Discrete-Time Signal Processing with DNA

Hua Jiang, Sayed Ahmad Salehi, Marc D. Riedel,* and Keshab K. Parhi

Department of Electrical Engineering, University of Minnesota, Minneapolis, Minnesota 55455, United States

ABSTRACT: We present a methodology for implementing discrete-time signal processing operations, such as filtering, with molecular reactions. The reactions produce time-varying output quantities of molecules as a function of time-varying input quantities according to a functional specification. This computation is robust and independent of the reaction rates, provided that the rate constants fall within coarse categories. We describe two approaches: one entails synchronization with a clock signal, implemented through sustained chemical oscillations; the other is “self-timed” or asynchronous. We illustrate the methodology by synthesizing a simple moving-average filter, a biquad filter, and a Fast Fourier Transform (FFT). Abstract molecular reactions for these filters and transforms are translated into DNA strand displacement reactions. The computation is validated through mass-action simulations of the DNA kinetics. Although a proof of concept for the time being, molecular filters and transforms have potential applications in fields such as biochemical sensing and drug delivery.

KEYWORDS: signal processing, molecular computing, DNA computing, DNA strand displacement, sequential circuits, recurrent circuits

Electronic systems performing computation are pervasive in our modern lives. In addition to general-purpose computers, we have all manner of embedded controllers in our machines and gadgets. Electronic interfacing with biological systems is common, for instance, in biomedical devices. Such interfacing permits computational processing of biological data and actuation of responses, for instance, targeted drug delivery.

In some situations, one might want to implement computation directly with biological mechanisms. For example, one might want to implement a molecular mechanism for detecting protein markers of cancer and producing drugs targeted precisely to cancerous cells.1,2 This could be achieved through genetic engineering, using bacterial or viral delivery systems. In essence, this approach is analogous to implementing embedded controllers with purely biological components.

Whereas electronic systems perform computation in terms of voltage, i.e., energy per unit charge, one can design molecular systems to perform computation in terms of molecular concentrations, i.e., molecules per unit volume. A particularly promising strategy for such computation is based on the mechanism of DNA strand displacement.3,4

The past few decades have seen a remarkable progress in the design of integrated circuits for signal processing for applications such as audio and video.5 In abstract terms, signal processing circuits are input-output systems that transform given time-varying input signals into desired time-varying output signals. By interconnecting a few basic building blocks, namely, multipliers, adder, and delay units, complex signal processing systems can be synthesized. Signal processing operations are either “discrete-time” meaning that values are sampled and produced at specific points in time or “continuous-time” meaning that inputs are continuously transformed into outputs.

In this paper we present a methodology for implementing discrete-time signal processing operations with molecular reactions. We present two different frameworks: (1) a synchronous approach that transfers concentrations based on a molecular clock, and (2) a “self-timed” or asynchronous approach that transfers concentrations between molecular types based on the absence of other types. We illustrate our methodology with the design of a moving-average filter, a biquad filter, and an 8-point Fast Fourier Transform (FFT). We present both synchronous and asynchronous versions of each of these constructs.

For clarity, the methodology is first presented in terms of abstract molecular types; then a mapping to DNA strand displacement reactions is discussed. Critical for mapping to DNA, all of our designs consist of bimolecular reactions, that is to say, reactions with exactly two reactant molecules. In the mapping to DNA, we use experimental parameters similar to those of Soloveichik et al.4 We generate differential equations corresponding to the mass-action chemical kinetics of the DNA reactions and simulate these to characterize the behavior of the designs. Such simulations of the chemical kinetics provide a reasonably accurate prediction of the actual in vitro behavior.

Special Issue: IWBDA 2012

Received: September 17, 2012

dx.doi.org/10.1021/sb300087n | ACS Synth. Biol. XXXX, XXX, XXX—XXX

© XXXX American Chemical Society
A molecular system consists of a set of chemical reactions, each specifying a rule for how different types of molecules combine. For instance,

\[ X_1 + X_2 \xrightarrow{k} X_3 \]

specifies that one molecule of \( X_1 \) combines with one molecule of \( X_2 \) to produce one molecule of \( X_3 \) at the rate \( k \), referred to as the rate constant. If these molecular dynamics are represented in terms of mass-action kinetics, then the reaction rates are proportional to the concentrations of the participating molecular types and to the rate constant. Accordingly, for the above reaction, the rate of change in the concentrations of \( X_1 \), \( X_2 \), and \( X_3 \) is given by

\[
\frac{d[X_1]}{dt} = \frac{d[X_2]}{dt} = \frac{d[X_3]}{dt} = k[X_1][X_2]
\]

(1)

where \([X]\) denotes the concentration of the entity \( X \). Such a differential equation representation suggests immediate parallels with the systems developed in the field of electrical engineering.

Most prior schemes for molecular computation depend on specific values of the rate constants, which limits the applicability since the rate constants are not constant at all; they depend on factors such as cell volume and temperature. Accordingly, the results of the computation are not robust. We aim for robust constructs: in our methodology we require only two coarse rate categories for the rate constants, i.e., \( k_{\text{fast}} \) and \( k_{\text{slow}} \). Given reactions with any such set of rates, the computation is exact. It does not matter how fast the “fast” reactions are or how slow the “slow” reactions are — only that all fast reactions fire relatively faster than slow reactions.

### METHODOLOGY

We present two frameworks for discrete-time signal processing. The first is a synchronous framework; the second is a “self-timed” or asynchronous framework. We will first illustrate both schemes with a simple example, a moving-average filter.

The circuit diagram for the filter is shown in Figure 1. It produces an output value that is one-half the current input value plus one-half the previous value. Given a time-varying input signal \( X \), the output signal \( Y \) is a moving average, i.e., a smoother version of the input signal. Since there is no feedback in the system, it is called a finite impulse response (FIR) filter.\(^6\)

#### a. Synchronous Framework

The synchronous framework is illustrated in general terms in Figure 2. As in an electronic system, this molecular system has separate constructs for computation and for memory. A clock signal synchronizes transfers signals between the two. Here we give a simplified description of these mechanisms pertaining to the moving-average filter. Molecular reactions for generating the clock signal, for the memory constructs, and for the computational constructs are described in detail in later sections.

The delay and computation elements for the moving average filter in Figure 3 are implemented by the reactions in Figure 4.

We assume that a two-phase clock generates and consumes the molecular types \( R \) and \( B \) in alternating fashion. In the presence of \( B \), the input signal \( X \) is transferred to molecular types \( A \) and \( C \); these are both reduced to half and transferred to \( D' \) and \( Y \), respectively. In the presence of \( R \), \( D' \) is transferred to \( D \). Details regarding how the clock signals \( R \) and \( B \) are generated are given in a later section.

#### b. Asynchronous Framework

The asynchronous framework is illustrated in Figure 5. It contains no clock signal; rather it is “self-timed” in the sense that a new phase of the computation begins when an external sink removes the entire quantity of molecules \( Y \), i.e., the previous output value, and

---

3. For more details on the implementation of the moving-average filter, see the appendix.
5. A. molecular system consists of a set of chemical reactions, each specifying a rule for how different types of molecules combine. For instance,

\[ X_1 + X_2 \xrightarrow{k} X_3 \]

specifies that one molecule of \( X_1 \) combines with one molecule of \( X_2 \) to produce one molecule of \( X_3 \) at the rate \( k \), referred to as the rate constant. If these molecular dynamics are represented in terms of mass-action kinetics, then the reaction rates are proportional to the concentrations of the participating molecular types and to the rate constant. Accordingly, for the above reaction, the rate of change in the concentrations of \( X_1 \), \( X_2 \), and \( X_3 \) is given by

\[
\frac{d[X_1]}{dt} = \frac{d[X_2]}{dt} = \frac{d[X_3]}{dt} = k[X_1][X_2]
\]

(1)

where \([X]\) denotes the concentration of the entity \( X \). Such a differential equation representation suggests immediate parallels with the systems developed in the field of electrical engineering.

Most prior schemes for molecular computation depend on specific values of the rate constants, which limits the applicability since the rate constants are not constant at all; they depend on factors such as cell volume and temperature. Accordingly, the results of the computation are not robust. We aim for robust constructs: in our methodology we require only two coarse rate categories for the rate constants, i.e., \( k_{\text{fast}} \) and \( k_{\text{slow}} \). Given reactions with any such set of rates, the computation is exact. It does not matter how fast the “fast” reactions are or how slow the “slow” reactions are — only that all fast reactions fire relatively faster than slow reactions.

### METHODOLOGY

We present two frameworks for discrete-time signal processing. The first is a synchronous framework; the second is a “self-timed” or asynchronous framework. We will first illustrate both schemes with a simple example, a moving-average filter.

The circuit diagram for the filter is shown in Figure 1. It produces an output value that is one-half the current input value plus one-half the previous value. Given a time-varying input signal \( X \), the output signal \( Y \) is a moving average, i.e., a smoother version of the input signal. Since there is no feedback in the system, it is called a finite impulse response (FIR) filter.\(^6\)

#### a. Synchronous Framework

The synchronous framework is illustrated in general terms in Figure 2. As in an electronic system, this molecular system has separate constructs for computation and for memory. A clock signal synchronizes transfers signals between the two. Here we give a simplified description of these mechanisms pertaining to the moving-average filter. Molecular reactions for generating the clock signal, for the memory constructs, and for the computational constructs are described in detail in later sections.

The delay and computation elements for the moving average filter in Figure 3 are implemented by the reactions in Figure 4.

We assume that a two-phase clock generates and consumes the molecular types \( R \) and \( B \) in alternating fashion. In the presence of \( B \), the input signal \( X \) is transferred to molecular types \( A \) and \( C \); these are both reduced to half and transferred to \( D' \) and \( Y \), respectively. In the presence of \( R \), \( D' \) is transferred to \( D \). Details regarding how the clock signals \( R \) and \( B \) are generated are given in a later section.

#### b. Asynchronous Framework

The asynchronous framework is illustrated in Figure 5. It contains no clock signal; rather it is “self-timed” in the sense that a new phase of the computation begins when an external sink removes the entire quantity of molecules \( Y \), i.e., the previous output value, and
supplies a new quantity of molecules $X$, i.e., the current input value.

In this framework, the moving-average filter is implemented by the reactions in Figure 6. The molecular types corresponding to signals are $X$, $A$, $C$, $R$, $G$, $B$, and $Y$. To illustrate the design, we use colors to categorize some of these types into three categories: $Y$ and $R$ in red; $G$ in green; and $X$ and $B$ in blue. The group of the first three reactions shown in the S1 column of Figure 6 transfers the concentration of $X$ to $A$ and to $C$, a fanout operation. The concentrations of $A$ and $C$ are both reduced to half, scalar multiplication operations. The concentration of $A$ is transferred to the output $Y$, and the concentration of $C$ is transferred to $R$. The transfer to $R$ is the first phase of a delay operation. Once the signal has moved through the delay operation, the concentration of $B$ is transferred to the output $Y$. Since this concentration is combined with the concentration of $Y$ produced from $A$, this is an addition operation.

The final group of three reactions shown in the S1 column of Figure 6 implements the delay operation. The concentration of $R$ is transferred to $G$ and then to $B$. Transfers between two color categories are enabled by the absence of the third category: red goes to green in the absence of blue; green goes to blue in the absence of red; and blue goes to red in the absence of green. The reactions are enabled by molecular types $r$, $g$, and $b$ that we call absence indicators. The absence indicators ensure that the delay element takes a new value only when it has finished processing the previous value.

In the group of reactions shown in the S2 column of Figure 6, molecules of types $R'$, $G'$, and $B'$ are generated from the signal types that we color-code red, green, and blue, respectively. The concentrations of the signal types remain unchanged. (These reactions appear to violate conservation of mass. In fact, when mapped to DNA reactions, there are external “fuel” types.) Meanwhile, $R'$, $G'$, and $B'$ are consumed by external sinks, denoted by $\varnothing$. When mapped to DNA, these reactions produce the “waste” types. Here, all reactions are expressly designed to have two reactants; this permits us to map the reactions to DNA strand displacement reactions effectively. This generation/consumption process ensures that equilibria of the concentrations of $R'$, $G'$, and $B'$ reflect the total concentrations of red, green, and blue color-coded types, respectively. Accordingly, we call $R'$, $G'$, and $B'$ color concentration indicators. They serve to speed up signal transfers between color categories.

In the group of reactions shown in the S3 column of Figure 6, molecules of the absence indicator types $r$, $g$, and $b$ are generated from external sources $S_r$, $S_g$, and $S_b$. At the same time, they are consumed when $R'$, $G'$, and $B'$ are present, respectively. Therefore, the absence indicators only persist in the absence of the corresponding signals: $r$ in the absence of red types; $g$ in the absence of green types; and $b$ in the absence of blue types. They only persist in the absence of these types because otherwise “fast” reactions consume them quickly.

Finally, the group of reactions shown in the S4 column of Figure 6 provides positive feedback kinetics. These reactions effectively speed up transfers between color categories as molecules in one category are “pulled” to the next by color concentration indicators. Note that the concentration of the input $X$ is sampled in the green-to-blue phase. We assume that an external source supplies the input. The output $Y$ is produced in the blue-to-red phase. We assume that an external sink consumes these molecules.

**Clock.** In electronic circuits, a clock signal is generated by an oscillatory circuit that produces periodic voltage pulses. For a molecular clock, we choose reactions that produce sustained oscillations in terms of chemical concentrations. With such oscillations, a low concentration corresponds to a logical value of zero; a high concentration corresponds to a logical value of one.

Techniques for generating chemical oscillations are well established in the literature. Classic examples include the Lotka-Volterra, the “Brusselator”, and the Arsenite-Iodate-Chlorite systems. Unfortunately, none of these schemes are quite

### Table 1: Molecular Reactions

<table>
<thead>
<tr>
<th>Reaction</th>
<th>Rate Constant</th>
<th>Product(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g + X \xrightarrow{k_{slow}} A + C$</td>
<td>$k_{slow}$</td>
<td>$A$, $C$</td>
</tr>
<tr>
<td>$2A \xrightarrow{k_{fast}} Y$</td>
<td>$k_{fast}$</td>
<td>$Y$</td>
</tr>
<tr>
<td>$2C \xrightarrow{k_{fast}} R$</td>
<td>$k_{fast}$</td>
<td>$R$</td>
</tr>
<tr>
<td>$b + R \xrightarrow{k_{slow}} G$</td>
<td>$k_{slow}$</td>
<td>$G$</td>
</tr>
<tr>
<td>$r + G \xrightarrow{k_{slow}} B$</td>
<td>$k_{slow}$</td>
<td>$B$</td>
</tr>
<tr>
<td>$g + B \xrightarrow{k_{slow}} Y$</td>
<td>$k_{slow}$</td>
<td>$Y$</td>
</tr>
</tbody>
</table>

**Figure 6.** Set of molecular reactions for the asynchronous implementation of the moving-average filter.
suitable for synchronous sequential computation: we require that the clock signal be symmetrical, with abrupt transitions between the phases. Here, we present a new design for an n-phase chemical oscillator \((n \geq 3)\). The clock phases are represented by molecular types \(P_1, P_2, ..., P_n\). First consider the reactions

\[
2S_1 \xrightarrow{k_{slow}} a_1 + 2S_{1'}, \quad 2S_2 \xrightarrow{k_{slow}} a_2 + 2S_{2'}, \quad ..., \\
2S_n \xrightarrow{k_{slow}} a_n + 2S_n
\]

(3)

\[
P_1 + a_1 \xrightarrow{k_{trans}} P_1, \quad P_2 + a_2 \xrightarrow{k_{trans}} P_2, \quad ..., \quad P_n + a_n \xrightarrow{k_{trans}} P_n
\]

(4)

In reactions 3, the molecular types \(a_1, a_2, ..., a_n\) are generated slowly and constantly, from source types \(S_1, S_2, ..., S_n\) whose concentrations do not change with the reactions. Here, all reactions are expressly designed to have two reactants; as discussed above, this permits us to map the reaction to DNA strand displacement reactions effectively.

In reactions 4, the types \(P_1, P_2, ..., P_n\) quickly consume the types \(a_1, a_2, ..., a_n\) respectively. Call \(P_i, P_{i'}, P_i\) the phase signals and \(a_i, a_{i'}, a_i\) the absence indicators. The latter are only present in the absence of the former. The reactions

\[
P_1 + a_n \xrightarrow{k_{trans}} P_1, \quad P_2 + a_1 \xrightarrow{k_{trans}} P_2, \quad ..., \quad P_n + a_{n-1} \xrightarrow{k_{trans}} P_1
\]

(5)

transfer one phase signal to another, in the absence of the previous one. The essential aspect is that, within the \(P_1, P_2, ..., P_n\) sequence, the full quantity of the preceding type is transferred to the current type before the transfer to the succeeding type begins. To achieve sustained oscillation, we introduce positive feedback. This is provided by the reactions

\[
2P_1 \xrightarrow{k_{slow}} I_1, \quad 2I_1 \xrightarrow{k_{slow}} 4P_1, \quad P_1 + I_1 \xrightarrow{k_{slow}} 3P_1
\]

\[
2P_2 \xrightarrow{k_{slow}} I_2, \quad 2I_2 \xrightarrow{k_{slow}} 4P_2, \quad P_2 + I_2 \xrightarrow{k_{slow}} 3P_2
\]

\[
\vdots
\]

\[
2P_n \xrightarrow{k_{slow}} I_n, \quad 2I_n \xrightarrow{k_{slow}} 4P_n, \quad P_{n-1} + I_n \xrightarrow{k_{slow}} 3P_n
\]

(6)

Consider the first three reactions. Two molecules of \(P_1\) combine with one molecule of \(P_2\) to produce three molecules of \(P_1\). The first step in this process is reversible: two molecules of \(P_1\) can combine, but in the absence of any molecules of \(P_n\) the combined form will dissociate back into \(P_1\). So, in the absence of \(P_n\), the quantity of \(P_1\) will not change much. Due to the first reaction, the transfer will occur at a rate that is superlinear in the quantity of \(P_1\); this speeds up the transfer and so provides positive feedback. Suppose that the initial quantity of \(P_1\) is set to some non-zero amount, and the initial quantity of the other types is set to zero. We will get an oscillation among the quantities of \(P_1, P_2, ..., P_n\).

One requirement for a clock in synchronous computation is that different clock phases should not overlap. In this paper we use a two-phase clock for synchronous structures: concentrations of molecular types representing clock phase “0” and clock phase “1” should not be present at the same time. To this end, we choose two nonadjacent phases, \(P_1\) and \(P_2\) in a four-phase oscillator, as the clock phases. For a clearer illustration, we use \(R(ed)\) to denote \(P_1\) and \(B(lue)\) to denote \(P_2\).

Our scheme for chemical oscillation works well. Figure 7 shows the concentrations of \(R\) and \(B\) as a function of time, obtained through differential equation simulations of the reactions 3, 4, 5 and 6. It may be noted that the two phases \(R\) and \(B\) are essentially non-overlapping.

## DELAY ELEMENTS

### a. Synchronous Delay Elements

To implement sequential computation, we must store and transfer signals across clock cycles. In electronic systems, storage is typically implemented with flip-flops. In our molecular system, we implement storage and transfer using a two-phase protocol, synchronized on phases of our clock. As Figure 8 illustrates,

![Figure 8. A delay element in the synchronous scheme.](image)

every memory unit is assigned two molecular types \(D'_{i}\) and \(D_i\). Here \(D'_{i}\) is the first stage and \(D_i\) the second. The blue phase reactions are

\[
B + D_i \xrightarrow{k_{trans}} \text{computations} + B
\]

\[
\text{computations} \xrightarrow{k_{trans}} D'_{i}
\]

(7)

Every delay unit releases the signal it stores in its second stage \(D_{i}\). The released signal is operated on by reactions in computational modules. These generate results and push them into the first stages of succeeding memory units. Note that \(D'_{i}\) molecules will be the first stage of any succeeding memory unit along the signal path. The red phase reactions are

\[
R + D'_{i} \xrightarrow{k_{trans}} D_{i} + R
\]

(8)

A delay unit transfers the signal it stores in \(D'_{i}\) to \(D_{i}\) preparing for the next cycle. It is the equivalent of a delay (D) flip-flop in an electronic system.

### b. Asynchronous Delay Elements

In our asynchronous scheme, we implement delay elements by transferring concentrations between molecular types based on the absence of any signal molecules.
of other types. Each delay element $DE_i$ is assigned three molecular types $R(\text{ed})$, $G(\text{reen})$, and $B(\text{lue})$. Let $b$ be an absence indicator for $B$. The element is implemented by the following reactions:

- **Phase 1** reactions: $b + R_i \xrightarrow{k_{\text{fast}}} G_i$ and $G_i \xrightarrow{k_{\text{fast}}} G_i$.
- **Phase 2** reactions: $b + G_i \xrightarrow{k_{\text{slow}}} B_i$, $B_i + G_i \xrightarrow{k_{\text{slow}}} B_i$.
- **Phase 3** reactions: $g + B_i \xrightarrow{k_{\text{slow}}} $ computations, $R_i + B_i \xrightarrow{k_{\text{slow}}} $ computations.

A computation cycle, in which an input value is accepted and an output value is computed, completes in three phases. The input $X$ is injected in phase 2 and the output $Y$ is collect in phase 1. In each phase the signals are transferred from molecular types in one color category to the next. Computations, including scalar multiplication, addition, and fanout, are carried out in phase 3, during the transfer from blue to red:

$\text{computations} \xrightarrow{k_{\text{fast}}} R_i$

This is illustrated in Figure 9(i).

![Figure 9](image)

**Figure 9.** (i) Implementing delay elements using the 3-phase asynchronous scheme. (ii) Cascaded delay elements implemented using the asynchronous scheme.

We use the notation $\text{Delay}(R_i,G_i,B_i,\{\text{output\_list}\})$ to represent the above three reactions. Here, $\{\text{output\_list}\}$ is a list of molecular types that $B_i$ should be transferred to during phase 3. The input $X$ is labeled blue, therefore, reactions

$g + X \xrightarrow{k_{\text{slow}}} $ computations, $R_i + X \xrightarrow{k_{\text{slow}}} $ computations

also fire in phase 3. We use $\text{Input}(X,\{\text{output\_list}\})$ to represent these reactions.

The computation reactions fire much faster than the transfer reactions, so molecules of $R_i$ are immediately produced from molecules of $B_i$. Thus, reactions in phase 3 effectively transfer blue signals to red signals. Note that $R_i$ produced in phase 3 will be a red type of any succeeding delay element $DE_i$ along the signal path from $DE_i$. In Figure 9ii, $R_1$ and $R_2$ are red; $G_1$ and $G_2$ are green; $B_1$ and $B_2$ are blue. The multiplier is the computation that occurs between the delay elements. $DE_i$ is a succeeding delay element of $DE_i$, so molecules of $B_i$ are transferred to $R_i$ in phase 3. For each delay element, the color concentration types $R'$, $G'$, and $B'$ are generated and consumed in the following reactions:

\[
\begin{array}{l}
2R_i \xrightarrow{k_{\text{fast}}} 2R_i + R', \ 2Y \xrightarrow{k_{\text{fast}}} 2Y + R', \ 2G_i \xrightarrow{k_{\text{fast}}} 2G_i + G', \\
2B_i \xrightarrow{k_{\text{slow}}} 2B_i + B', \ 2X \xrightarrow{k_{\text{slow}}} 2X + B', \ 2R' \xrightarrow{k_{\text{slow}}} \emptyset, \ 2G' \xrightarrow{k_{\text{slow}}} \emptyset, \ 2B' \xrightarrow{k_{\text{slow}}} \emptyset.
\end{array}
\]

(9)

So molecules of $R'$, $G'$, and $B'$ are generated by types of the corresponding color categories; they are consumed by external sinks. The equilibrium levels of these three types are determined by total concentrations of all the red types, blue types, and green types, respectively. Note that these reactions are in the “fast” category, since the color concentration types cannot lag the signal types. We use $\text{Conc}(R_{(\text{red\_type\_list})})$, $\text{Conc}(G_{(\text{green\_type\_list})})$, $\text{Conc}(B_{(\text{blue\_type\_list})})$ to represent Reactions 9. For example, $\text{Conc}(R_{(R,Y)})$ represents $2R_i \xrightarrow{k_{\text{slow}}} 2R_i + R', \ 2Y \xrightarrow{k_{\text{slow}}} 2Y + R', \ 2R_{(R,Y)} \xrightarrow{k_{\text{slow}}} \emptyset$. For delay elements, the following reactions generate the absence indicator types $r$, $g$, and $b$:

\[
\begin{array}{l}
2S_r \xrightarrow{k_{\text{slow}}} 2S_r + r, \ R' + r \xrightarrow{k_{\text{slow}}} R', \ 2S_g \xrightarrow{k_{\text{slow}}} 2S_g + g, \\
G' + g \xrightarrow{k_{\text{slow}}} G', \ 2S_b \xrightarrow{k_{\text{slow}}} 2S_b + b, \ B' + b \xrightarrow{k_{\text{slow}}} B'
\end{array}
\]

(10)

We use the notation $\text{Abs}(S_r,S_g,S_b,r,g,b,R',G',B')$ to represent these reactions. Here $r$, $g$, and $b$ are continually and slowly generated. However, they persist only in the absence of the corresponding color-coded types, since they are quickly consumed by $R'$, $G'$, and $B'$, respectively, if these are present. All transfers are initiated by absence indicators and then sped up by the color concentration indicators. The transfers initiated by the absence indicators are slow, and those initiated by the color concentration indicators are fast. This mitigates against “leakage”, e.g., some transferring from $G_i$ to $B_i$ before all of transferring from $R_i$ to $G_i$ is complete.

Note that, in any system, there are only three color concentration indicators $(R', G', B')$ and three absence indicators $(r, g, b)$, regardless of the number of delay elements. These types help enable and speed up signal transfers for all reactions in the corresponding color categories. Through these common indicators, the corresponding phases of all delay elements are synchronized: all of the delay elements must wait for each to complete its current phase before they can move to the next phase.

### Computations

General signal processing operations can be specified in terms of three basic modules: scalar multiplication, addition, and delay elements. Above, we discussed delay elements in detail. Scalar multiplication and addition are much simpler.

**Scalar multiplication** performs the operation

$$y = \frac{c_2}{c_1} x$$

where $c_1$ and $c_2$ are constants. This operation is implemented by choosing reactions with the appropriate coefficients:

$$c_2 X \rightarrow c_1 Y$$

Every time this reaction fires, $c_1$ molecules of $X$ get transferred to $c_2$ molecules of $Y$. Once the reaction has fired to completion,
an in the former concentration of $X$ the synchronous and asynchronous schemes. Implementations of multiplication and addition are the same in $Y$ concentration of $f_i$.

Once both of these reactions have fired to completion, the concentration of $Y$ will be the former concentration of $X_2$. It should be noted that implementations of multiplication and addition are the same in the synchronous and asynchronous schemes.

**Biquad Filter**

We illustrate the general design method with a second detailed example, a biquad filter. The circuit diagram for the filter is shown in Figure 10. Since this design has feedback, it is called an infinite impulse response (IIR) filter. Biquad filters are basic building blocks of modern signal processing systems. Highly stable, high-order filters can be implemented by cascaded biquad blocks.

**a. Synchronous Biquad Filter.** The following reactions provide a synchronous implementation of the biquad filter.

\[
B + X \xrightarrow{k_{\text{int}}} B + A + D'_1 \\
B + D_1 \xrightarrow{k_{\text{int}}} B + F + C + D'_2 \\
B + D_2 \xrightarrow{k_{\text{int}}} B + H + E \\
2A \xrightarrow{k_{\text{int}}} A_2, 2A_2 \xrightarrow{k_{\text{int}}} A_4, 2A_4 \xrightarrow{k_{\text{int}}} Y, \\
2C \xrightarrow{k_{\text{int}}} C_2, 2C_2 \xrightarrow{k_{\text{int}}} C_4, 2C_4 \xrightarrow{k_{\text{int}}} Y, \\
2E \xrightarrow{k_{\text{int}}} E_2, 2E_2 \xrightarrow{k_{\text{int}}} E_4, 2E_4 \xrightarrow{k_{\text{int}}} Y, \\
2F \xrightarrow{k_{\text{int}}} F_2, 2F_2 \xrightarrow{k_{\text{int}}} F_4, 2F_4 \xrightarrow{k_{\text{int}}} X, \\
2H \xrightarrow{k_{\text{int}}} H_2, 2H_2 \xrightarrow{k_{\text{int}}} H_4, 2H_4 \xrightarrow{k_{\text{int}}} X \\
R + D'_1 \xrightarrow{k_{\text{int}}} R + D_1 \\
R + D'_2 \xrightarrow{k_{\text{int}}} R + D_2
\]

In the set of reactions 11, with clock phase $B$, signal $X$ is transferred to $A$ and $D'_1$; $D_1$ is transferred to $C$, $F$, and $D'_2$; $D_2$ is transferred to $H$ and $E$. In the set of reactions 12, temporary signals $A$, $C$, $E$, $F$, and $H$ are multiplied by $(1/8)$ and transferred to either $X$ or $Y$. In the set of reactions 13, with clock phase $R$, signal transfers inside memory units take place.

**b. Asynchronous Biquad Filter.** An asynchronous implementation of the biquad filter is shown in Figure 10i. It is realized by the following reactions:

- Delay elements: $\text{Delay}(R_1,G_1,B_1,\{R_2,F,C\}), \text{Delay}(R_2,G_2,B_2,\{H,E\})$.
- System input: $\text{Input}(X,\{R,A\})$.
- Scalar multiplications: $\text{Mult}(A,Y,8,1),(C,Y,8,1),(E,Y,8,1),\ldots$.
- Concentration indicators: $\text{Conc}(R',\{R_1,R_2,Y\}), \text{Conc}(G',\{G_1,G_2\}), \text{Conc}(B',\{B_1,B_2,X\})$.
- Absence indicators: $\text{Abs}(S_1,S_2,S_3,r_g,b_r,G',B')$.

**8-Point RFFT**

The $N$-point Discrete Fourier Transform (DFT)\(^8\) is defined mathematically as

\[
X[k] = \sum_{n=0}^{N-1} x(n) e^{-2\pi ijn/N}, k = 0, 1, \ldots, N-1
\]

(here $i$ is the imaginary unit).

The DFT represents decomposition of a sequence of values $x(n)$ into components of different frequencies, $X[k]$. In essence, it transforms a signal from the time domain to the spectral or frequency domain. Such a transform is useful for a wide variety of applications; indeed, it is ubiquitous in communication systems.

The so-called Fast Fourier Transform (FFT) algorithm is a computationally less expensive alternative to the DFT. In the FFT, the input sequence is first divided to two subsequences: the even and odd indexed samples. Replacing the original sequence by a sum of such shorter sequences simplifies the computation.
transform. Repeating this, the entire transform can be converted to a set of 2-point DFT elements called “butterfly” (BF) units. A BF unit is shown in Figure 13i.

It is easy to show that for real valued signals, we have

\[ X[k] = X^*[N - k] \]  

(15)

So it is not necessary to compute all of the FFT coefficients. By eliminating the computations of \(((N/2) - 1)\) conjugate-symmetric outputs and separating them into real and imaginary parts, one obtains the structure shown in Figure 11, called a real-valued FFT (RFFT).

The RFFT architecture is implemented using computation modules and delay units, as well as the elements shown in Figure 13. Because concentration of a molecular type cannot be negative, we use one molecular type for positive and another one for negative part of each signal. Finally these two parts cancel out each other, and the one that has larger concentration determines the sign of that signal. For example, if \(x_p\) and \(x_n\) represent positive and negative part of signal \(x\), then the following reactions performs the positive/negative cancellation:

\[ x_p + x_n \xrightarrow{k_{\text{mix}}} \emptyset \]  

(16)

(Here \(\emptyset\) denotes a waste product that we do not track.) For simplicity, in Figure 13 we present only reactions related to positive part. The reactions for negative part are analogous.

\section*{SIMULATION RESULTS}

Mapping to DNA Strand Displacement. Given a specification of an abstract set of molecular reactions that implements the requisite computation, the next step is to map it to specific molecular reactions. We describe a mapping to DNA strand displacement reactions. The reader is referred to Soloveichik et al. for a detailed discussion of this mechanism.\(^4\) The following is a simple example.

Consider the DNA strand displacement reaction shown in Figure 14. Here a single strand of DNA \(R_1\) replaces the top strand of a double-strand DNA \(L_1\) this generates a double strand \(H\) and a single strand \(B\). (This reaction is reversible.) One of the top strands of the double strand \(H\) can be replaced by a single strand \(R_2\) generating a single strand \(O\). Then \(O\) replaces the top strand of \(T\), releasing \(P\). (Note that the strands \(L_1, G_1\), and \(T\) are “fuel” sources. It is assumed that there is an abundant source of these; the concentrations do not matter.) The signals are the concentrations of \(R_1, R_2,\) and \(P\). This sequence of strand displacements implements the abstract chemical reaction: \(R_1 + R_2 \rightarrow P\).

**Simulation Results.** To validate our designs for moving-average filter, the biquad filter, and the Fast Fourier Transform, we map their bimolecular reactions to DNA strand displacement reactions, using the method discussed in Soloveichik et al.\(^4\) We generate the corresponding system of kinetic differential equations using similar parameters to Soloveichik et al. We simulate the differential equations to characterize the behavior of the system.

The initial concentrations of auxiliary complexes is set to \(C_{\text{max}} = 10^{-5} \text{ M}\), and the maximum strand displacement rate constant is \(q_{\text{max}} = 10^6 \text{ M}^{-1} \text{ s}^{-1}\). The rate constant for the “slow” reactions is set to \(k_{\text{slow}} = 5.56 \times 10^6 \text{ M}^{-1} \text{ s}^{-1}\). For “fast”
reactions it is set to $k_{\text{fast}} = 3 \times k_{\text{slow}}$. The initial concentrations of $S_r$, $S_p$, and $S_b$ are set to 1 nM.

The simulation results for the synchronous and asynchronous moving-average filters are shown in Figures 15 and 16, respectively. The input is a time-varying signal concentration $X$ with both high-frequency and low-frequency components. The output is a time-varying signal concentration $Y$. Molecules of $X$ are injected and molecules of $Y$ are collected from the systems every 72 h for the synchronous scheme and every 20 h for the asynchronous scheme. The figure shows the theoretical output, i.e., an exact calculation of filtering, as well as simulation results. As expected, the system performs “notch” filtering. The simulation results show that even for a ratio $\lambda = k_{\text{fast}}/k_{\text{slow}}$ as low as 3, the systems perform well. In fact, in experimental implementations of DNA strand displacement systems, a ratio $\lambda$ greater than 1000 is readily achievable. When $\lambda$ is close to 1, i.e., fast reactions are not much faster than slow reactions, the concentrations of the absence indicators $r$, $g$, and $b$ are high.

Figure 13. Abstract molecular reactions for different elements in Figure 12: (i) BF; (ii) multiplication by $\sqrt{2}$; (iii) multiplexer.

Figure 14. Example of DNA strand displacement.

Figure 15. Simulation results for the DNA implementation of the synchronous moving average filter.

Figure 16. Simulation results for the DNA implementation of the asynchronous moving average filter.
even when the concentrations of $R'$, $G'$, or $B'$ are high. Also, the computational modules slow down. Accordingly, the accuracy of the computation degrades.

For the 8-point RFFT we choose 1, 1.5, 1, 0, 1, 1.5, 1, 0 as the input sequence. Because the RFFT architecture has 4 parallel inputs, the input sequence is injected to the systems in two stages. Concentrations for the inputs in the first and second stages and their corresponding outputs are tabulated in Table 1.

Figures 19 and 20 show the simulation results for the synchronous and asynchronous biquad filters, respectively. Table 2 compares the simulation results of the three operations, namely, the moving-average filter, the biquad filter, and the RFFT transform, in both the synchronous and asynchronous frameworks. The error in this table is computed as the difference between the output value obtained by simulation, $O_s$, and the theoretical output, $O_t$. On the one hand, Table 2 shows that the synchronous scheme has lower error. On the other hand, it is much slower than the asynchronous scheme.

### Table 1. Input and Output Order for the 8-Point RFFT Structure

<table>
<thead>
<tr>
<th>stage</th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>second</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 2. Comparison for Moving-Average, Biquad, and FFT Designs, Synchronous and Asynchronous

| system   | scheme | no. of reactants | no. of reactions | calculation time (h) | % error $(\frac{|O_s - O_t|}{O_t} \times 100)$ |
|----------|--------|------------------|------------------|----------------------|---------------------------------|
| moving average  | synchronous | 22 | 29 | 80 | 4.21 |
| biquad    | asynchronous | 16 | 24 | 20 | 5.67 |
| RFFT      | synchronous | 37 | 44 | 80 | 8.63 |
|           | asynchronous | 32 | 46 | 50 | 12.79 |

### CONCLUSION

There is a rich body of research discussing the implementation of computation and logical functions with genetic regulatory elements, and yet, despite concerted efforts by the synthetic biology community, progress in the design of genetic circuits seem to have stalled at the level of circuits with perhaps 7–15 components. In vivo engineering of such circuits is hard work, fraught with experimental difficulties.

In contrast, in vitro molecular computation with DNA strand displacement is following a Moore’s Law-like trajectory in the scaling of its complexity. Indeed, Erik Winfree, Lulu Qian, and their students at Caltech can routinely go from a design of a DNA circuit with several hundred components to experimental validation of it in a matter of weeks.

One of the great successes of electronic circuit design has been in abstracting and scaling the design problem. The physical behavior of transistors is understood in terms of differential equations, say, with models found in tools such as SPICE. However, the design of circuits occurs at more abstract levels, in terms of switches, gates, and modules.
Research in molecular computation could benefit from this hierarchical approach.

Although pertaining to biology, the contributions of this paper are not experimental nor empirical; rather they are constructive and conceptual. We have presented two general methodologies for implementing signal processing with molecular reactions. One of our methodologies uses clocking to implement synchronous computation. The other one is self-timed and asynchronous. In both cases, the computation itself is essentially rate-independent, meaning that within a broad range of values for the rate constants, the computation is exact and independent of the specific rates.

Certainly, engineering complex new reaction mechanisms in any experimental domain is a formidable task; for in vivo systems, there are likely to be many experimental constraints on the choice of reactions. However, the techniques that we have presented here are robust and scalable. Tackling general problems in molecular computation this way could be transformative for applications such as drug delivery and metabolic engineering.

■ AUTHOR INFORMATION

Corresponding Author
*E-mail: mriedel@umn.edu.

Notes
The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This research is supported, in parts, by the NSF CAREER Award 0845650, NSF CCF 0946601, and NSF CCF 1117168.

■ REFERENCES


