Digital Signal Processing with Molecular Reactions

Hua Jiang, Marc D. Riedel and Keshab K. Parhi

Department of Electrical and Computer Engineering
University of Minnesota
200 Union St. S.E., Minneapolis, MN 55455
{hua, mriedel, parhi}@umn.edu

Abstract

This paper presents a methodology for implementing digital signal processing (DSP) operations such as filtering with molecular reactions. Molecular reactions that produce time-varying output quantities of molecules as a function of time-varying input quantities are designed according to a DSP specification. Unlike all previous schemes for molecular computation, the methodology produces designs that are dependent only on coarse rate categories for the reactions ("fast" and "slow"). Given such categories, the computation is exact and independent of the specific reaction rates.

The methodology is illustrated with the design of a simple moving-average filter as well as a more complex biquad filter. Both designs are translated into DNA strand displacement reactions. The designs are validated through transient stochastic simulations of the chemical kinetics at the DNA reactions level. Although conceptual for the time being, the proposed methodology has potential applications in domains of synthetic biology such as biochemical sensing and drug delivery.

1. Introduction

The past few decades have seen remarkable progress in the design of integrated circuits for digital signal processing (DSP) for applications such as audio and video processing [1]. A typical signal processing operation produces an output signal by filtering or transforming an input signal. Examples are smoothing a signal with a moving-average filter and performing a fast Fourier transform (FFT). We aim to apply and extend this expertise to the domain of molecular computation.

Just as electronic systems implement computation in terms of voltage (*energy per unit charge*), molecular systems can compute in terms of molecular concentrations (*molecules per unit volume*). A variety of computational constructs have been proposed [2], [3], [4], [5]. Our prior work includes constructs for *logic*, *memory*, and *arithmetic* [6], [7].

The impetus is not to create computational systems *per se*. Molecular computation will never compete with conventional computers made of silicon integrated circuits for tasks such as number crunching. Rather, the ultimate goal is to create "embedded controllers" – cells and viruses that are engineered to perform useful molecular computation *in situ* where it is needed, for instance for drug delivery and biochemical sensing. Exciting work in this vein includes [8], [9], [10].

The design of effective embedded controllers will entail computational processing, performed in terms of molecular reactions. Such computational processing could take the form: "If molecular type X is present, produce molecular type Y" where X is, say a protein marker of cancer and Y is a chemotherapy drug. Or it could be more complicated: "If X is present and Y is not present, or vice-versa, then produce Z" (i.e., an exclusive-or function). Or it could be time-varying computation: "Produce an output quantity Y that changes as X changes, but more smoothly" (i.e., low-pass filtering).

This paper discusses techniques for implementing DSP operations such as filtering with molecular reactions. From a DSP specification, we demonstrate how to synthesize molecular reactions that produce time-varying output concentrations of molecules as a function of time-varying input concentrations. We implement the operations through a "self-timed" protocol that transfers concentrations between molecular types based on the absence of other types. We illustrate our methodology with the design of a simple moving average filter as well as a more complex biquad filter.

2. Computational Model

A molecular system consists of a set of chemical reactions, each specifying a rule for how types of molecules combine. For instance,

$$X_1 + X_2 \xrightarrow{k} X_3, \tag{1}$$

specifies that one molecule of X_1 combines with one molecule of X_2 to produce one molecule of X_3. The value k is called the *rate constant*. We model the molecular dynamics in terms of *mass-action kinetics*: reaction rates are proportional to (1) the concentrations of the participating molecular types; and (2) the rate constant. Accordingly, for the reaction above, the rate of change in the concentrations of X_1, X_2 and X_3 is

$$-\frac{d[X_1]}{dt} = -\frac{d[X_2]}{dt} = \frac{d[X_3]}{dt} = k[X_1][X_2], \tag{2}$$

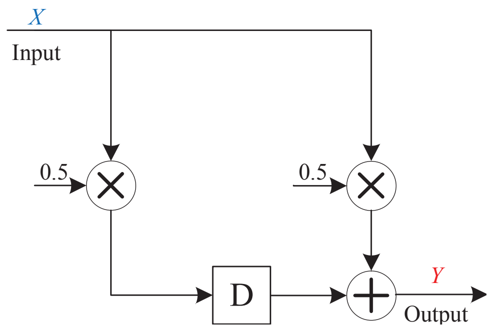
(here $[\cdot]$ denotes concentration).

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.

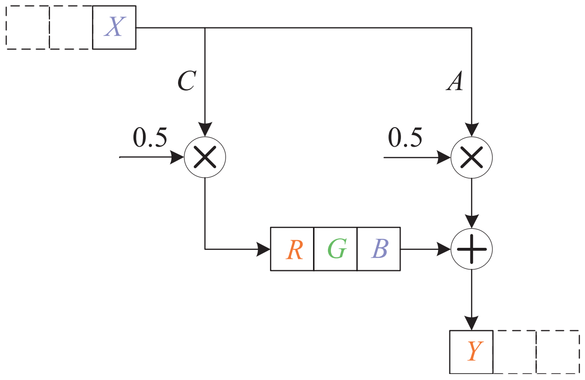
We aim for robust constructs: in our methodology we require only two coarse values for the rate constants, i.e., k_{fast} and k_{slow}. Given such coarse values for these constants, the computation is exact. It does not matter how fast the "fast" reactions are or how slow the "slow" reactions are – only that all fast reactions fire relatively faster than slow reactions.

We target DNA-based computation via strand displacement as our experimental chassis. Our contribution can be positioned as the *front-end* of a design flow. The output of our methodology is a set of abstract molecular reactions. Soloveichik *et al.* have developed a "DNA assembler" [11]; this constitutes the *back-end*. They have shown that the kinetics of molecular reactions can be *emulated* with DNA strand displacements. Reaction rates are controlled by designing sequences with different binding strengths. The binding strengths are controlled by the length and sequence composition of "toehold" sequences of DNA. Different reaction rates can be easily realized by designing DNA strands with different toehold lengths [11]. They have shown that that *any* system consisting of unimolecular reactions (i.e., those with a single reactant) and bimolecular reactions (i.e., those with two reactants) can be emulated by such DNA strand displacement reactions.

In DNA strand displacement systems, the reaction rates for unimolecular reactions and bimolecular reactions are different. The rates for unimolecular reactions depend on the initial concentration of auxiliary complexes. For design simplicity, all of our designs consist of bimolecular reactions. We map these to DNA strand displacement reactions, using similar experimental parameters as [11]. We generate differential equations corresponding to the DNA reactions and obtain transient solutions. Such simulations of the chemical kinetics provide a reasonably accurate prediction of the actual *in vitro* behavior.



(a) Schematic of the two-tap moving average filter.



(b) The filter in a three-phase configuration.

Figure 1: A two-tap moving average filter.

3. Example: A Moving-Average Filter

A sequential system computes output values that are a function of the current input values as well as prior input values. Here "current" and "previous" refer to successive signal values that are supplied by some external source. Our system consumes the input molecular types, and so resets the input signal to zero. Our system accepts new input values only after the current output value is cleared, i.e., after some external source consumes all of the output molecular type.

We illustrate our design methodology with a detailed example: a finite impulse response (FIR) filter. An FIR filter is shown in Figure 1a. This system computes a *moving average*: given a time-varying input signal X, the output Y is a smoother version of it. More precisely, the output is one-half the current input value plus one-half the previous value.

Our implementation consists of the following set of reactions. We present the reactions in their entirety and then provide the rationale for the design. We validate the design after mapping it to DNA strand displacement reactions. We present the simulation results in Section 6.

$$\begin{aligned} g + X &\xrightarrow{\mathrm{k_{slow}}} A + C \\ 2C &\xrightarrow{\mathrm{k_{fast}}} R \\ 2A &\xrightarrow{\mathrm{k_{fast}}} Y \end{aligned} \tag{3}$$

$$\begin{aligned} b + R &\xrightarrow{\mathrm{k_{slow}}} G \\ r + G &\xrightarrow{\mathrm{k_{slow}}} B \\ g + B &\xrightarrow{\mathrm{k_{slow}}} Y \end{aligned} \tag{4}$$

$$\begin{aligned} 2R &\xrightarrow{\mathrm{k_{fast}}} 2R + R' \\ 2Y &\xrightarrow{\mathrm{k_{fast}}} 2Y + R' \\ 2G &\xrightarrow{\mathrm{k_{fast}}} 2G + G' \\ 2B &\xrightarrow{\mathrm{k_{fast}}} 2B + B' \\ 2X &\xrightarrow{\mathrm{k_{fast}}} 2X + B' \\ 2R' &\xrightarrow{\mathrm{k_{fast}}} \varnothing \\ 2G' &\xrightarrow{\mathrm{k_{fast}}} \varnothing \\ 2B' &\xrightarrow{\mathrm{k_{fast}}} \varnothing. \end{aligned} \tag{5}$$

$$\begin{aligned} 2S_r &\xrightarrow{\mathrm{k_{slow}}} 2S_r + r \\ 2S_g &\xrightarrow{\mathrm{k_{slow}}} 2S_g + g \\ 2S_b &\xrightarrow{\mathrm{k_{slow}}} 2S_b + b \\ R' + r &\xrightarrow{\mathrm{k_{fast}}} R' \\ G' + g &\xrightarrow{\mathrm{k_{fast}}} G' \\ B' + b &\xrightarrow{\mathrm{k_{fast}}} B' \end{aligned} \tag{6}$$

$$\begin{aligned} R' + X &\xrightarrow{\mathrm{k_{fast}}} A + C \\ G' + R &\xrightarrow{\mathrm{k_{fast}}} G \\ B' + G &\xrightarrow{\mathrm{k_{fast}}} B \\ R' + B &\xrightarrow{\mathrm{k_{fast}}} Y. \end{aligned} \tag{7}$$

The molecular types corresponding to *signals* are X, A, C, R, G, B and Y. These are labeled in Figure 1b. To elucidate the design, we color-code some of these types into three categories: Y and R in red; G in green; and X and B in blue.

In the group of reactions (3), the concentration of X is transferred 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. We discuss this operation below.) 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 group of reactions (4) 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*. (We discuss these types below.) 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 (5), 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 include "waste" types.) Here, all reactions are expressly designed to have two reactants; as discussed in Section 2, 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 (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 (7) 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.

4. General DSP System Synthesis

Building on the example in the last section, we present a general methodology for performing DSP with molecular reactions. DSP operations are specified in terms of four basic modules: *fanout*, *scalar multiplication*, *addition*, and *delay elements*. We discuss constructs for each of these modules. We illustrate the general design method with a second detailed example, a biquad filter.

4.1. Scalar Multiplier

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:

$$c_1 X \longrightarrow c_2 Y. \tag{8}$$

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, i.e., fully consumed all molecules of X, the requisite operation of scalar multiplication is complete.

As discussed in the introduction, a constraint on our designs is that all reactions should be bimolecular reactions. Accordingly, c_1 should be a power of 2. Suppose $c_1 = 2^n$. Then reaction (8) can be replaced by the set of reactions

$$\begin{array}{rcl} 2X & \xrightarrow{\mathrm{k_{fast}}} & X_2 \\ 2X_2 & \xrightarrow{\mathrm{k_{fast}}} & X_4 \\ & \vdots & \\ 2X_{2^{n-1}} & \xrightarrow{\mathrm{k_{fast}}} & c_2 Y. \end{array} \tag{9}$$

We use the notation

$$Multiply(X, Y, c1, c2)$$

to denote the collection of Reactions (9), where $c_1 = 2^n$.

4.2. Adder

Addition performs the operation

$$y = x_1 + x_2.$$

This operation is implemented by choosing two or more reactions with the same product:

$$\begin{array}{rcl} 2X_1 & \xrightarrow{\mathrm{k_{fast}}} & 2Y \\ 2X_2 & \xrightarrow{\mathrm{k_{fast}}} & 2Y. \end{array} \tag{10}$$

Again, we choose bimolecular reactions instead of unimolecular transfers, such as $X_1 \xrightarrow{\mathrm{k_{fast}}} Y$. Once both of these reactions have fired to completion, the concentration of Y will be the former concentration of X_1 plus the former concentration of X_2.

4.3. Fanout

The fanout operation duplicates concentrations. It is implemented by choosing a reaction producing several different products from a single reactant:

$$2X \xrightarrow{\mathrm{k_{fast}}} 2Y_1 + 2Y_2. \tag{11}$$

Once this reaction has fired to completion, both the concentration of Y_1 and the concentration of Y_2 will be equal to the former concentration of X.

A *transfer module* is a special case of a fanout module. It simply transfers a molecular concentration from one type to another:

$$2X \xrightarrow{\mathrm{k_{fast}}} 2Y. \tag{12}$$

Transfer modules are used to resolve type assignment conflicts.

4.4. Delay Element

Delay elements are at the core of digital signal processing. They stores signals values temporarily, allowing for iterative processing. We implement delay elements by transferring concentrations between molecular types based on the absence of other types. Each delay element DE_i is assigned three molecular types $R(ed)_i$, $G(reen)_i$ and $B(lue)_i$. It is implemented by the following reactions.

Phase 1 reactions:

$$\begin{array}{rcl} b + R_i & \xrightarrow{\mathrm{k_{slow}}} & G_i \\ G' + R_i & \xrightarrow{\mathrm{k_{fast}}} & G_i \end{array} \tag{13}$$

Phase 2 reactions:

$$\begin{array}{rcl} r + G_i & \xrightarrow{\mathrm{k_{slow}}} & B_i \\ B' + G_i & \xrightarrow{\mathrm{k_{fast}}} & B_i \end{array} \tag{14}$$

Phase 3 reactions:

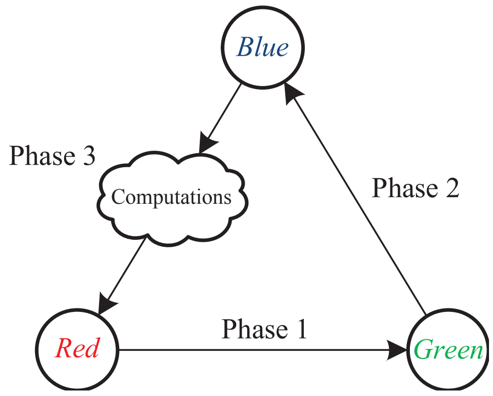
$$\begin{array}{rcl} g + B_i & \xrightarrow{\mathrm{k_{slow}}} & Computations \\ R' + B_i & \xrightarrow{\mathrm{k_{fast}}} & Computations \end{array} \tag{15}$$

We use the notation

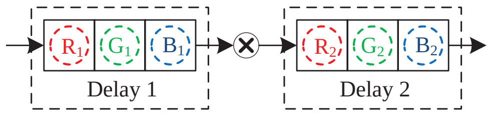
$$Delay(R_i, G_i, B_i, \{output\_list\})$$

to represent Reactions (13), (14), and (15). Here, $\{output\_list\}$ is a list of molecular types that B_i should be transferred to during Phase 3. In addition, system input X is labeled blue, therefore, reactions

$$\begin{array}{rcl} g + X & \xrightarrow{\mathrm{k_{slow}}} & Computations \\ R' + X & \xrightarrow{\mathrm{k_{fast}}} & Computations \end{array} \tag{16}$$



(a) The three-phase transfer scheme.



(b) Cascaded delay elements.

Figure 2: Implementing delay elements.

also fire in Phase 3. We use

$$Input(X, \{output\_list\})$$

to represent these reactions.

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:

$$Computations \xrightarrow{\mathrm{k_{fast}}} R_j. \tag{17}$$

This is illustrated in Figure 2a. The computation reactions fire much faster than the transfer reactions, so molecules of R_j are immediately produced from molecules of B_i. Thus, reactions in Phase 3 effectively transfer blue signals to red signals.

Note that R_j produced in Phase 3 will be a red type of any succeeding delay element DE_j along the signal path from DE_i. In Figure 2b, 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_2 is a succeeding delay element of DE_1, so molecules of B_1 are transferred to R_2 in Phase 3.

For each delay element, the color concentration types R', G', and B' are generated and consumed in the following reactions:

$$\begin{aligned}
2R_i &\xrightarrow{\mathrm{k_{fast}}} 2R_i + R' \\
2Y &\xrightarrow{\mathrm{k_{fast}}} 2Y + R' \\
2G_i &\xrightarrow{\mathrm{k_{fast}}} 2G_i + G' \\
2B_i &\xrightarrow{\mathrm{k_{fast}}} 2B_i + B' \\
2X &\xrightarrow{\mathrm{k_{fast}}} 2X + B' \\
2R' &\xrightarrow{\mathrm{k_{fast}}} \varnothing \\
2G' &\xrightarrow{\mathrm{k_{fast}}} \varnothing \\
2B' &\xrightarrow{\mathrm{k_{fast}}} \varnothing
\end{aligned} \tag{18}$$

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

$$\begin{gathered}
Conc(R', \{red\_type\_list\}) \\
Conc(G', \{green\_type\_list\}) \\
Conc(B', \{blue\_type\_list\})
\end{gathered}$$

to represent Reactions 18. For example,

$$Conc(R', \{R_1, Y\})$$

represents

$$\begin{aligned}
2R_1 &\xrightarrow{\mathrm{k_{fast}}} 2R_1 + R' \\
2Y &\xrightarrow{\mathrm{k_{fast}}} 2Y + R' \\
2R' &\xrightarrow{\mathrm{k_{fast}}} \varnothing.
\end{aligned}$$

For delay elements, the following reactions generate the absence indicator types r, g, and b:

$$\begin{aligned}
2S_r &\xrightarrow{\mathrm{k_{slow}}} 2S_r + r \\
R' + r &\xrightarrow{\mathrm{k_{fast}}} R' \\
2S_g &\xrightarrow{\mathrm{k_{slow}}} 2S_g + g \\
G' + g &\xrightarrow{\mathrm{k_{fast}}} G' \\
2S_b &\xrightarrow{\mathrm{k_{slow}}} 2S_b + b \\
B' + b &\xrightarrow{\mathrm{k_{fast}}} B'
\end{aligned} \tag{19}$$

We use

$$Abs(S_r, S_g, S_b, r, g, b, R', G', B')$$

to represent these reactions.

Here r, g and b are continually and slowly generated. However, they only persist 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' and B') and three absence indicators (r, g and 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 the delay elements must wait for each to complete its current phase before they can move to the next phase.

4.5. Example of a Biquad filter

We illustrate our synthesis method with a second example, an infinite impulse response (IIR) biquad filter. Biquad filters are basic building blocks of modern DSP systems. Highly stable, high-order filters can be implemented by cascaded biquad blocks [1]. A biquad filter is shown in Figure 3a and the corresponding molecular types are labeled in Figure 3b. It is realized by the following reactions.

Delay elements:

$$\begin{array}{l} Delay(R_1, G_1, B_1, \{R2, F, C\}) \\ Delay(R_2, G_2, B_2, \{H, E\}) \end{array} \tag{20}$$

System input:

$$Input(X, \{R_1, A\}) \tag{21}$$

Scalar multiplications:

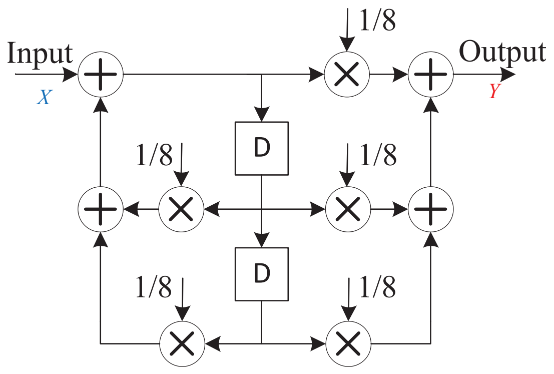
$$\begin{array}{l} Mult(A, Y, 8, 1) \\ Mult(C, Y, 8, 1) \\ Mult(E, Y, 8, 1) \\ Mult(F, X, 8, 1) \\ Mult(H, X, 8, 1) \end{array} \tag{22}$$

Concentration indicators:

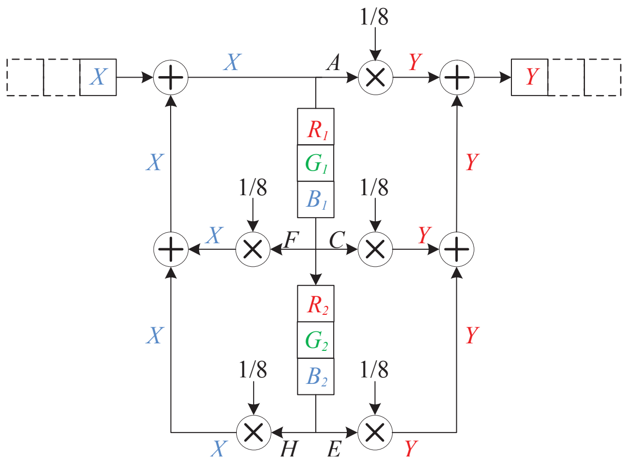
$$\begin{array}{l} Conc(R', \{R1, R2, Y\}) \\ Conc(G', \{G1, G2\}) \\ Conc(B', \{B1, B2, X\}) \end{array} \tag{23}$$

Absence indicators:

$$Abs(S_r, S_g, S_b, r, g, b, R', G', B') \tag{24}$$



(a) Schematic of the biquad filter.



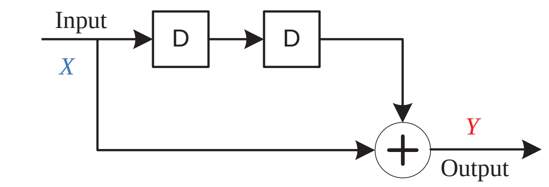
(b) The filter in a three-phase configuration.

Figure 3: A biquad filter.

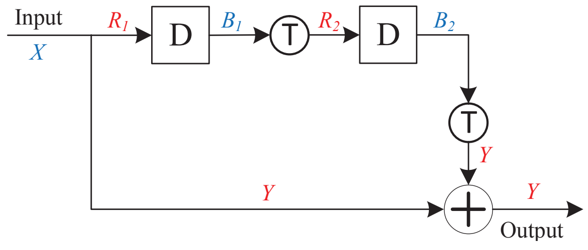
5. Synthesis Flow

We present guidelines for an automated synthesis flow. The DSP system is represented by a block diagram $G(V, E)$, where the vertex set V represents basic modules – scalar multiplication, addition, fanout and delay element – and the edge set E represents connections. Each edge e_i is assigned a molecular type. The concentration of this type represents the signal flowing through e_i. The system is synthesized as follows:

1) Each delay element $DE_i \in V$ is assigned three color-coded molecular types R_i, G_i and B_i. Here R_i corresponds to the input edge, G_i is the internal storage molecule type, and B_i corresponds to the output edge.
2) The system input and output are assigned types X and Y, respectively. (For simplicity, we only consider systems with a single input and a single output. However, the method easily generalizes to systems with multiple inputs and outputs.)
3) The incoming edges of each adder are assigned



(a) A filter with two delay elements directly connected.



(b) Molecular type assignment.

Figure 4: An example of molecular type assignment. Transfer modules are denoted by a circle with letter "T".

the same molecular type as the outgoing edge. With all the inputs assigned the same type, the system implicitly performs an addition operation: each reaction produces a concentration that is added to the sum.

4) If there are assignment conflicts, transfer modules are included. For instance, if an adder has been assigned two conflicting types T_1 and T_2, say because its inputs are from different delay operations, then a transfer reaction is included:

$$2T_1 \xrightarrow{\mathrm{k_{fast}}} 2T_2.$$

This reaction transfers the concentration of T_1 to T_2.

5) Next, if there are any unassigned edges, these are assigned arbitrary molecular types (without creating conflicts).
6) With all edges assigned non-conflicting molecular types, reactions are generated for each vertex according to the template of Reactions (8) to (16).
7) Finally, the common indicator reaction set (18) and (19) are included.

Figure 4 gives an example of transfer modules. Figure 4a shows a simple filter for time-interleaved input data. It contains two delay elements. Since these two delay elements are directly connected, a transfer module is included for converting B_1 to R_2. Similarly, a second transfer module is included for transferring B_2 to Y, the molecular type for the adder. These molecular type assignments are shown in Figure 4b.

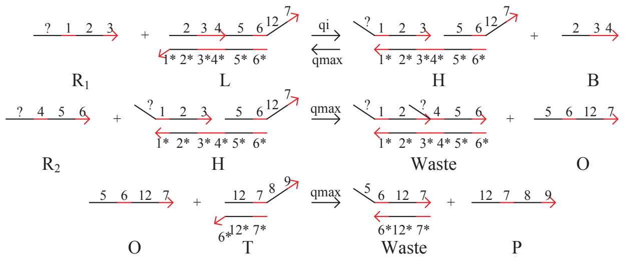


Figure 5: An example of DNA strand displacement.

For a DSP system with n delay elements, there are $3n + 2$ molecular types to represent the n delay elements as well as the system input and output. Accounting for the absence and color concentration indicators, there are an additional 9 molecular types. The number of intermediate types will vary according to system architecture.

6. Simulations

6.1. Mapping to DNA Strand Displacement

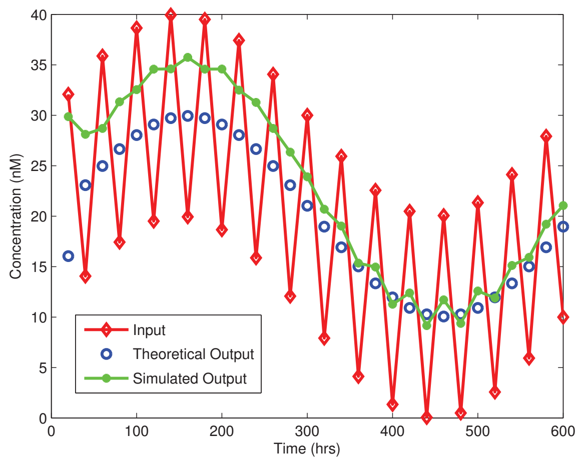
Given a specification of an abstract molecular reaction network that implements the requisite computation, the next step is to map it to specific molecular reactions. We describe a mapping to DNA strand displacement reactions. The reader is referred to [11] for a detailed discussion of this mechanism. Here we illustrate with an example.

Consider the DNA strand displacement reaction shown in Figure 5. Here a single strand of DNA R_1 replaces the top strand of a double-strand DNA L; 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, G 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:

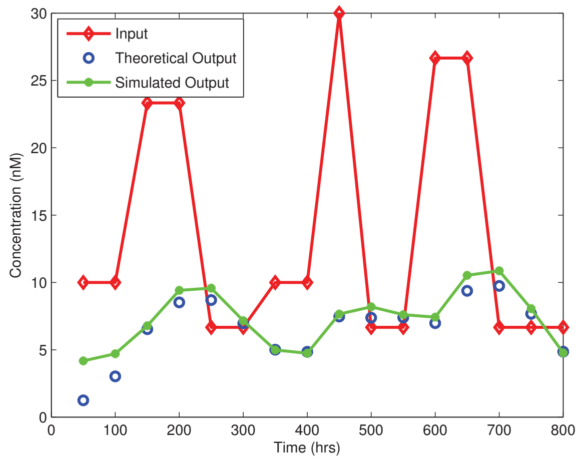
$$R_1 + R_2 \xrightarrow{\mathrm{k}} P.$$

6.2. Simulation Results

To validate our designs for the moving-average and biquad filters, we map the reactions presented in Section 3 and 4 to DNA strand displacement reactions, using the method in [11]. We generate the corresponding system of kinetic differential equations



(a) The moving-average filter.



(b) The biquad filter.

Figure 6: Simulation results for DNA-level designs.

and simulate these. We use similar parameters to [11]: The initial concentrations of auxiliary complexes is $C_{max} = 10^{-5}M$ and the maximum strand displacement rate constant is $q_{max} = 10^{6}M^{-1}s^{-1}$. The rate constant for the "slow" reactions is set to $k_{\text{slow}} = 5.56 \times 10^{4}M^{-1}s^{-1}$. For "fast" reactions it is set to $k_{\text{fast}} = 3 \times k_{\text{slow}}$. The initial concentrations of S_r, S_g, and S_b are set to $1nM$.

The simulation results for the moving-average filter are shown in Figure 6a. 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 system every 20 hours. The figure shows the theoretical output, i.e., an exact calculation of filtering, as well as simulation results.

We see that our design performs very well, filtering out the high-frequency component as expected. The simulated output concentration does not quite track the theoretical output concentration; it is higher than it should be for high input concentrations. The explanation for this is that, for high input concentrations, the reactions fire quickly, so the computational cycle completes early. Before the next cycle begins, some "leakage" of the output concentration occurs.

The simulation results for the biquad filter are shown in Figure 6b. Here molecules of X are injected and molecules of Y are collected from the system every 50 hours. We supply step-like and impulse-like changes in X. 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 λ greater than 1000 is readily achievable. When λ 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 even when the concentrations of R', G', or B' are high. Also, the computational modules slow down. Accordingly, the accuracy of the computation degrades.

7. Remarks

The methodology presented in this paper is *self-timed* and *asynchronous* in the sense that computational cycles only begin when all molecules of the output type Y are consumed by an external sink. The computation itself is essentially *rate-independent*, meaning that within a broad range of values for the kinetic constants, the computation is exact and independent of the specific rates.

An alternative strategy would be to use *clocking* to implement *synchronous* computation. We have presented such a strategy in [6]. In that work, we describe a strategy for generating a *clock signal* through robust, sustained chemical oscillations. We implement *memory elements* by transferring concentrations between molecular types in alternating phases of the clock.

Although pertaining to biology, the contributions of this paper are not experimental nor empirical; rather they are constructive and conceptual. 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. Such features could be transformative for applications such as drug delivery and metabolic engineering.

Acknowledgement

This work is supported by NSF grants #CCF-0946601 and #CCF-1117168. A preliminary publication of this work has appeared as [12].

References

[1] K. K. Parhi, *VLSI Digital Signal Processing Systems*. John Wiley & Sons, 1999.

[2] A. Arkin and J. Ross, "Computational functions in biochemical reaction networks," *Biophysical Journal*, vol. 67, no. 2, pp. 560 – 578, 1994.

[3] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, "An autonomous molecular computer for logical control of gene expression," *Nature*, vol. 429, no. 6990, pp. 423–429, 2004.

[4] L. Qian and E. Winfree, "Scaling up digital circuit computation with DNA strand displacement cascades," *Science*, vol. 332, no. 6034, pp. 1196–1201, 2011.

[5] M. N. Win and C. D. Smolke, "higher-order cellular information processing with synthetic RNA devices," *Science*, vol. 322, no. 5900, pp. 456–460, 2008.

[6] H. Jiang, M. D. Riedel, and K. K. Parhi, "Synchronous sequential computation with molecular reactions," in *Design Automation Conference*, 2011, pp. 836–841.

[7] P. Senum and M. D. Riedel, "Rate-independent constructs for chemical computation," *PLoS ONE*, vol. 6, no. 6, 2011.

[8] J. C. Anderson, E. J. Clarke, A. P. Arkin, and C. A. Voigt, "Environmentally controlled invasion of cancer cells by engineered bacteria," *Journal of Molecular Biology*, vol. 355, no. 4, pp. 619–627, 2006.

[9] S. Venkataramana, R. M. Dirks, P. W. K. Rothemund, E. Winfree, and N. A. Pierce, "An autonomous polymerization motor powered by DNA hybridization," *Nature Nanotechnology*, vol. 2, pp. 490–494, 2007.

[10] S. Venkataramana, R. M. Dirks, C. T. Ueda, and N. A. Pierce, "Selective cell death mediated by small conditional RNAs," *Proceedings of the National Academy of Sciences*, vol. 107, no. 39, pp. 16 777–16 782, 2010.

[11] D. Soloveichik, G. Seelig, and E. Winfree, "DNA as a universal substrate for chemical kinetics," *Proceedings of the National Academy of Sciences*, vol. 107, no. 12, pp. 5393–5398, 2010.

[12] H. Jiang, A. P. Kharam, M. D. Riedel, and K. K. Parhi, "A synthesis flow for digital signal processing with biomolecular reactions," in *IEEE International Conference on Computer-Aided Design*, 2010, pp. 417–424.