., Bazargan, K. & Lilja, D. J. An Architecture for Fault-Tolerant Computation with Stochastic Logic. *IEEE Tran. on Comp.* **60**(1), 93–105 (2011).
59. Alaghi, A. & Hayes j., P. Survey of Stochastic Computing. *ACM Transactions on Embedded computing systems (TECS)* **12**, 92 (2013).
60. Parhi, K. K. & Liu, Y. Computing Arithmetic Functions Using Stochastic Logic by Series Expansion. *IEEE Transactions on Emerging Technologies in Computing (TETC)*. https://doi.org/10.1109/TETC.2016.2618750 (2016).
61. Parhi, K K. Analysis of Stochastic Logic Circuits in Unipolar, Bipolar and Hybrid Formats. In Circuits and Systems (ISCAS), 2017 IEEE International Symposium on, pp. 1–4 (2017).
62. Li, Peng, Lilja, D. J., Qian, W., Riedel, M. D. & Bazargan, K. Logical Computation on Stochastic Bit Streams with Linear Finite-state Machines. *Computers, IEEE Transactions on* **63**(6), 1474–1486 (2014).
63. Liu, Y. & Parhi, K. K. Computing Polynomials Using Unipolar Stochastic Logic. ACM Journal on Emerging Technologies in Computing Systems (JETC), **13**(3) (2017).
64. Liu, Y. & Parhi, K. K. Computing Hyperbolic Tangent and Sigmoid Functions Using Stochastic Logic Functions Using Stochastic Logic. Proc. 2016 Asilomar Conference on Signals, Systems and Computers, 1580–1585 (2016).
65. Salehi, S.A., Liu, Y., Riedel, M. & Parhi, K. K. Computing Polynomials with Positive Coefficients using Stochastic Logic by Double-NAND Expansion. Proc. 2017 ACM Great Lakes Symposium on VLSI (GLSVLSI), 471–474 (2017).
66. Parhi, K. K. Stochastic Logic Implementations of Polynomials with All Positive Coefficients by Expansion Methods. IEEE Transactions on Circuits and Systems II: Express Briefs, https://doi.org/10.1109/TCSII.2017.2756862 (2017).

## Acknowledgements

## Author Contributions

S.A.S., M.R. and K.P. developed fractional molecular reactions. K.P. designed the experiments. S.A.S. and X.L. simulated molecular and DNA reactions. S.A.S., M.R. and K.P. wrote the paper.

## Additional Information

**Supplementary information** accompanies this paper at https://doi.org/10.1038/s41598-018-26709-6.

**Competing Interests:** The authors declare no competing interests.

**Publisher's note:** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.