A Framework for Computing Discrete-Time Systems and Functions using DNA

A DISSERTATION

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

Sayed Ahmad Salehi

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

Doctor of Philosophy

Advisors: Keshab K. Parhi and Marc D. Riedel

July, 2017

Acknowledgements

This thesis has been done under supervision of two advisors, Professor Keshab K. Parhi and Professor Marc D. Riedel. With different research backgrounds and approaches, they acutely supervised my research in totally different aspects. I am thankful to both of them because my research achievements could not be accomplished without their thoughtful guidance, constructive criticisms, and fruitful scientific discussions. Furthermore, I sincerely appreciate their encouragement, support, kindness, and true understanding of my concerns during my graduate studies at the University of Minnesota.

I would like to express my deeply-felt thanks to other members of my thesis committee: Professor Kiarash Bazargan and Professor Antonia Zhai. Their invaluable comments and feedback helped me improve the quality of this research.

I would also like to thank Dr. Vishwesh Kulkarni for our productive discussions about my research. I am grateful to my current and former members of our research group. In particular, Dr. Yingjie Lao, Dr. Chuan Zhang, Dr. Hua Jiang, Dr. Te-Lung Kung, Dr. Mojtaba Bandarabadi, Abdolreza Rashno, Sandhya Koteshwara, Bhaskar Sen, Sandeep Avvaru, Denis Chu, Zisheng Zhang, Yin Liu, and Anoop Koyily for all the good time I spent with them.

I want to acknowledge my thanks to National Science Foundation for their financial support of this research, under grants CCF-1117168 and CCF-14234707, without which I would not have been able to develop my scientific discoveries.

Last, but most importantly, my sincerest gratitude goes to my family: my wonderful mom Badrosadat, my dad Sayed Kamal, my brother Sayed Mohamad, and my beautiful sister Sabihesadat. Definitely I would not be standing where I am today without their faithful love and support.

Dedication

To my parents for their endless, faithful, and unconditional love and support.

Abstract

Due to the recent advances in the field of synthetic biology, molecular computing has emerged as a non-conventional computing technology. A broad range of computational processes has been considered for molecular implementation. In this dissertation, we investigate the development of molecular systems for performing the following computations: signal processing, Markov chains, polynomials, and mathematical functions.

First, we present a *fully asynchronous* framework to design molecular signal processing algorithms. The framework maps each delay unit to two molecular types, i.e., first-type and second-type, and provides a 4-phase scheme to synchronize data flow for any multi-input/multi-output signal processing system. In the first phase, the input signal and values stored in all delay elements are consumed for computations. Results of computations are stored in the first-type molecules corresponding to the delay units and output variables. During the second phase, the values of the first-type molecules are transferred to the second-type molecules for the output variable. In the third phase, the concentrations of the first-type molecules are transferred to the second-type molecules associated with each delay element. Finally, in the fourth phase, the output molecules are collected. The method is illustrated by synthesizing a simple finite-impulse response (FIR) filter, an infinite-impulse response (IIR) filter, and an 8-point real-valued fast Fourier transform (FFT). The simulation results show that the proposed framework provides faster molecular signal processing systems compared to prior frameworks.

We then present an overview of how continuous-time, discrete-time and digital signal processing systems can be implemented using molecular reactions. We also present molecular sensing systems where molecular reactions are used to implement analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). These converters can be used to design mixed-signal processing molecular systems. A complete example of the addition of two molecules using digital implementation is described where the

concentrations of two molecules are converted to digital by two 3-bit ADCs, and the 4-bit output of the digital adder is converted to analog by a 4-bit DAC.

Furthermore, we describe implementation of other forms of molecular computation. We propose an approach to implement any first-order Markov chain using molecular reactions in general and DNA in particular. The Markov chain consists of two parts: a set of states and state transition probabilities. Each state is modeled by a unique molecular type, referred to as a data molecule. Each state transition is modeled by a unique molecular type, referred to as a control molecule, and a unique molecular reaction. Each reaction consumes data molecules of one state and produces data molecules of another state. The concentrations of control molecules are initialized according to the probabilities of corresponding state transitions in the chain. The steady-state probability of the Markov chain is computed by the equilibrium concentration of data molecules. We demonstrate our method for the Gamblers Ruin problem as an instance of the Markov chain process. We analyze the method according to both the stochastic chemical kinetics and the mass-action kinetics model.

Additionally, we propose a novel *unipolar molecular encoding* approach to compute a certain class of polynomials. In this molecular encoding, each variable is represented using two molecular types: a type-0 and a type-1. The value is the ratio of the concentration of type-1 molecules to the sum of the concentrations of type-0 and type-1 molecules. With the new encoding, CRNs can compute any set of polynomial functions subject only to the limitation that these polynomials can be expressed as linear combinations of Bernstein basis polynomials with positive coefficients less than or equal to 1. The proposed encoding naturally exploits the expansion of a power-form polynomial into a Bernstein polynomial. We present molecular encoders for converting any input in a standard representation to the fractional representation, as well as decoders for converting the computed output from the fractional to a standard representation.

Lastly, we expand the unipolar molecular encoding for bipolar molecular encoding and propose simple molecular circuits that can compute multiplication and scaled

addition. Using these circuits, we design molecular circuits to compute more complex mathematical functions such as e^{-x}, $\sin(x)$, and $\text{sigmoid}(x)$. According to this approach, we implement a molecular perceptron as a simple artificial neural network.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

The field of synthetic biology has advanced remarkably in the last 20-25 years. The progress in the broad field of synthetic biology continues to accelerate at a rate even faster than Moore's law that refers to doubling in the number of transistors on an integrated circuit (IC) chip every 18 months. A similar growth in synthetic biology is referred as Carlson's law [3], [4]. Due to the remarkable advancements in the field of synthetic biology, biomolecular systems are emerging as new technologies for performing computation. For bimolecular systems the concentrations of molecules, i.e., number of molecules per unit volume, represent signal values, in the same way that for electronic systems voltages, i.e., energy per unit charge, represent signal values. One can design molecular systems to perform signal processing or other forms of computation in terms of molecular concentrations.

The idea of computation directly with molecular reactions, as opposed to writing computer programs to analyze chemical systems, dates back to the early work by Conard [5] and the seminal work of Adleman [6].In this context, a chemical reaction

network (CRN) transforms input concentrations of molecular types into output concentrations, and thus implements computation. (It should be noted that the equilibrium concentrations of the output molecules are considered as the computed output of the system.) In other words, CRNs are used as a programming language for designing molecular computing systems. These designed programs, i.e., CRNs, are technology-independent and can be realized by any physical molecular system. In this research, the designed CRNs are mapped to DNA, a promising technology for such systems.

Although the ability to compute using biological and chemical molecules, as an alternative to computing using silicon ICs has been demonstrated [7], the incentive of molecular computing is not to compete with electronic circuits in terms of computational speed or size. Electronic circuits perform computations on the scale of nanoseconds whereas the computational rate of molecular systems is measured in minutes or even hours (typically 10-15 orders of magnitude slower). For example, when using a molecular system to monitor a protein 4 times a day, one requires a sample period of 21,600 seconds whereas, the sampling period for the electronic circuits is 1 ns with a clock speed of 1 GHz. Fortunately, today's DNA circuits can meet these sample rate constraints for simple circuits. Due to the advancement in the semiconductor technology, electronic circuits have been scaled to the size of nanometers. The size of molecular computing systems is also in the order of nanometers.

The main advantage of molecular computing systems is their environment of application. Whereas electronic circuits are pervasive in industrial and commercial applications, in some situations, it is more appropriate to implement computation directly with biological mechanisms. Molecular computing has the potential to revolutionize monitoring concentrations or rates of change of concentrations of proteins and targeted drug delivery. For example, one might want to implement a molecular mechanism for detecting protein markers of cancer and for producing drugs targeted precisely to cancerous cells. In fact, biomolecular circuits are the best alternative for electronic circuits and other computing technologies for in vivo applications. They are compatible with

living cells and, unlike electronic circuits, biomolecular circuits do not need batteries. Biomolecular circuits can obtain the required energy from resources, such as heat, when inside living bodies.

This research discusses two categories of molecular computing systems. The first group is made up of molecular reactions that perform discrete-time signal processing. The second group is compromised of molecular systems for other forms of computation. For the first group, only one molecular type is used to represent the value of each signal/variable. This is the traditional way of molecular signal representation where, the signal/variables value is directly defined by the concentration of the assigned molecular type. For the second group, however, each signal/variable is represented using two molecular types: type-0 and type-1. The variables value is defined as the ratio of concentration of molecule type-1 over the total concentration of type-0 and type-1 molecules. This novel representation, proposed for the first time in this research, is referred to as fractional coding. Fractional coding empowers the computational capability of chemical reactions and enables them to compute more complex mathematical functions.

1.2 Contribution

The contributions of this research can be listed as follows.

1- The most challenging parts of molecular implementation of signal processing algorithms are signal flow controlling and delay (memory) units. This thesis proposes an asynchronous forward signal flow scheme that is able to implement multiple-input/multiple-output signal processing systems including delay units. Prior work [2] has proposed two other schemes, i.e., *synchronous* and *RGB* schemes, for signal flow control and has verified them for the implementation of finite impule response (FIR) and infinite impulse response (IIR) filters. In this thesis, we implement FIR and IIR filters and 8-point real-valued FFT algorithm using both prior signal flow schemes and

the proposed signal flow scheme. Then we compare them with respect to the number of required molecular types and reactions, computational speed and accuracy, and robustness.

2- In order to increase the robustness of molecular computing systems, we propose mixed (digital and analog) signal processing structures. Since digital is more robust than analog, part of the computation can be performed in digital. We propose required analog to digital and digital to analog converters by molecular reactions.

3- Markov chains have been used to model different systems. For the first time, we propose a systematic method for implementing Markov chains by molecular reactions.

4-We propose fractional coding for molecular systems that bridges molecular computing and stochastic logic.

5- Based on the proposed unipolar fractional coding, we present a systematic method to design molecular reactions for computing Bernstein polynomials. The method is used to compute a wide range of general polynomials with both the input and output in the unit interval $[0, 1]$.

6- Based on the unipolar and bipolar fractional coding, we propose molecular reactions for computing mathematical functions. We show that some common complex functions such as $\sin x$, e^{-x}, $\log(1 + x)$, and $sigmoid(x)$ can be computed by molecular reactions. Since all of these reactions are bimolecular reactions, i.e., each reaction has only two reactants, they are compatible with natural DNA and can be implemented by DNA systems with a high level of accuracy.

1.3 Outline of the Dissertation

This dissertation is organized as follows:

Chapter 2, provides the development process employed in this research for implementation of molecular computing systems. The process consists of three phases: design,

simulation, and implementation. The chapter describes the tools and methods used in each phase.

Chapter 3 presents a new design framework for discrete-time signal processing systems by molecular reactions. The presented framework is a fully-asynchronous scheme. The DNA implementation of the new framework is compared to prior frameworks.

Chapter 4 reviews molecular implementation of continuous-time, discrete-time, and digital signal processing systems. It also presents molecular sensing systems where molecular reactions are used to implement analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). The chapter provides several examples including a complete example of the addition of two molecular signals using digital implementation. For this example, the concentrations of two input molecules are converted to digital by two 3-bit ADCs, and the 4-bit output of the digital adder is converted to analog by a 4-bit DAC.

Chapter 5 discusses a systematic method that can be used in order to synthesize molecular reactions for computing any first order Markov chain process. This chapter theoretically analyze the synthesized reactions and validate them for DNA implementation.

Chapter 6 introduces unipolar fractional coding as a new non-standard representation of variables by molecules. Based on this molecular coding the chapter presents a systematic method for synthesis of chemical reactions that are able to compute polynomials.

Chapter 7 introduces bipolar representation and very simple molecular reactions for operations such as multiplication and addition. This chapter then presents a systematic method for synthesis of chemical reactions that are able to compute mathematical functions. The chapter also describes implementation of molecular perceptron using the fractional molecular coding.

Finally, Chapter 8 concludes remarks and points out some of the possible future research directions.

Chapter 2

Design and Modeling of Molecular Computing Systems

Generally speaking, a product development process for a system consists of three main phases: design (programming), simulation (modeling), and implementation. Similarly, we develop the desired molecular systems through several iterations of a design-simulation-implementation cycle. This chapter describes methods and tools we use in each development phase.

2.1 Design (Programming)

A common way to begin the design of a system is representing it using an abstract level. For example, for electronic circuits, the design begins in different levels of abstraction such as system blocks, register transfer levels (RTLs), gates, and transistors. Analogous levels of abstraction exist for biological systems: multicellular organisms, single cells, signaling pathways, genetic regulatory networks, proteins, and molecular dynamics and reactions. In this research, we use *molecular reactions* as the abstract level to design, analyze, and discuss target computing molecular systems.

Chemical reaction network (CRN) is commonly used as a describing and programming language for molecular reactions. As we describe later in this chapter, CRNs have well-defined theory and simulation software tools. Furthermore, due to the recent advances in DNA nanotechnology, it is possible to map and synthesize nearly arbitrary CRNs by DNA reactions. Thus, we can benefit from the advantages of CRNs by selecting *molecular reactions* as the abstract level for the design phase.

A CRN consists of a set of molecular reactions. For example, the simple CRN, represented in (2.1), is composed of two reactions; the first reaction in the first line and the second reaction in the second line.

$$\begin{aligned} A + B &\xrightarrow{\mathrm{k_1}} C \\ 2C &\xrightarrow{\mathrm{k_2}} A \end{aligned} \tag{2.1}$$

The first reaction says that one molecule of type A combines with one molecule of type B to produce one molecule of type C. The rate constant, k_1, denotes the speed of this reaction. Similarly, the second reaction says that two molecules of C react and form one molecule of A.

In the design phase, we synthesize chemical reactions such that, in terms of molecular concentrations, the system produces a specific output for each input. In other words, the output concentration is a desired function of the input concentration.

In the next phase, we discuss how the dynamic behavior of each chemical reaction and the whole CRN can be quantitatively modeled and simulated.

2.2 Simulation (Modeling)

Assuming that molecular concentrations and reaction rate constants are well-defined, there are two main models for simulation of CRNs: stochastic model and mass-action kinetic model. Both models deal with molecular concentrations. However, the stochastic model is used when the number of molecules is small (as small as hundreds of molecules)

and the mass-action kinetic model is applicable to systems with a sufficiently large number of molecules.

In the stochastic model, the molecular concentrations are considered as discrete values, while in the mass-action kinetic model, the concentrations are continuous variables. It has been shown that if the number of molecules increases, the stochastic model converges to the mass-action kinetic model, and for molecular concentration of infinity, both models are the same.

2.2.1 Stochastic model

Based on the stochastic model, each reaction is fired randomly provided there is enough number of reacting molecules [8][9]. In fact, the probability of firing each reaction is proportional to the rate constant and the number of reacting molecules available in the system.

In the stochastic model, the behavior of the CRN is simulated by the sequence of reactions. The firing probabilities are updated after the completion of each reaction.

Suppose for the CRN shown in (2.1), the initial concentrations of A, B, and C are 15, 10, and 5 molecules, respectively. The firing probabilities for the first reaction, $P(R1)$, and the second reaction, $P(R2)$, can be calculated as

$$P(R1) = \frac{k_1\binom{15}{1}\binom{10}{1}}{k_1\binom{15}{1}\binom{10}{1} + k_2\binom{5}{2}}$$

and

$$P(R2) = \frac{k_2\binom{5}{2}}{k_1\binom{15}{1}\binom{10}{1} + k_2\binom{5}{2}}.$$

Depending on which reaction takes place, the firing probabilities of reactions are updated for the next reaction. The calculation can be continued until either there is no possible firing reaction, the same pattern of firing reaction repeats, or a sufficiently large number of reactions are completed. For a CRN, even with a particular initial concentration, the sequence of fired reactions is not the same if the simulation is repeated.

Therefore, the simulation is repeated enough times to obtain the distribution of the final output.

2.2.2 Mass-action kinetic model

The second model, i.e., Mass-action kinetic model, is based on the mass-action law, where the concentrations of molecules are continuous variables and their time variation can be described by ordinary differential equations (ODEs). The concentration of molecule A is denoted $[A]$, and typically its unit is moles per liter. It is noticeable that one mole is 6.02×10^{23} molecules, and the symbol M is used for moles per liter. For the CRN shown in (2.1), the model leads to the following ODEs:

$$\begin{aligned} \frac{d[A]}{dt} &= -k_1[A][B] + k_2[C]^2 \\ \frac{d[B]}{dt} &= -k_1[A][B] \\ \frac{d[C]}{dt} &= k_1[A][B] - 2k_2[C]^2 \end{aligned} \tag{2.2}$$

In general, the ODEs produced by the mass-action model of CRNs can be solved by standard numerical techniques, and thus one can generate the time variation dynamics of molecular concentrations.

For both models, there are some software tools that perform simulations for different CRNs accordingly. In this thesis, however, we use our own MATLAB code for the stochastic kinetic model and a Mathematica code written by Caltech for the mass-action kinetic model.

2.3 Implementation

One should notice that the contributions of this research are neither experimental nor empirical; rather, they are constructive and conceptual. CRNs, as a fundamental model of computation, are used to design systems for performing desired computations. However, in order to validate practical aspects of our theoretical designs, we map them

to DNA reactions. These DNA reactions are then simulated to verify the functionality and performance of the design.

There are three reasons behind why we choose DNA to validate the physical implementation of our designs:

1. DNA is a medium with biological origin. This means that DNA development of our designs can be realized *in vivo* or *in vitro* and potential application of our designs for smart drugs and protein monitoring.

2. DNA for the community of synthetic biology is like silicon for the electronics community. As development of silicon devices has made it feasible to produce low-cost complex electronic circuits, DNA technology is reducing the cost and time of constructing artificial biological systems. Moreover, new synthesis technologies are increasing the length and accuracy of the synthesized DNA molecules. Although it has not yet been achieved, the technology is heading toward making the DNA design phase and the DNA fabrication phase independent; designers only think of what DNA molecules they should use and then let technology figure out how to realize them.

3. Much work was involved in developing automated tools that map CRNs to DNA reactions. Fortunately, there are already some software tools that can produce DNA reactions for CRNs. In this research we use such a tool that is a Mathematica code developed by Caltech. The code produces DNA reactions for a given CRN and simulates them based on the mass-action kinetic model. Such simulation predicts the behavior of the actual DNA implementation with an acceptable accuracy.

We describe two approaches used to produce DNA reactions that can emulate the kinetic of CRNs. Both approaches are based on the toehold-mediated DNA strand-displacement reactions. Toehold-mediation was first introduced in [10] for the construction of DNA tweezers. We can map a molecular reaction to a set of DNA strand displacement (DSD) reactions using the *toehold mediated* mechanism if we consider similar strands of DNA as one molecular type.

Approach 1: We briefly describe the first approach of mapping chemical reactions to DNA strand displacement reactions with an example. The reader is referred to Soloveichik et al. for a detailed discussion of this mechanism [11]. The following is a simple example.

Consider the DNA strand displacement reaction shown in Figure 2.1. 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: $R_1 + R_2 \xrightarrow{k} P$.

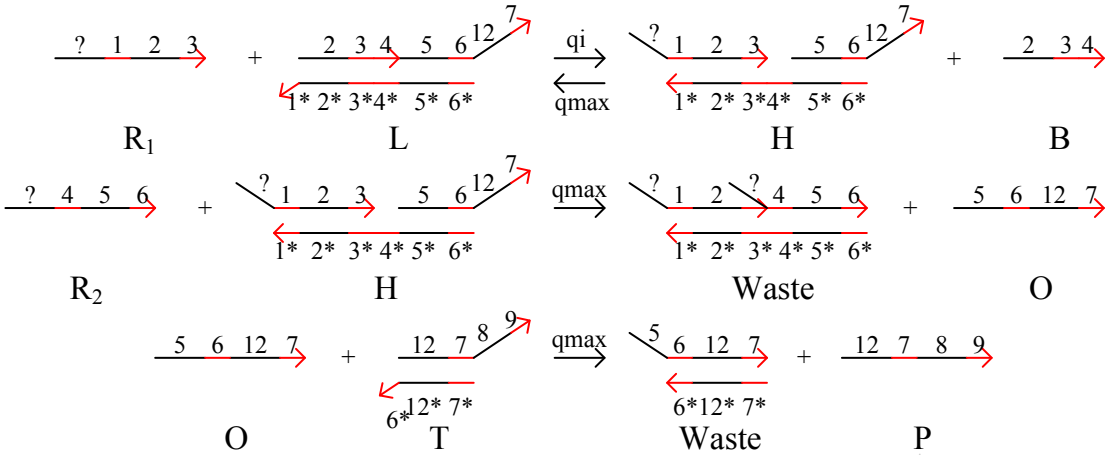


Figure 2.1: An example of DNA strand displacement.

Approach 2: We describe the second approach for chemical reactions with the same pattern, i.e., bimolecular reactions with one product. We use the template presented in Figure 2.2 for the implementation of these reactions by DNA strand-displacement reactions.

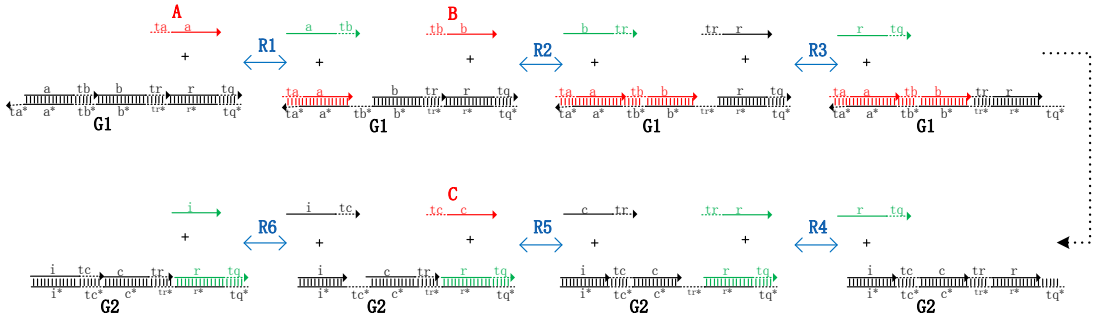


Figure 2.2: **DNA implementation of $A + B \rightarrow C$.** According to the methodology developed in [1], a sequence of six DNA strand displacement reactions, $R1 - R6$, implement bimolecular reaction $A + B \rightarrow C$.

Fig. 2.2 shows a sequence of six DNA reactions, R1-R6, that implement molecular reaction $A + B \rightarrow C$. All DNA reactions are based on the toehold mediated mechanism first presented in [10]. The primary molecules, A, B, and C, are represented by single strand DNA molecules – red strands in Fig. 2.2 – composed of a toehold and a main domain part. The initial system provides required gate and auxiliary molecules, i.e., DNA molecules G1, G2, <tr r>, <c tr>, and <i tc> – black strands in Fig. 2.2. Furthermore, the concentration of gate and auxiliary strands are initialized to be large enough to efficiently supply the sequence of DNA reactions to continue as long as the primary molecules last.

Each reaction in the sequence of DNA reactions produces the mediating toehold for the next reaction. The sequence starts when the toehold domain of input molecule A, i.e., ta, binds with its WatsonCrick complementary domain in gate G1, i.e., ta*. This leads to the binding of whole molecules of A to gate G1. Similarly, through reaction R2, the DNA molecule B binds to gate G1, and in Reaction R5, the output DNA molecule C is released from gate G2. For details of the mechanism, the reader is referred to [1]. The authors in [1] have experimentally validated that the sequence of DNA strand displacement reactions in Fig. 2.2 do implement the expected kinetics

for the desired bimolecular reaction. They also showed that the rate constant can be tuned by adjusting the initial concentrations of gates and auxiliary molecules. The linear, double-stranded DNA molecules used in the mechanism can be derived from biologically synthesized (plasmid) DNA. Compared to the first approach, for the second approach, compatibility with natural DNA leads to the reduction of errors associated with chemically synthesized DNA.

The following chapters discuss molecular implementation of signal processing and other forms of computations using the development process described in this chapter.

Chapter 3

Asynchronous Discrete-time Signal Processing

General signal processing algorithms can be specified in terms of two basic modules: computation and delay (memory) units. The computation module is mainly composed of multiplication and addition, and its molecular implementation has been realized in prior work [12][2]. The most challenging parts of molecular implementation of signal processing algorithms, however, are delay (memory) units and signal transfer among delay units and computation units. In this chapter we discuss a methodology to implement signal processing systems including delay units. We propose a framework that controls signal flow among computation and delay units. This framework is called fully *asynchronous schemes*.

3.1 Prior Work

For discrete-time systems the corresponding computations start after the inputs are sampled at specific points in time. In these systems the timings of signal transfers need to be synchronized in order to avoid any interference in computations. The concept

of a computational cycle in a molecular system is critical. Two different synchronization schemes have been proposed in prior work; these include: fully-synchronous and globally-synchronous locally-asynchronous. Fully synchronous systems are synchronized by a two-phase clock [13, 2]. In a globally-synchronous locally-asynchronous systems, three proteins, referred as Red (R), Green (G) and ($Blue$) are introduced. The transfer of R to G, G to B and B to R completes a computational cycle. The global RGB *clock* provides global synchronization [14, 2]. Typically, RGB clocked systems are faster than the fully-synchronous systems, as the latter involve more phases of transfers. The protein transfer operation is a slow operation and is the bottleneck in molecular systems with respect to sample period. Although fully-synchronous systems require a two-phase clock, this clock is designed from a 4-phase protein transfer mechanism. This section presents a brief review of the fully-synchronous and the RGB systems.

All reactions in the discrete-time system are implemented using only two coarse rate categories for the reaction rate constants, i.e., k_{fast} and k_{slow}. Given reactions with any such set of rates, the computation is correct. 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 illustrate both schemes with a simple example, a moving-average filter. In fact, it is a first-order discrete-time low-pass filter. The circuit diagram for the filter is shown in Figure 3.1. 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 [15].

3.1.1 Fully-Synchronous Framework

In this framework a global clock signal synchronizes signal transfers in the system. For a molecular clock, reactions are chosen that produce sustained oscillations in terms of chemical concentrations. With such oscillations, a low concentration corresponds

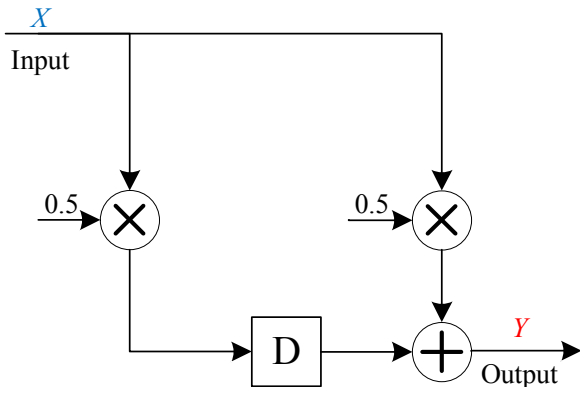


Figure 3.1: Block diagram for the moving-average filter [2].

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 [16, 17]. Unfortunately, none of these schemes is quite suitable for synchronous sequential computation: the required clock signal should be symmetrical, with abrupt transitions between the phases. A new design was proposed in [2] and [13] for multi-phase chemical oscillator. For a 4-phase oscillator the phases can be represented by molecular types $R,\ G,\ B,\ V$. First consider the reactions

$$\begin{aligned} 2S_r &\xrightarrow{\text{k}_{\text{slow}}} r + 2S_r \\ 2S_g &\xrightarrow{\text{k}_{\text{slow}}} g + 2S_g \\ 2S_b &\xrightarrow{\text{k}_{\text{slow}}} b + 2S_b \\ 2S_v &\xrightarrow{\text{k}_{\text{slow}}} v + 2S_v \end{aligned} \tag{3.1}$$

$$\begin{aligned} R + r &\xrightarrow{\text{k}_{\text{fast}}} R \\ G + g &\xrightarrow{\text{k}_{\text{fast}}} G \\ B + b &\xrightarrow{\text{k}_{\text{fast}}} B \\ V + v &\xrightarrow{\text{k}_{\text{fast}}} V. \end{aligned} \tag{3.2}$$

In reactions (3.1), the molecular types $r,\ g,\ b,\ v$ are generated slowly and constantly, from source types S_r, S_g, S_b, S_v, whose concentrations do not change with the reactions.

In reactions (3.2), the types *R, G, B, V* quickly consume the types *r, g, b, v*, respectively. Call *R, G, B, V* the phase signals and *r, g, b, v* the absence indicators. The latter are only present in the absence of the former. The reactions

$$\begin{aligned} R + v &\xrightarrow{\text{k}_{\text{slow}}} G \\ G + r &\xrightarrow{\text{k}_{\text{slow}}} B \\ B + g &\xrightarrow{\text{k}_{\text{slow}}} V \\ V + b &\xrightarrow{\text{k}_{\text{slow}}} R \end{aligned} \tag{3.3}$$

transfer one phase signal to another, in the absence of the previous one. The essential aspect is that, within the *R, G, B, V* 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

$$\begin{aligned} 2G &\underset{k_{fast}}{\overset{k_{slow}}{\rightleftharpoons}} I_G \\ R + I_G &\xrightarrow{\text{k}_{\text{slow}}} 3G \\ 2B &\underset{k_{fast}}{\overset{k_{slow}}{\rightleftharpoons}} I_B \\ G + I_B &\xrightarrow{\text{k}_{\text{slow}}} 3B \\ 2V &\underset{k_{fast}}{\overset{k_{slow}}{\rightleftharpoons}} I_V \\ B + I_V &\xrightarrow{\text{k}_{\text{slow}}} 3V \\ 2R &\underset{k_{fast}}{\overset{k_{slow}}{\rightleftharpoons}} I_R \\ V + I_R &\xrightarrow{\text{k}_{\text{slow}}} 3R \end{aligned} \tag{3.4}$$

Consider the first two reactions. Two molecules of G combine with one molecule of R to produce three molecules of G. The first step in this process is reversible: two molecules of G can combine, but in the absence of any molecules of R, the combined form will dissociate back into G. So, in the absence of R, the quantity of G will not change much. In the presence of R, the sequence of reactions will proceed, producing one molecule of G for each molecule of R that is consumed. Due to the first reaction $2G \xrightarrow{\text{k}_{\text{slow}}} I_G$, the

transfer will occur at a rate that is super-linear in the quantity of G; this speeds up the transfer and so provides positive feedback. Suppose that the initial quantity of R 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 R, G, B, and V.

One requirement for a clock in synchronous computation is that different clock phases should not overlap. A two-phase clock is used 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, two nonadjacent phases, say R and B in a four-phase $RGBV$ oscillator, are chosen as the clock phases. The scheme for chemical oscillation works well. Figure 3.2 shows the concentrations of R and B as a function of time, obtained through differential equation simulations of the Reactions (3.1), (3.2), (3.3), and (3.4). It may be noted that the two phases R and B are essentially non-overlapping.

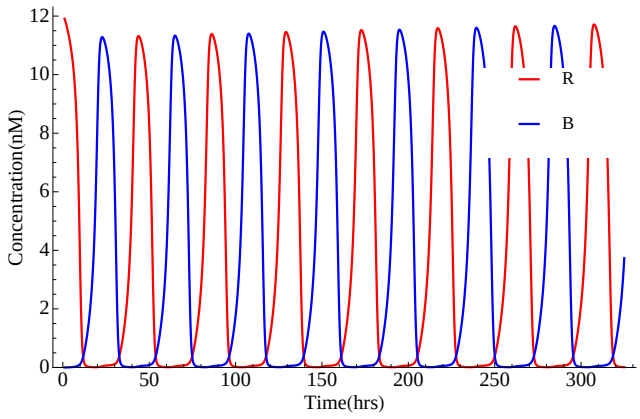


Figure 3.2: simulation results for R and B phases of a four-phase oscillator [2].

The delay and computation elements for the moving average filter in Figure 3.3 are implemented by the reactions in Figure 3.4. As Figure 3.3 shows each delay element, D, is modeled by two molecular types, D and D'. 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.

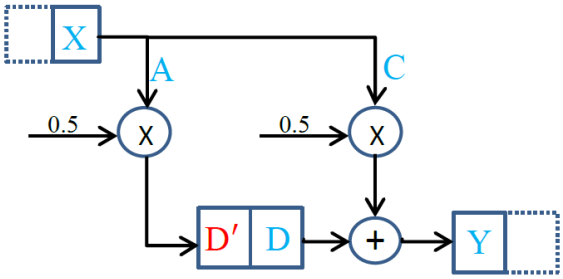


Figure 3.3: Block diagram for synchronous implementation of the moving-average filter [2].

S1	S2
$B + X \xrightarrow{k_{slow}} A + C + B$ $2\,A \xrightarrow{k_{fast}} D'$ $2\,C \xrightarrow{k_{fast}} Y$ $B + D \xrightarrow{k_{slow}} Y + B$	$R + D' \xrightarrow{k_{slow}} D + R$

Figure 3.4: Set of molecular reactions for the synchronous implementation of the moving-average filter [2].

Therefore, in the following phase B, half of the new sample adds with the half of the previous sample stored in D.

3.1.2 Globally-Synchronous Locally-Asynchronous Framework (RGB)

The globally-synchronous locally-asynchronous framework is illustrated in Figure 3.5. 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. Each delay element in this framework is modeled by three molecular

types, namely RGB. Figure 3.6 shows how the computations in asynchronous framework are performed in three phases and how delay elements are implemented using three molecular types R_i, G_i, B_i.

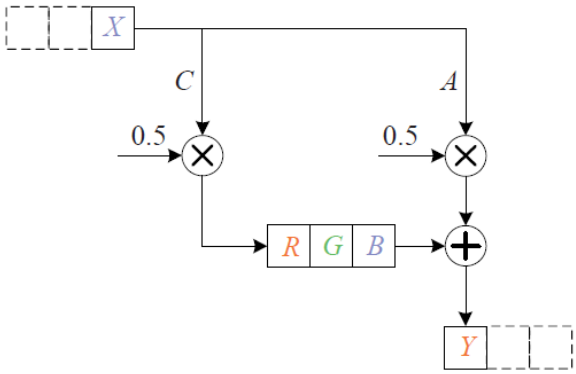


Figure 3.5: Block diagram for the asynchronous implementation of the moving-average filter [2].

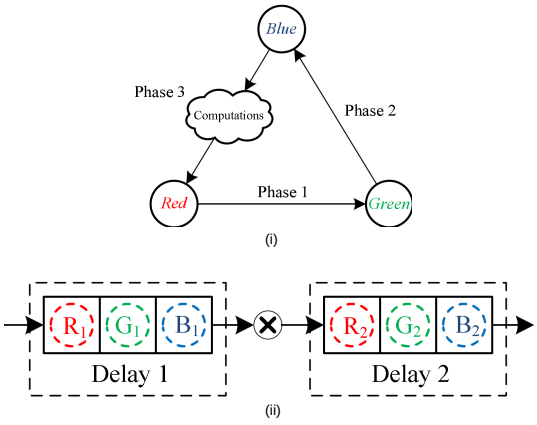


Figure 3.6: (i) Implementing delay elements using the 3-phase asynchronous scheme. (ii) Cascaded delay elements implemented using asychronous scheme [2].

In this framework, the moving-average filter is implemented by the reactions in Figure 3.7. 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 3.7 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 3.7 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.

S1	S2	S3	S4
$g + X \xrightarrow{k_{slow}} A + C$	$2\,R \xrightarrow{k_{fast}} 2R + R'$	$2S_r \xrightarrow{k_{slow}} 2S_r + r$	$R' + X \xrightarrow{k_{fast}} A + C$
$2\,A \xrightarrow{k_{fast}} Y$	$2\,Y \xrightarrow{k_{fast}} 2Y + R'$	$2S_g \xrightarrow{k_{slow}} 2S_g + g$	$G' + R \xrightarrow{k_{fast}} G$
$2\,C \xrightarrow{k_{fast}} R$	$2\,G \xrightarrow{k_{fast}} 2G + G'$	$2S_b \xrightarrow{k_{slow}} 2S_b + b$	$B' + G \xrightarrow{k_{fast}} B$
$b + R \xrightarrow{k_{slow}} G$	$2\,B \xrightarrow{k_{fast}} 2B + B'$	$R' + r \xrightarrow{k_{fast}} R'$	$R' + B \xrightarrow{k_{fast}} Y$
$r + G \xrightarrow{k_{slow}} B$	$2\,X \xrightarrow{k_{fast}} 2X + R'$	$G' + g \xrightarrow{k_{fast}} G'$	
$g + B \xrightarrow{k_{slow}} Y$	$2\,R' \xrightarrow{k_{fast}} \emptyset$	$B' + b \xrightarrow{k_{fast}} B'$	
	$2\,G' \xrightarrow{k_{fast}} \emptyset$		
	$2\,B' \xrightarrow{k_{fast}} \emptyset$		

Figure 3.7: Set of molecular reactions for the asynchronous implementation of the moving-average filter [2].

In the group of reactions shown in the $S2$ column of Figure 3.7 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. 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, and provide global synchronization.

In the group of reactions shown in the $S3$ column of Figure 3.7, 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 reactions shown in the S4 column of Figure 3.7 provide 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. The output Y is produced in the blue-to-red phase.

Although the RGB scheme doesn't have an independent global clock signal it provides a global synchronization by categorizing signals into three phases, so called RGB phases. Many local RGB blocks enable locally-asynchronous computation while global color concentrations, R',G',B', provide global synchronization. In fact, they form a nonsymmetric clock dependent on the signal values of local RGB blocks.

3.2 Fully Asynchronous Scheme

This section presents an asynchronous 4-phase method for implementing discrete-time signal processing algorithms with molecular reactions. The proposed synthesis flow guarantees a conflict-free scheduling for any arbitrary DFG related to a DSP operation including computations and delay elements.

We present a new approach for designing and implementing discrete-time signal processing algorithms with molecular reactions. In the new framework, each delay element of the structure is assigned two molecular types, D_i and D'_i. Transferring signals among delay elements is implemented by transferring concentrations between molecular types assigned to delay elements. The entire computation is completed in four phases. Signal transfers in each phase are triggered by the absence indicators of the other phases. In the proposed scheme, two types of transfer are not allowed. These restrictions are illustrated in Figure 1. First, all outgoing edges of a node must be scheduled in the same phase. Figure 3.8 illustrates a violation of this constraint. Second, if outgoing edges of a node are scheduled at phase "i", none of its incoming edges can be scheduled at phase "$i + 1$". Figure 1(b) illustrates a violation of this constraint.

A synthesis approach for mapping any DSP algorithm to molecular reactions is described as follows:

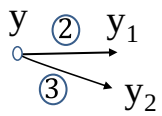
1- Draw the data flow graph (DFG) according to the block diagram of the DSP algorithm. Replace the output node y by nodes y and y', and each delay element D_k by a pair of nodes D_k and D'_k.

2- Assign phase 1 to the outgoing edges of the input node and the outgoing edges of each D'_k node.

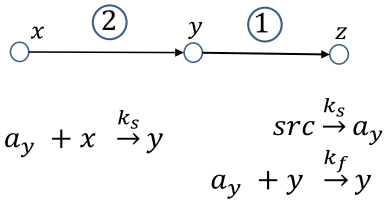
3- Assign phase 2 to the fan out edge of output node (y).

4- All edges between D_k and D'_k are scheduled to phase 3.

5- The outgoing edge of y' is scheduled to phase 4.



(a)



(b)

Figure 3.8: Two types of signal transfer not allowed in our molecular scheme: (a) Outgoing edges scheduled in different times (b) Incoming edge with assigned phase *i+1* for a node with outgoing edge assigned to phase *i*.

6- The molecular reactions for absence indicators, computations, and signal transfers are synthesized according to the assigned scheduling phases.

The proposed 4-phase method is now illustrated by three DSP operations: an FIR filter, a first-order IIR filter and an FFT computation for real-valued signals.

***a. FIR filter*:** Figure 3.9(a) shows a three-tap FIR filter. For simplicity, all tap coefficients are assumed to be 1. The flow graph in Figure 3.9(b) illustrates the phase assignments.

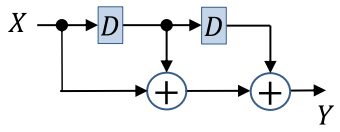
The molecular reactions producing the absence indicator for each phase of this flow graph are described by (3.5). a_i's ($i = 1, 2, 3, 4$) denote the absence indicators for phase i.

$$src \xrightarrow{k_s} a_1 + a_2 + a_3 + a_4$$

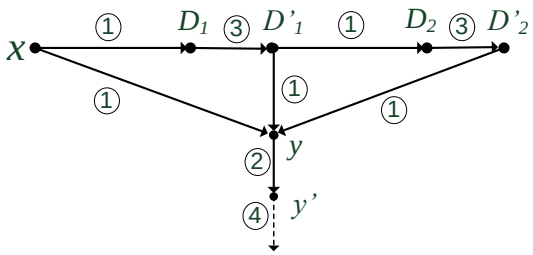
Phase 1:

$$a_1 + x \xrightarrow{k_f} x$$

$$a_1 + D'_1 \xrightarrow{k_f} D'_1$$



(a)



(b)

Figure 3.9: A three-tap FIR filter: (a) Block diagram, (b) Data flow graph and scheduling based on the proposed method.

$$a_1 + D'_2 \xrightarrow{k_f} D'_2$$

Phase 2: $$a_2 + y \xrightarrow{k_f} y \tag{3.5}$$

Phase 3: $$a_3 + D_1 \xrightarrow{k_f} D_1$$

$$a_3 + D_2 \xrightarrow{k_f} D_2$$

Phase 4: $$a_4 + y' \xrightarrow{k_f} y'$$

Then, reactions (3.6) provide the signal transfers associated with related absence indicators. Signal transfers of each phase are enabled by the absence indicator of the previous phase. Note that these are all slow reactions.

Phase 1: $$x + a_4 \xrightarrow{k_s} D_1 + y$$

$$D'_1 + \quad a_4 \xrightarrow{k_s} D_2 + y$$

$$D'_2 + \quad a_4 \xrightarrow{k_s} y$$

Phase 2: $$y + \ a_1 \xrightarrow{k_s} y' \tag{3.6}$$

Phase 3: $$D_1 + \ a_2 \xrightarrow{k_s} D'_1$$

$$D_2 + \ a_2 \xrightarrow{k_s} D'_2$$

According to reactions (3.5) and (3.6), molecules of x, D'_1, and D'_2 transfer in the first phase. After all molecules of x, D'_1, and D'_2 are transferred, phase 2 starts and y is transferred to y'. In phase 3, $D_1$and D_2 transfer, respectively, to D'_1 and D'_2 after all molecules of y transfer to y'. Concentration of $D'_1$and D'_2 are stored to be used for the computation of the next output. Thus, each pair of D_i and D'_i $(i = 1, 2)$ functions as a delay element.

One should notice that the final output y' is collected whenever the absence indicator of the third phase, a_3, is nonzero, implying the third phase has been completed. While the new input is also injected at the same time, it is not used by the system until all molecules of y' are collected.

***b. IIR Filter*:** As another simple DSP operation, we describe the 4-phase method for a simple first-order IIR filter. The block diagram of this filter is shown in Figure 3(a). The filter contains a multiplication by 0.5 inside a feedback loop. From steps 1 to 5 of the synthesis flow, the scheduled 4-phase flow graph for the filter is obtained as shown in Figure 3.10(b).

The set of required absence indicator reactions are illustrated in (3.7).

$$src \xrightarrow{k_s} a_1 + a_2 + a_3 + a_4$$

Phase 1: $$a_1 + x \xrightarrow{k_f} x$$

$$a_1 + D'_1 \xrightarrow{k_f} D'_1$$

Phase 2: $$a_2 + y \xrightarrow{k_f} y \tag{3.7}$$

Phase 3: $$a_3 + D_1 \xrightarrow{k_f} D_1$$

Phase 4: $$a_4 + y' \xrightarrow{k_f} y'$$

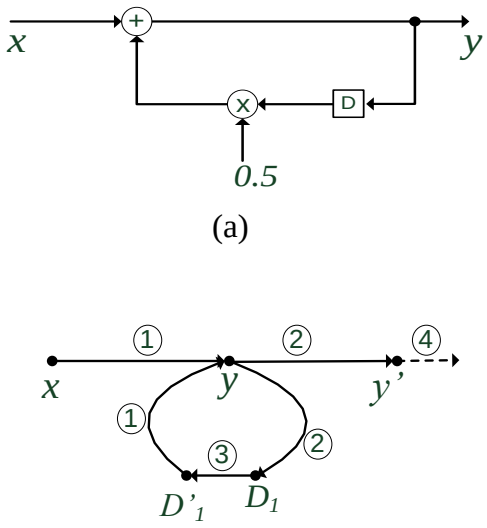


Figure 3.10: An IIR filter: (a) Block diagram, (b) Data flow graph and scheduling for molecular implementation.

Signal transfers and computations are implemented by the reactions in (3.8).

$$\begin{aligned}
&\textit{Phase 1:} && x + a_4 \xrightarrow{k_s} y \\
& && D'_1 + \ a_4 \xrightarrow{k_s} Ty \\
& && 2Ty \xrightarrow{k_f} y \\
&\textit{Phase 2:} && y + \ a_1 \xrightarrow{k_s} y' + D_1 \\
&\textit{Phase 3:} && D_1 + \ a_2 \xrightarrow{k_s} D'_1
\end{aligned} \tag{3.8}$$

Note that the third reaction in (3.8), related to the multiplication by 0.5, fires to completion much faster than the transfer reactions. Each two Ty molecules are immediately combined to produce one y molecule. In other words, D'_1 is transferred to temporary molecules Ty and in the same phase, Ty is multiplied by 0.5 to produce y. The presented method can be easily generalized for DSP algorithms with more than one input/output. The following example illustrates such an algorithm with four inputs and four outputs.

c. *Real-valued FFT (RFFT):* Discrete Fourier transform (DFT) computes the spectral contents of a signal at various frequencies. Fast Fourier transform (FFT) computes DFT using a fast approach when the number of required multiplications can be reduced from $O(N^2)$ to $O(Nlog_2N)$ [13]. We implement FFT, as a canonical algorithm in DSP, with molecular reactions. Molecular implementation of FFT can be used to monitor the frequency content of a protein over time in applications such as drug delivery or cell growth modeling. Like all of the physical signals the concentration of input molecules is a real-valued signal. Therefore, we consider implementation of an FFT system with real-valued inputs, called RFFT. Figure 3.11(a) shows the block diagram for a 4-parallel 8-point RFFT. 8 samples of the input signal, $x(n)$, arrive in two stages. In the first stage, $x(0)$ to $x(3)$ arrive while multiplexers choose their select input $s1$. In the second stage $x(4)$ to $x(7)$ arrive and multiplexers select input $s2$. All of the internal datapaths For an RFFT structure can be real-valued (not complex-valued) datapaths [14]. For more information about RFFT the reader is referred to [15].

The proposed synthesis method assigns scheduling of phases to the flow graph as shown in Figure 3.11(b). Multiplexers in Figure 3.11(a) are implemented as shown in Figure 3.12.

As figure 3.12 shows when $s1$ is nonzero x transfers to z and when $s2$ is nonzero y transfers to z. So signal $s1$ and $s2$ are control signals for multiplexers. In the first stage $s1$ is nonzero while $s2$ is zero. At the end of each stage $s1$ and $s2$ toggle to be ready for the next stage. For this purpose, in the second phase $s1$ transfers to $s2'$ and $s2$ transfers to $s1'$ simultaneously. Then in the fourth phase, $s2'$ transfers to $s2$ and $s1'$ transfers to $s1$. The circular flow graph at the bottom of Figure 3.11(b) represents the toggling of $s1$ and $s2$. This flow graph is implemented by the reactions in (3.5).

$$s1 + a_1 \xrightarrow{k_s} s2'$$

$$s2' + a_3 \xrightarrow{k_s} s2$$

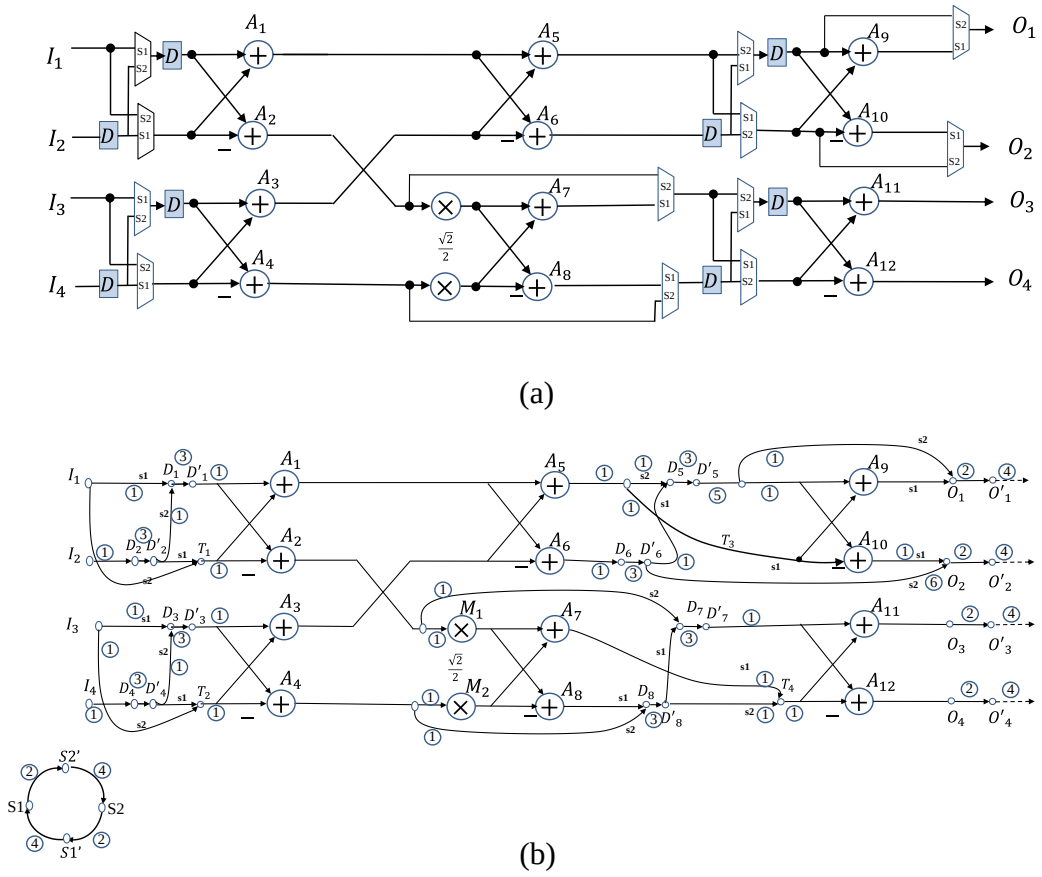


Figure 3.11: 4-parallel 8-point RFFT: (a)Block diagram, (b)Data flow graph and scheduling obtained by the proposed method.

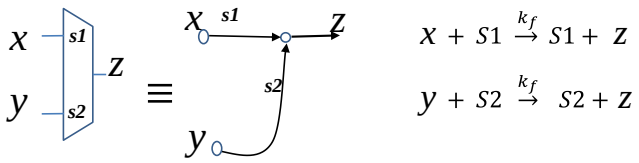


Figure 3.12: Implementation of multiplexer by molecular reactions.

$$s2 + a_1 \xrightarrow{k_s} s1' \tag{3.5}$$

$$s1' + a_3 \xrightarrow{k_s} s1$$

Similar to FIR and IIR filters, the reactions related to the computation, signal transfer, and absence indicators for each phase can be synthesized from the flow graph in Figure 3.11(b). Multiplication by $\frac{\sqrt{2}}{2}$ is implemented using $(\frac{1}{2} + \frac{1}{8} + \frac{1}{16})$ approximation.

In general a signal value can be negative, while concentration of a molecular type can't be negative. Therefore, we use one molecular type for positive and another one for negative part of each signal. We perform computations and signal transfers for each part independently. Finally these two parts cancel out each other and the one with larger concentration determines the sign and value of the signal [5]. For example x_p and x_n represent positive and negative part of signal x, and (3.6) describes positive-negative cancellation reaction by transferring equal concentrations of positive and negative parts to an external sink, ϕ.

$$x_p + x_n \xrightarrow{k_f} \phi \tag{3.6}$$

In order to improve the accuracy and speed of the implemented molecular systems, we add three sets of reactions to them. We call these reactions: *threshold*, *negative feedback*, and *positive feedback* reactions.

Threshold reactions: When a type of molecule exists in the system, its absence indicator is nearly zero but not exactly zero. Although they are very slow a small nonzero value of absence indicator can fire the next phase reactions before completion of the current phase reactions. To avoid this, we initially inject a small concentration of so-called threshold molecules, T_x. The first reaction in (3.11) is a fast reaction. Thus, the absence indicator molecules, a_x, can't fire slow reactions before consuming T_x. In other words, the concentration of a_x must be more than the concentration of T_x, in order to fire signal transfer reactions. When x is present, the second reaction in (3.11) replenishes

the threshold molecules T_x.

$$a_x + T_x \xrightarrow{k_f} \phi$$
$$x + a_x \xrightarrow{k_f} x + T_x \tag{3.11}$$

Negative feedback reactions: The absence indicator molecules, a_x, are produced constantly from the *src*. If x doesn't exist for a while, the concentration of a_x becomes larger and larger. Then when x is produced it takes more time to consume all molecules of a_x. The first reaction in (3.12) limits the increase of a_x concentration. The second reaction in (3.12) controls T_x in the same manner.

$$2a_x \xrightarrow{k_s} a_x + T_x$$
$$2T_x \xrightarrow{k_s} a_x + T_x \tag{3.12}$$

Positive feedback reaction: As shown in Figure 3.13, in positive feedback reactions, destination of a signal transfer, z, is used to speed up the signal transfer y to z. In other words, when the first reaction in Figure 3.13 starts, second reaction speeds up its completion.

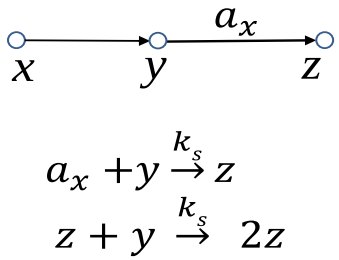


Figure 3.13: Speeding up signal transfers by positive feedback.

3.3 SIMULATION RESULTS

The molecular reactions are mapped to DNA-strand displacement reactions. Critical for mapping to DNA strands, all of our reactions are bimolecular reactions [9]. We

simulated the kinetic of reactions in our designs exploring the mechanism and software tools for DNA-strand displacement developed by Winfree's group at Caltech.

For all DNA simulations for the presented designs we used the following parameters: The initial concentration of auxiliary complexes, $C_{max} = 10^{-5}M$, the maximum strand displacement rate constant, $q_{max} = 10^{6}M^{-1}s^{-1}$, $k_s = 5.56 \times 10^{4}M^{-1}s^{-1}$ and $k_f = 10 \times k_s$. The initial concentration for the source molecular type, src, is set to 0.2 nM. The simulation results for the FIR and IIR filters are shown in Figure 3.14 and Figure 3.15, respectively.

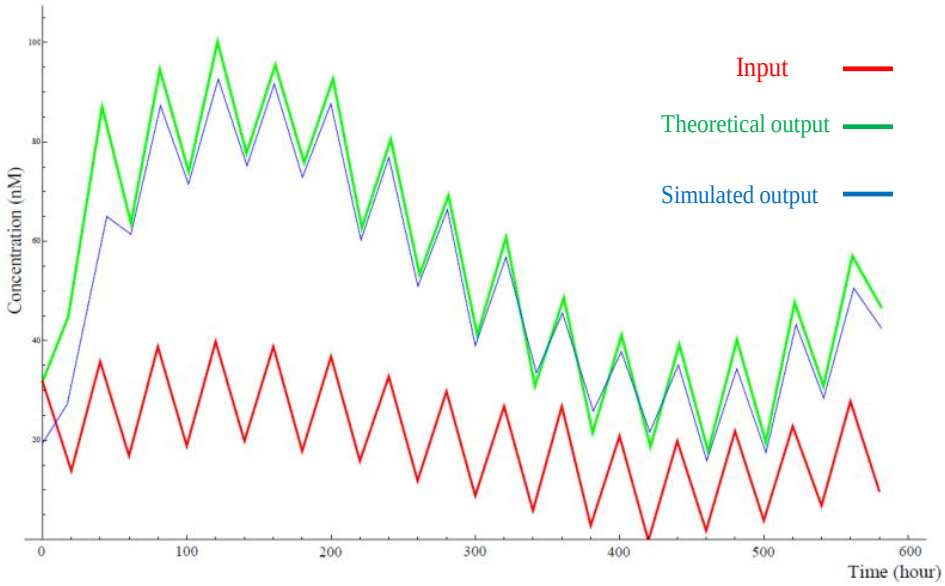


Figure 3.14: Simulation results for FIR filter.

The input is a time-varying signal x with both high frequency and low frequency components. The output is a time-varying concentration y'. For the FIR filter, molecules of x are injected into the system and molecules y' are collected from the system every 20 hours. For IIR filter the injection/collection time is every 30 hours. The Figures show the theoretical outputs as well as simulated outputs. The simulated outputs track the theoretical outputs with some errors. The errors come mainly from the leakage among molecular types. Although it has one more delay element and more number of

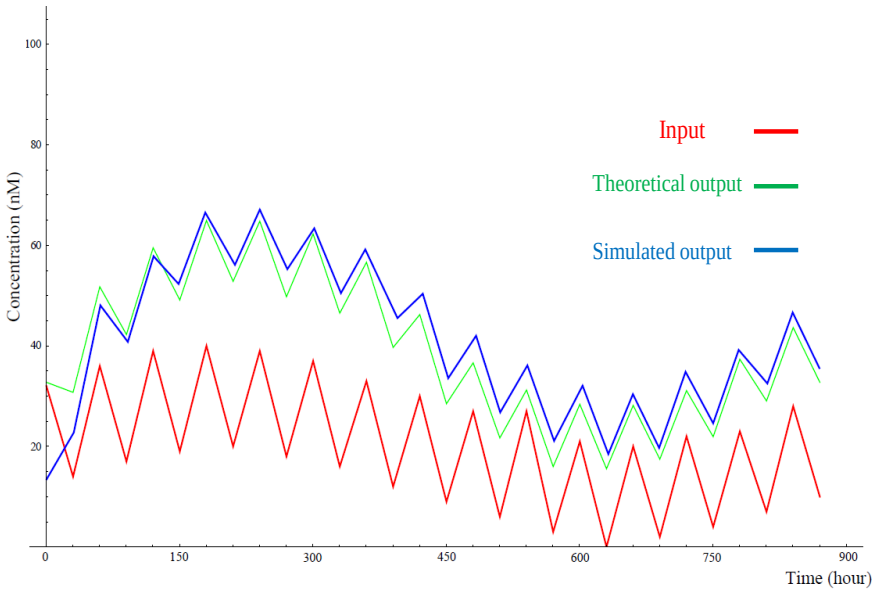


Figure 3.15: Simulation results for IIR filter.

signal transfers, the average relative error for the FIR filter is less than the IIR filter. Generally speaking, IIR filters have higher errors than FIR filters since feedback in such filters leads to error accumulation. Therefore, we considered longer interval between output collections for IIR filter in order to improve its output accuracy.

For an 8-point 4-parallel RFFT implementation, the simulation results are illustrated in Figure 3.16.

Concentrations for the inputs in the first and second stages and their corresponding theoretical outputs are tabulated in Table 1. The injection for inputs and collection for outputs are scheduled once every 250 hours.

Table 2 summarizes the simulation results of the three operations, namely, the FIR filter, the IIR filter, and the RFFT transform in the proposed framework. The errors in this table are computed as the difference between the output value obtained by simulation, o_s, and the theoretical output, o_t. Table 2 shows that as the complexity of operation is increased, or equivalently the number of reactions in the system is increased, the calculation time and the output error increase.

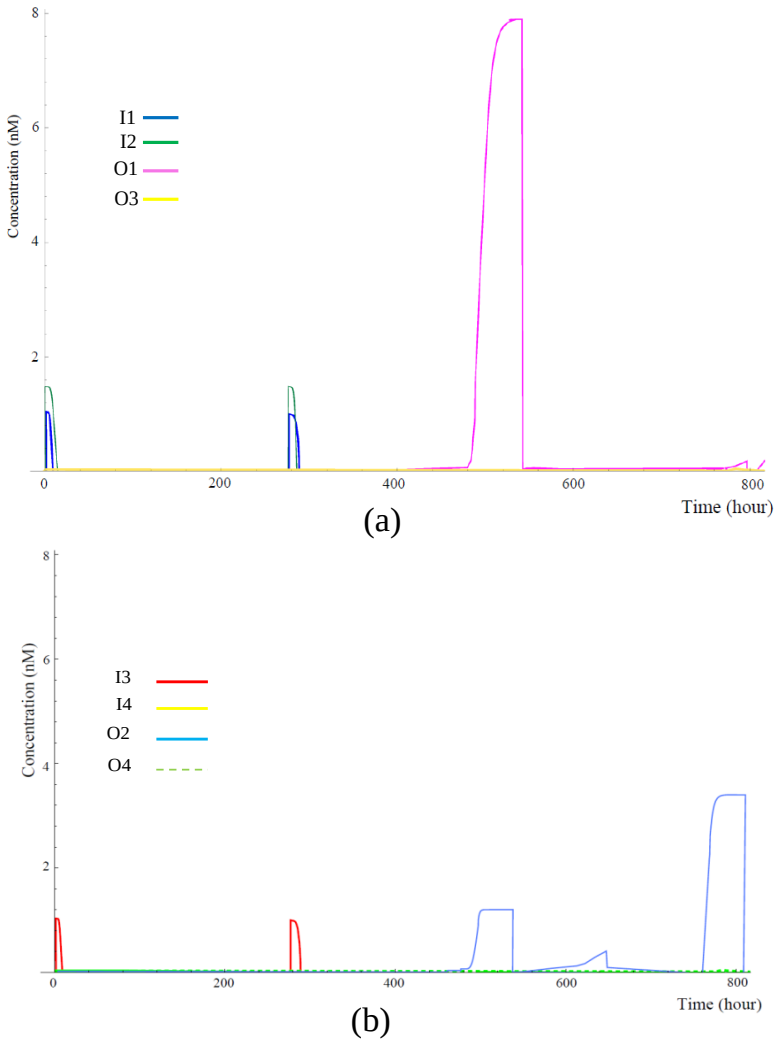


Figure 3.16: Simulation results for 8-point RFFT.

3.4 COMPARISON

To evaluate the performance of our presented method, we compare the RFFT implementation with prior work in Table 3. As the table shows our 4-phase implementation is the fastest one; however, its accuracy is degraded. It is noticeable that even if we allow longer calculation time for 4-phase RFFT, it doesn't improve its output accuracy significantly. The 4-phase and synchronous schemes have less number of reactions and reactants compared to the RGB scheme. However, the number of reactants is not a limiting factor because DNA strands can easily generate a vast number of reactant types.

Although the *in-vitro* simulation results using DNA strands validate the functionality of the method, it is essential to improve the speed and robustness of the method. Future work will be directed towards synthesis of signal processing functions using DNA with one to two orders of magnitude faster sampling rates.

Chapter 4

Mixed-Signal Molecular Computing Systems

Computing or signal processing systems can either be analog or discrete-time. In *analog* processing, the input and output correspond to continuous-time signals. In *discrete-time* processing, the continuous-time signal is first sampled using a sampler, then processed in discrete time steps, and finally converted to a continuous-time signal if necessary by some form of interpolation. If the sampled signal in a discrete-time system is also discretized in amplitude, then it is referred to as a *digital* signal. A digital signal processing (DSP) system requires an analog-to-digital converter (ADC), processing of digital signals and finally a digital-to-analog conversion (DAC). Most information processing systems today store, process or transmit digital information. Discrete-time signal processing provides significantly higher accuracy than continuous-time since the delay elements can be realized with high-precision. In [18], it was recognized that the strength of a molecule was significantly degraded in an analog delay line with increase in the order of the system or the number of delays. In contrast, delay lines implemented in a discrete-time molecular or DNA system do not suffer from significant degradation. Digital processing provides even higher robustness and precise control in processing

the signal in temporal or spectral domain than discrete-time signals. We differentiate discrete-time as sampled in time but continuous in amplitude and digital as sampled in time and discretized in amplitude.

This chapter presents synthesis of molecular computing systems that can be analog, discrete-time or digital. Analog and discrete-time processing of molecular systems have been considered in prior work. Synthesizing molecular and DNA reactions to implement continuous-time linear filters was first presented in [19]. Signal processing systems, implemented as either discrete-time or digital, contain delay elements. Delay elements transfer the molecules from their inputs to outputs without altering the concentration every computation cycle. Delay elements were first synthesized using molecular reactions in [2]. As described in Chapter 3, these systems can operate either in a fully-synchronous manner [13] using a two-phase clock, or in a locally-asynchronous globally-synchronous manner [2, 14], or in a fully-asynchronous manner [20] and [21]. The goal of this chapter is two-fold. First, this chapter presents a review of past work on continuous-time and discrete-time processing systems. Second, a new methodology to synthesize molecular ADCs and molecular DACs are presented. Molecular and DNA implementations of a complete digital processing system using ADC, digital computing and DAC are presented. These molecular designs can be scaled up with respect to their complexity. However, due to the resource limitation in living cells, they are more suitable for *in vitro* implementation, particularly by DNA.

One should notice that discrete-time continuous-amplitude molecular systems are not reviewed in this chapter because they have been discussed in Chapter 3.

4.1 Molecular Continuous-Time Systems

Molecular implementations of continuous-time or analog systems have been described in many past publications [22]-[25]. Study of analog molecular systems is important since it has been proven that computations in living cells are mostly analog [22]-[24].

Analog computations can be implemented with chemical reaction networks (CRNs) efficiently with respect to the number of reactions and molecular species. For example, as presented for the first time in [24] and [25], implementing a molecular adder via analog computation is simple: we have two input concentrations to be added; both are transferred to the same molecular type by means of two independent reactions. In one application of an *in vivo* analog adder, two inputs may correspond to regulating the expressions of a common protein from two independent genetic promoters [24]. Analog multiplication can be simply implemented by two molecular reactions [26]:

$$\begin{array}{rcl} x+y & \xrightarrow{\mathrm{k_1}} & x+y+z \\ z & \xrightarrow{\mathrm{k_2}} & \varnothing \end{array} \tag{4.1}$$

From mass-action kinetics model we have

$$\frac{dz}{dt} = k_1 xy - k_2 z \tag{4.2}$$

where x,y, and z are molecular concentrations of their corresponding molecular types. In the steady-state $\frac{dz}{dt} = 0$, thus, $z = \frac{k_1}{k_2} xy$. The output z represents a scaled version of the product xy. Analog implementation of more complex functions such as square roots and logarithmic additions have been presented in [25]. Implementation of linear continuous-time systems with biochemical reactions has been presented in [19]. We briefly describe this method with an example. Each signal, u, is represented by the difference in concentration between two particular molecular types, u^+ and u^-, where u^+ and u^- are defined as:

$$u^+ = \begin{cases} u & if \quad u > 0 \\ 0 & otherwise \end{cases} \tag{4.3}$$

and

$$u^- = \begin{cases} |u| & if \quad u < 0 \\ 0 & otherwise. \end{cases} \tag{4.4}$$

Any linear continuous-time system can be implemented using three building blocks: integrator, gain and summation. Using mass-action kinetics model, these blocks can be

approximated by a minimal set of chemical reactions, referred as: catalysis, degradation, and annihilation reactions described by (4.5), (4.6), and (4.7), respectively.

$$u^{\pm} \xrightarrow{\gamma} u^{\pm} + y^{\pm} \tag{4.5}$$

$$u^{\pm} \xrightarrow{\gamma} \varnothing \tag{4.6}$$

$$u^{+} + u^{-} \xrightarrow{\eta} \varnothing, \tag{4.7}$$

where γ and $\eta \in R^{+}$. Reaction (4.5) is a concise representation of the following two reactions:

$$\begin{aligned} u^{+} &\xrightarrow{\gamma} u^{+} + y^{+} \\ u^{-} &\xrightarrow{\gamma} u^{-} + y^{-}. \end{aligned} \tag{4.8}$$

This notation is also adopted for other reactions with double superscripts. For each molecular type, an annihilation reaction is necessary to ensure a minimal representation of the molecule. For example, if y is used in a reaction network, the reaction $y^{+}+y^{-} \longrightarrow \varnothing$ should be added.

Integration: Reactions (4.9) implement integration, $y(t) = \int_0^t u(\tau)d\tau + y(0)$ with $t \in R$:

$$u^{\pm} \xrightarrow{\alpha} u^{\pm} + y^{\pm}, \tag{4.9}$$

where $\alpha \in R^{+}$. For these reactions we have

$$\left.\begin{aligned} \tfrac{dy^{+}}{dt} &= \alpha u^{+} \\ \tfrac{dy^{-}}{dt} &= \alpha u^{-} \end{aligned}\right\} \Rightarrow \tfrac{dy}{dt} = \tfrac{dy^{+}}{dt} - \tfrac{dy^{-}}{dt} \tag{4.10}$$

$$= \alpha u^{+} - \alpha u^{-} = \alpha u \Rightarrow y(t) = \alpha \textstyle\int_0^t u(\tau)d\tau + y(0). \tag{4.11}$$

Gain and Summation: The following reactions output a linear combination of the input signals, u_i, with corresponding gain k_i.

$$\begin{aligned} u_i^{\pm} &\xrightarrow{\gamma k_i} u_i^{\pm} + y^{\pm} \\ y^{\pm} &\xrightarrow{\gamma} \varnothing, \end{aligned} \tag{4.12}$$

where y represents the output, $k_i, \gamma \in R^+$ for $i \in 1, 2, ..., n$. In the special case $n = 1$, this chemical representation approximates the gain block, $y = k_1 u_1$ for $k \geq 0$. For $n \geq 2$ this chemical representation approximates the summation block, $y = \sum_{i=1}^{n} k_i u_i$ [19]. Suppose $U(s)$ and $Y(s)$ represent the Laplace transforms of input and output, respectively. Any linear I/O system with the transfer function $\frac{Y(s)}{U(s)} = \frac{B(s)}{A(s)}$ can be approximated by using integration, gain, and summation blocks where $B(s) = b_n s^n + b_{n-1} s^{n-1} + ... + b_1 s + b_0$ and $A(s) = s^m + a_{m-1} s^{m-1} + ... + a_1 s + a_0$ and $m \geq n$. Figure 4.1 illustrates how $\frac{Y(s)}{U(s)}$ can be constructed using these basic building blocks [27, 28].

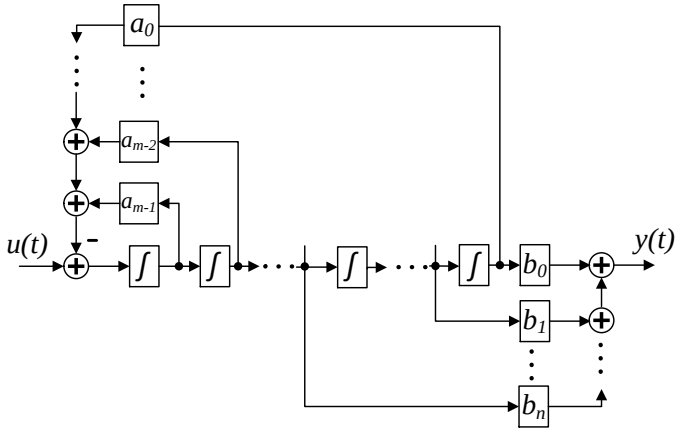


Figure 4.1: Constructing linear I/O systems based on transfer function $\frac{Y(s)}{U(s)} = \frac{B(s)}{A(s)}$, using integration, gain, and summation blocks.

A PI controller has been implemented in [19] using these blocks. Here, we illustrate an example molecular implementation of a first-order low-pass continuous-time filter, shown in Figure 4.2. The transfer function for this filter is $\frac{1}{s+a_0}$. It can be approximated by the following reactions:

$$x(t) = u(t) - a_0 y(t) \rightarrow \left\{ \begin{array}{rcl} y^{\pm} & \xrightarrow{\gamma a_0} & y^{\pm} + x^{\mp} \\ u^{\pm} & \xrightarrow{\gamma} & u^{\pm} + x^{\pm} \\ x^{\pm} & \xrightarrow{\gamma} & \varnothing \\ x^{+} + x^{-} & \xrightarrow{\eta} & \varnothing \\ u^{+} + u^{-} & \xrightarrow{\eta} & \varnothing \end{array} \right. \tag{4.13}$$

$$\frac{dy}{dt} = x(t) \rightarrow \left\{ \begin{array}{rcl} x^{\pm} & \xrightarrow{\gamma} & x^{\pm} + y^{\pm} \\ y^{+} + y^{-} & \xrightarrow{\eta} & \varnothing \end{array} \right. \tag{4.14}$$

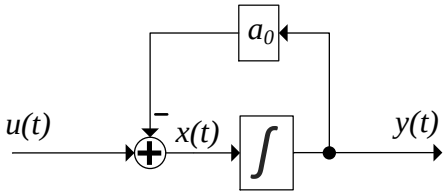


Figure 4.2: A first order low-pass continuous-time filter.

4.2 Digital Sensing and Computing Molecular Systems

Although analog computing systems are important due to their efficiency and their application in *in vivo* systems, digital computing systems are more robust [29, 24, 30]. In fact, regardless of the implementation technology, the fundamental reason for the robustness of the digital computation lies in information theory: information is coded across many 1-bit-precise interacting computational channels in the digital approach but on one channel in the analog approach [24].

Although complex molecular digital systems may be impractical today, these will be practical in near future as synthetic biology is seeing remarkable progress for synthesizing more complex systems *in vitro* especially from DNA. As a practical *in vitro* example, implementation of a scalable digital system, so called *seesaw gates*, with DNA strand-displacement reactions have been used to implement simple logical AND/OR gates, and 2-bit-precise square roots in [30].

Roughly speaking, in a digital molecular system, absence or existence of a molecular type defines whether the related signal is logically '0' or '1', respectively. More precisely, if the concentration of a molecular type is close to 0 nM it represents logical '0', while if it is close to a distinguishable nonzero value, it represents logical '1'. In this chapter,

for *in vitro* DNA implementations, we consider concentrations near 1 nM as the logical value '1' and near 0 nM as logical value '0'.

Molecular digital systems require molecular analog-to-digital conversion (ADC). This section, presents a new molecular implementations of ADCs and DACs. Figure 4.3 illustrates a complete digital system.

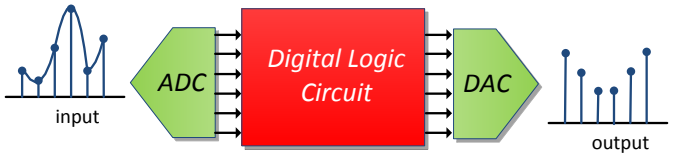


Figure 4.3: Block diagram of a general system developed in this chapter.

We present molecular implementations of a k-bit analog to digital converter and a k-bit digital to analog converter. We also review the molecular implementation of basic digital logic gates. Using these gates, we demonstrate a 3-bit molecular binary adder including two ADCs required to sample and digitize the two input operands and a DAC to output an analog signal. A DNA implementation of the complete system is also demonstrated in Section 4.3. It can be noted that all of the molecular reactions are rate-independent. In other words, no matter what the speed rates of the reactions are and how they may change during the computation, the steady-state concentrations compute the correct desired outputs.

4.2.1 Analog to Digital Converter (ADC)

This subsection describes molecular implementation of analog to digital converter. A 3-bit example is considered. Let the input molecular type, i, have an analog concentration between 0 nM and 8 nM. The output is a 3-bit digital number $x = x_2x_1x_0$. Each bit is considered as logical '0' if its concentration is approximately 0 nM and logical '1' if its concentration is approximately 1 nM.

We start with the most significant bit, x_2. This bit should be set to 1 when i is larger than 4 nM and to zero when i is less than 4 nM. Reactions (4.15) implement a one-bit comparator that determines x_2. The initial concentration of T_2 represents the threshold for the comparator which is set to 4 nM.

$$\begin{aligned} i + T_2 &\longrightarrow w_2 \\ i + x_{2n} &\longrightarrow x_2 + i \\ T_2 + x_2 &\longrightarrow x_{2n} + T_2 \end{aligned} \tag{4.15}$$

In the first reaction, i and T_2 molecules combine and the one with larger initial concentration remains and the other one vanishes. The first reaction is independent of the second and third reactions because i and T_2 remain unaltered in the second and third reactions. However, activation of the second or third reactions depends on the outcome of the first reaction. After completion of the first reaction only one of the second or third reactions is active. If i is larger than T_2, the third reaction stops firing while the second reaction transfers all molecules of x_{2n} to x_2. Alternately, if i is less than T_2, second reaction stops and third reaction transfers x_2 to x_{2n} completely. x_2 and x_{2n} are initialized to 0 nM and 1 nM, respectively. Note that in general for a k-bit ADC, each bit, i.e., x_j where $j = 0, 1, ..., k-1$, is modeled by two molecular types, i.e., x_j and x_{jn}, called the bit and its complement molecular types. All of the x_j species are initialized to 0 nM and x_{jn} species are initialized to 1 nM. Furthermore, for each j, the total concentration of x_j and x_{jn}, is constantly 1 nM, i.e., if the concentration of x_j is C, then the concentration of x_{jn} is $(1 - C)$, both in nM.

Table 4.1 shows the final concentrations for i, x_2 and w_2 after completion of Reactions (4.15). i_0 denotes the initial concentration of i. If $i_0 > T_2$ then i can be used to compute the second bit of x, *i.e.*, x_1. If $i_0 < T_2$ then w_2 can be used to determine x_1. Reactions (4.16) and (4.17) determine x_1 for the above two cases. The initial concentrations for both threshold molecules, T_1 and T_1', are equal to 2 nM. Similar to Reactions (4.15), the first three reactions of (4.16) implement a one-bit comparator. However, here, the molecular concentration of i and T_1 are compared to determine x_1

when x_2 is nonzero. This is equivalent to comparing initial i_0 to 6 nM. Similarly the first three reactions of (4.17) compare w_2 and T_1' to determine x_1 when x_2 is zero. This is equivalent to comparing initial i_0 to 2 nM.

Table 4.1: Stable concentration of molecules i, x_2, and w_2 after completion of Reactions (4.15).

	i	w_2	x_2
$i_0 < 4$	0	i_0	0
$i_0 > 4$	$i_0 - 4$	4	1

$$\begin{aligned} x_2 + i + T_1 &\longrightarrow w_1 + x_2 \\ x_2 + i + x_{1n} &\longrightarrow x_1 + i + x_2 \\ x_2 + T_1 + x_1 &\longrightarrow x_{1n} + T_1 + x_2 \\ x_{2n} + w_1 &\longrightarrow i + T_1 + x_{2n} \end{aligned} \tag{4.16}$$

$$\begin{aligned} x_{2n} + w_2 + T_1' &\longrightarrow w_1' + x_{2n} \\ x_{2n} + w_2 + x_{1n} &\longrightarrow x_1 + w_2 + x_{2n} \\ x_{2n} + T_1' + x_1 &\longrightarrow x_{1n} + T_1' + x_{2n} \\ x_2 + w_1' &\longrightarrow w_2 + T_1' + x_2 \end{aligned} \tag{4.17}$$

Before the concentration of x_2 reaches its stable value, both x_2 and x_{2n} may have nonzero concentrations and both sets of Reactions (4.16) and (4.17) can be fired. The fourth reactions of (4.16) and (4.17) are added to undo undesired reactions fired during the transient time. For example, when the final concentration of x_2 is zero the fourth reaction of (4.16) transfers w_1 back to i and T_1 in order to undo the first reaction. The initial concentrations for x_1 and x_{1n} are 0 nM and 1 nM, respectively. After x_1 and x_{1n} are stabilized to their final concentrations, depending on the initial value of i, one of them has the concentration of 1 nM and the other 0 nM.

Except i, none of the molecular types participating in Reactions (4.15) is altered by Reactions (4.16) and (4.17). However, Reactions (4.16) and (4.17) need the final concentrations of x_2 and x_{2n} from Reactions (4.15). Thus, the concentrations of molecules

of Reactions (4.16) and (4.17) reach stable values after reactions in (4.15) are completed. For different values of i_0, Table 4.2 shows the final concentrations after Reactions (4.16) and (4.17) are completed.

Table 4.2: Stable molecular concentrations after completion of Reactions (4.16) and (4.17).

	i	w_2	w_1	w_1'	x_2	x_1
$i_0 < 2$	0	0	0	$\underline{i_0}$	0	0
$2 < i_0 < 4$	0	$\underline{i_0 - 2}$	0	2	0	1
$4 < i_0 < 6$	0	4	$\underline{i_0 - 4}$	0	1	0
$6 < i_0$	$\underline{i_0 - 6}$	4	2	0	1	1

Finally, in order to determine the least significant bit (LSB) of x, i.e., x_0, depending on i_0's value, the molecular types underlined in Table 4.2 are used. For each range of i_0, the concentration of its related molecular type is compared to 1 nM to determine x_0. For example when $i_0 > 6$, Reactions (4.18) are used to determine x_0. The initial concentration of threshold molecules T_0 is 1 nM. Because both x_2 and x_1 are nonzero for $i_0 > 6$, the first three reactions compare i with 1 nM. It is equivalent to comparing i_0 with 7 nM. That is to say, for $i_0 > 6$, x_0=1 nM if i_0 >7 nM and x_0=0 nM if i_0 <7 nM. The last two reactions of (4.18) are used to undo the undesirable combination of i and T_0 during the transient time when any of x_2 or x_1 is zero.

$$\begin{aligned} x_2 + x_1 + i + T_0 &\longrightarrow w_0 + x_2 + x_1 \\ x_2 + x_1 + i + x_{0n} &\longrightarrow x_0 + i + x_2 + x_1 \\ x_2 + x_1 + T_0 + x_0 &\longrightarrow x_{0n} + T_0 + x_2 + x_1 \\ x_{2n} + w_0 &\longrightarrow i + T_0 + x_{2n} \\ x_{1n} + w_0 &\longrightarrow i + T_0 + x_{1n} \end{aligned} \tag{4.18}$$

Similarly for each range of i_0 five reactions are used to determine x_0. Due to space limit, these three sets of reactions, each containing five reactions, are not listed here.

The number of bits or the resolution of ADC can be increased by adding the required comparisons and their related undo reactions. In general for k-bit ADC

$2^{k+1} + 2(k-1)$ molecular types are required while the number of required reactions is $\sum_{j=1}^{k} (j+2)2^{j-1} = (k+1)2^k - 1$. The precision (sensitivity) of ADC depends on its acceptable input range and the number of its output bits.

Figure 4.4 shows results for the mass-action kinetic model simulation of the proposed ADC for different values of i_0 .

4.2.2 Molecular Digital Logic Circuits

In this section we demonstrate how digital designs can be implemented by molecular reactions. We describe molecular implementations of simple logic AND/OR/XOR gates, a binary adder, and a square-root unit. The method we use here for implementing logical gates is similar to the method presented in [12]. However, in [12] three regulation bit operation reactions are needed for each bit, Whereas these reactions are not required in our complete system implementation due to the self-regulated bits output by the proposed ADC. Here, self-regulated means for each bit only the related molecular type, x_j, or its complement, x_{jn}, but not both, has stable non-zero concentration.

Logic Gates

We only consider two-input gates AND, OR, and XOR. Gates with more than two inputs can be easily implemented by cascading two-input gates. Let X and Y denote the inputs of a gate and Z the output.

AND Gate: We start with an AND gate. The output of a logical AND gate is '1' only if both inputs are '1'. It means that if either X='0' or Y='0' then the output Z should be zero. In other words, when concentration of x_n or y_n, i.e., complement molecular types of inputs, is nonzero molecules of z should be transferred to z_n in order to set Z='0'. This can be implemented by Reactions (4.19).

$$\begin{aligned} x_n + z &\longrightarrow x_n + z_n \\ y_n + z &\longrightarrow y_n + z_n. \end{aligned} \tag{4.19}$$

When both x and y have stable nonzero concentrations, all molecules of z_n should be transferred to z in order to set Z='1'. This can be implemented by Reactions (4.20).

$$\begin{aligned} x + y &\longrightarrow x + y + z' \\ 2z' &\longrightarrow \varnothing \\ z' + z_n &\longrightarrow z. \end{aligned} \tag{4.20}$$

In the first reaction of (4.20), x combines with y to generate z', an indicator that Z should be set to '1'. z' is transferred to an external sink, denoted by $\varnothing$, in the second reaction. (This could be a waste type whose concentration we do not track.) When molecules of both x and y are present, these reactions maintain the concentration of z' at an equilibrium level. When either x or y is not present, z' gets cleared out. In the last reaction, z' transfers z_n to z.

One should note that the input concentrations don't change in logic computations. This enables the outputs of the ADC to be input to other logic gates if needed.

OR Gate: The output of an OR gate is '1' if any of its inputs is '1'. For molecular implementation it means that if either x or y has nonzero concentration then all molecules of z_n should be transferred to z. It is implemented by Reactions (4.21). In the other case, i.e., when both inputs have zero concentrations, molecules of z should be transferred to z_n as implemented by Reactions (4.22).

$$\begin{aligned} x + z_n &\longrightarrow x + z \\ y + z_n &\longrightarrow y + z. \end{aligned} \tag{4.21}$$

$$\begin{aligned} x_n + y_n &\longrightarrow x_n + y_n + z' \\ 2z' &\longrightarrow \varnothing \\ z' + z &\longrightarrow z_n. \end{aligned} \tag{4.22}$$

XOR Gate: The output of a two-input XOR gate is '1' when inputs are complements of each other. In molecular implementation it means that when either x and y_n or x_n and y have nonzero concentrations, molecules of z_n should be transferred to z as

implemented by Reactions (4.23). For the inputs with the same logical level the output should set to zero and molecules of z should be transferred to z_n. This is implemented by Reactions (4.24).

$$\begin{aligned} x_n + y &\longrightarrow x_n + y + z' \\ x + y_n &\longrightarrow x + y_n + z' \\ 2z' &\longrightarrow \varnothing \\ z' + z_n &\longrightarrow z. \end{aligned} \tag{4.23}$$

$$\begin{aligned} x_n + y_n &\longrightarrow x_n + y_n + z'_n \\ x + y &\longrightarrow x + y + z'_n \\ 2z'_n &\longrightarrow \varnothing \\ z'_n + z &\longrightarrow z_n. \end{aligned} \tag{4.24}$$

NAND, NOR, and XNOR gates can be implemented by exchanging z and its complement in the transfer reactions, z_n in the opposite directions of those of the AND, OR, and XOR gates, respectively.

Binary Adder

By cascading AND, OR, and XOR gates we implement more complex digital systems such as a 3-bit adder. The adder consists of one half adder (HA) for the LSB and two full adders (FA) as shown in Figure 4.5a. Internal schematics of HA and FA are shown in Figure 12b. A general n-bit adder can be easily implemented by extending 3-bit adder using additional FAs for new bits.

Cascaded gates for the adder are implemented by molecular reactions presented in Section IV.B. However, other molecular logic gates such as seesaw gates [30] can also be used. In order to verify the functionality of the 3-bit adder we implement the structure shown in Figure 4.6.

Two analog concentrations, x and y, are converted to two 3-bit digital data using the proposed ADC. These two digital numbers are added using the 3-bit adder. The output, $s = s_3 s_2 s_1 s_0$, is a 4-bit digital number representing the digital sum of x and y.

Figure 4.7 shows the simulation results for different concentrations of inputs, x and y.

Square-root Unit

As another example of digital computing, we implement square-root of a 4-bit number. Figure 4.8 shows the schematic of its circuit. In Figure 4.8, the three-input NAND gate can be implemented by cascading a two-input AND gate with a two-input NAND gate. However, it is more efficient to implement three-input NAND by reactions (4.25). In these reaction x_1, x_2, and x_3 are inputs and y is the output.

$$\begin{aligned} x_{1n} + y_n &\longrightarrow x_{1n} + y \\ x_{2n} + y_n &\longrightarrow x_{2n} + y \\ x_{3n} + y_n &\longrightarrow x_{3n} + y \\ x_1 + x_2 &\longrightarrow x_1 + x_2 + x_{12} \\ x_{12} + x_3 &\longrightarrow x_3 + y' \\ 2y' &\longrightarrow \varnothing \\ y' + y &\longrightarrow y_n \end{aligned} \tag{4.25}$$

The strategy used for the direct implementation of three-input NAND in (4.25) is similar to that of two-input NAND.

Figure 4.9 shows the simulation results for the square root circuit implemented by molecular reactions.

4.2.3 Digital to Analog Converter (DAC)

After performing computations in digital form, in order to convert the computed signal to its analog form, a DAC is required. Using recombinase-based logic and memory, a DAC has been implemented in [31]. For this DAC various digital combinations of the input inducers result in multiple levels of analog gene expression outputs on the basis of the varying strengths of the promoters used and the sum of their respective outputs.

This section presents molecular implementations of a k-bit DAC with controlling the impact of each bit on the analog output concentration. Reactions (4.26) show a 1-bit template for implementing DAC.

$$\begin{aligned} x_j + V_j &\longrightarrow out + x_j + M_j \\ out + x_{jn} + M_j &\longrightarrow x_{jn} + V_j \end{aligned} \tag{4.26}$$

where x_j and x_{jn}, respectively, represent the input bit and its complement molecular type. *out* is the analog output of DAC with initial concentration of zero. Molecular type V_j denotes the value of the input bit. In other words, it defines the amount of concentration that is added to the output if input bit, x_j, is nonzero. If x_j is the LSB then V_j is initialized to 1 nM and if it is the bit next to the LSB then V_j is initialized to 2 nM and so on.

Even when the stable value of x_j is zero, during the transient state x_j may have nonzero concentration. The second reaction of (4.26) prevents undesired output increase due to the nonzero concentration of x_j in transient state. M_j controls the amount of deducted concentration from the output such that this amount is the same as the amount added to output undesirably during the transient state. In other words, without M_j, the second reaction continues transferring *out* molecules to V_j during the steady-state. However, this degrades the effects of other bits on the DAC's output, since the molecular type *out* is common for all bits. The initial concentration for M_j is zero.

The 1-bit template presented here can be easily extended to a $k-bit$ DAC; for each additional bit, one instance of Reactions (4.26) is added. Therefore, to construct a k-bit DAC, a chemical reaction network including k copies of the 1-bit template are used with proper initial values of V_j. As an example, Reactions (4.27) illustrate a 4-bit DAC using the proposed template. The initial concentrations of V_0, V_1, V_2, and V_3 are 1, 2, 4, and

8 nM, respectively.

$$\begin{aligned}
x_0 + V_0 &\longrightarrow out + x_0 + M_0 \\
out + x_{0n} + M_0 &\longrightarrow x_{0n} + V_0 \\
x_1 + V_1 &\longrightarrow out + x_1 + M_1 \\
out + x_{1n} + M_1 &\longrightarrow x_{1n} + V_1 \\
x_2 + V_2 &\longrightarrow out + x_2 + M_2 \\
out + x_{2n} + M_2 &\longrightarrow x_{2n} + V_2 \\
x_3 + V_3 &\longrightarrow out + x_3 + M_3 \\
out + x_{3n} + M_3 &\longrightarrow x_{3n} + V_3
\end{aligned} \tag{4.27}$$

4.2.4 A complete molecular digital System

We now illustrate molecular implementation of a digital adder where concentrations of two analog molecules x and y are converted to 3-bit digital, then added using a binary adder, and the 4-bit output is converted to an analog value s. Two molecular ADCs, a molecular digital adder, and a molecular DAC are used to construct a complete system as shown in Figure 6.1. The functionality of the complete molecular system is verified.

Figure 4.11 shows the simulation results for the complete system illustrated in Figure 6.1 for different input concentrations.

4.3 DNA Implementation

This section describes mapping of the molecular reactions to DNA. We illustrate mapping the complete digital adder of Section IV.D including ADC, adder and DAC to DNA strand displacement reactions.

Considering each strand (single or double) of DNA as a molecule, it is possible to implement CRNs with DNA strand-displacement mechanism. For example Figure 4.12 shows DNA strand-displacement primitive for implementing $A + B \underset{r}{\overset{f}{\rightleftharpoons}} C + D$.

Toehold 1 of strand A starts binding to its complement toehold 1^* of B. Then branch migration happens and domain 2 of A displaces domain 2 of strand $2-3$. Finally, toehold 3 and 3^* are separated and two new strands (molecules), C and D, are produced.

A general method of mapping CRNs to DNA strand-displacement reactions has been presented in [11] by Soloveichik, et. al. In their method based on the number of reactants a chemical reaction is converted to a series of DNA strand-displacement reactions similar to Figure 4.12. Similarly, for our design we generate the corresponding DNA reactions and simulate the system using the kinetic differential equations to characterize the behavior of the system.

The initial concentrations of auxiliary complexes is set to $C_{max} = 10^{-5}$M, and the maximum strand displacement rate constant is $q_{max} = 10^6$ M^{-1} s^{-1}. For all of the reactions the rate constant is considered as $10^5 M^{-1} S^{-1}$. Figure 4.13 shows the ODE simulation results for the DNA implementation of the complete system illustrated in Figure 6.1 for different inputs.

4.4 Discussion and Concluding Remarks

This chapter presented methodologies for implementing continuous-time and digital processing with molecular reactions. Several examples are presented to illustrate the approaches presented in the chapter.

Although pertaining to biology, the contributions of this chapter are neither experimental nor empirical; rather they are constructive and conceptual. We design robust digital logic with molecular reactions. For the molecular digital systems, our designs do not depend on specific reaction rates; the computation is accurate for a wide range of rates. This is crucial for mapping the design to DNA substrates.

Intense efforts by the synthetic biology community have been devoted to the implementation of computation and logical functions with genetic regulatory elements [32]-[36]. For example design of robust logical circuits using chemically wired cells have

been presented in [29] for single logic gates. Also genetic circuits consisting of multi-layer logical gates have been implemented in single cell in [37]. Yet, progress seems to have stalled at the complexity level of circuits with perhaps 7-15 components. In fact, *in vivo* engineering of such circuits is full of 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. Thus, due to their complexity, systems presented in this chapter are more likely to be physically realizable *in vitro* than *in vivo*.

The impetus of the field is not computation *per se*; chemical systems will never be useful for number crunching. Rather the field aims for the design of custom, embedded biological "sensors" and "controllers" – viruses and bacteria that are engineered to perform useful tasks *in situ*, such as cancer detection and drug therapy. Exciting work in this vein includes [38, 39, 40, 41].

One should notice that there is quantization error in the ADC component. This is similar to the quantization error for other types of ADC usually used in digital signal processing systems [42]. The error can decrease the accuracy of system. The quantization error can be reduced by increasing the ADC resolution and, consequently, increasing the number of bits of ADC and DAC components.

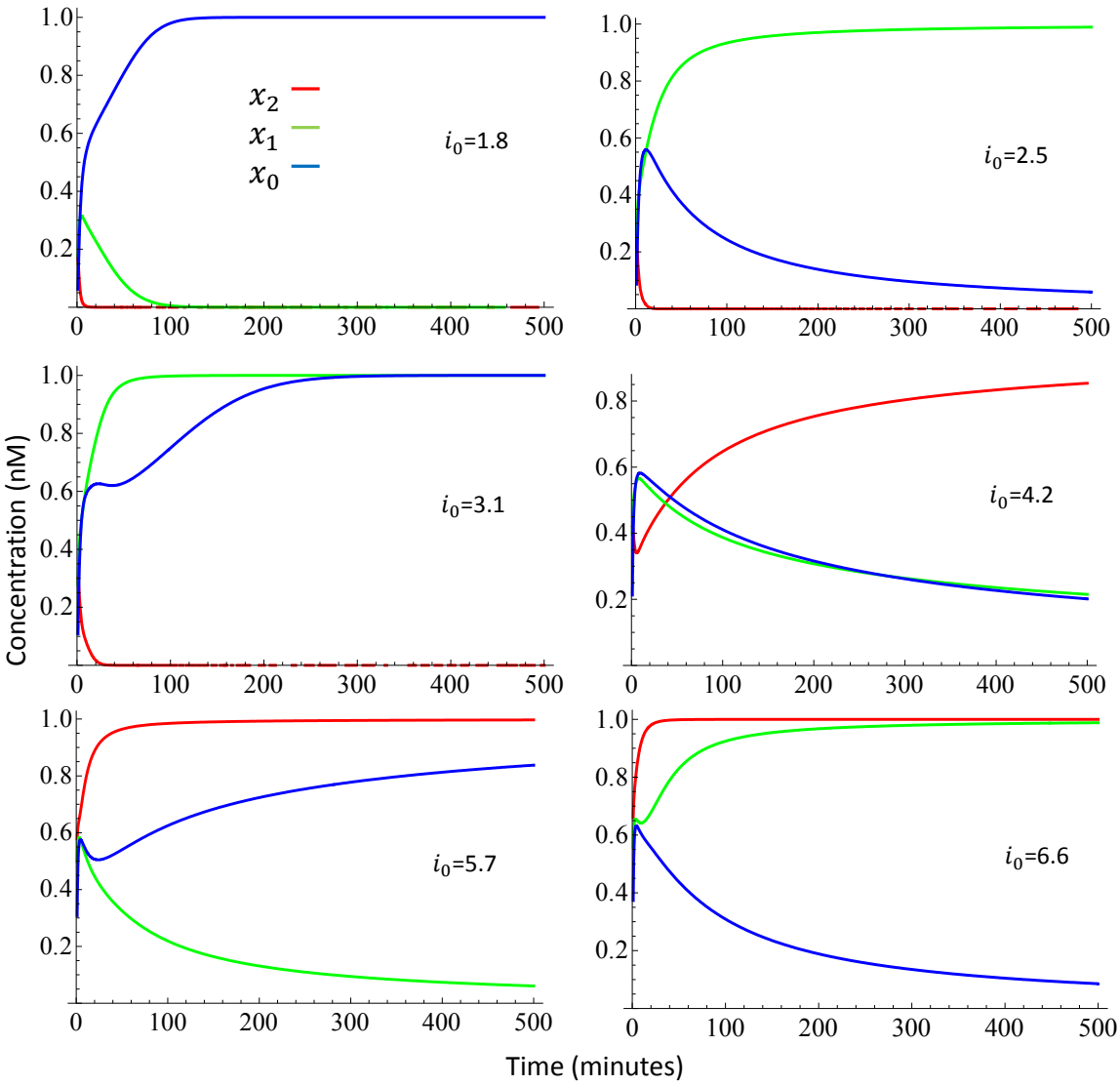


Figure 4.4: Simulation results of 3-bit molecular ADC for different input concentrations.

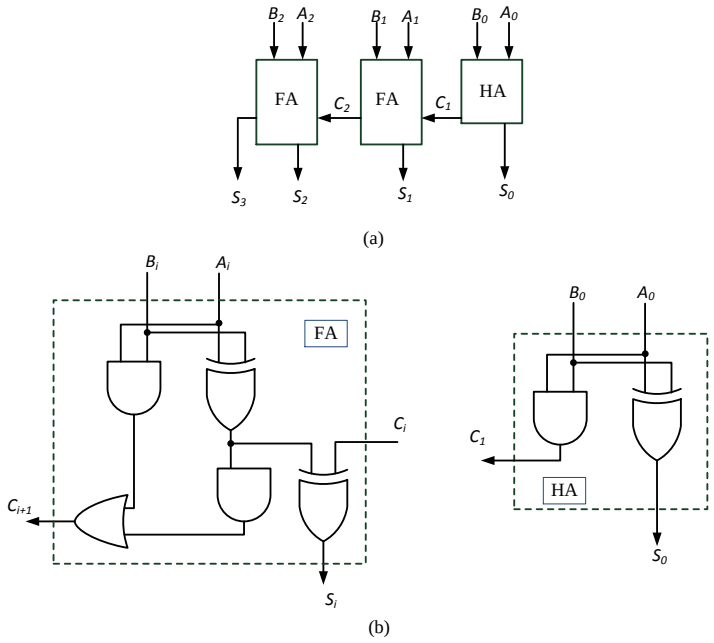


Figure 4.5: Schematic of the 3-bit adder; (a) Block diagram, (b) Internal circuits for HA and FA blocks.

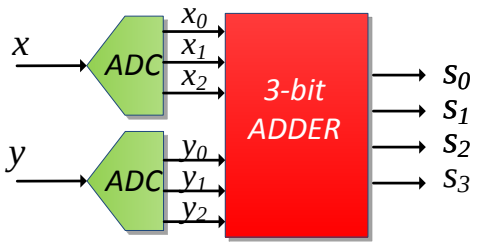


Figure 4.6: Block diagram of the system for verifying molecular 3-bit adder.

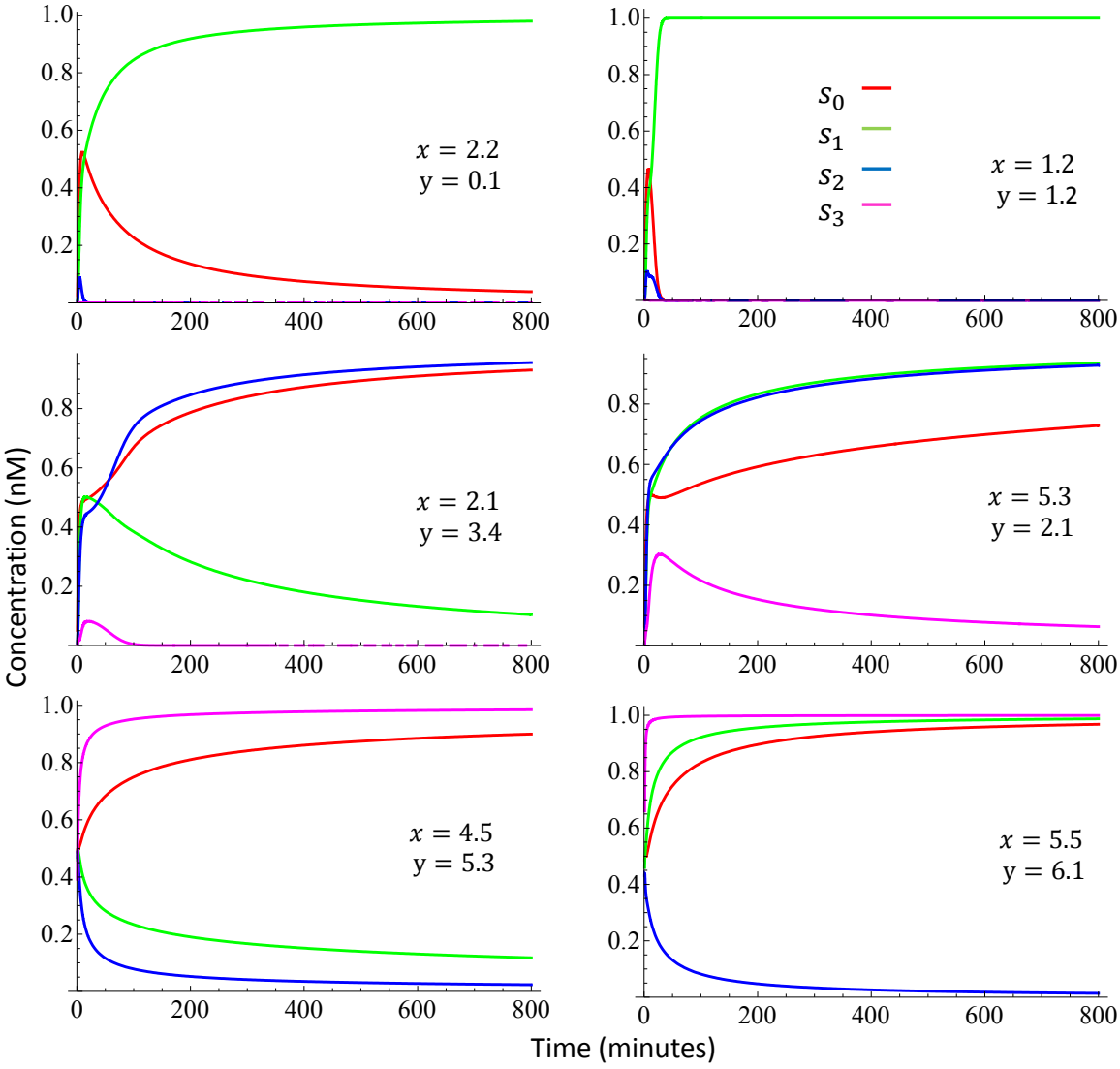


Figure 4.7: Simulation results of the molecular implementation of the system shown in Figure 4.6.

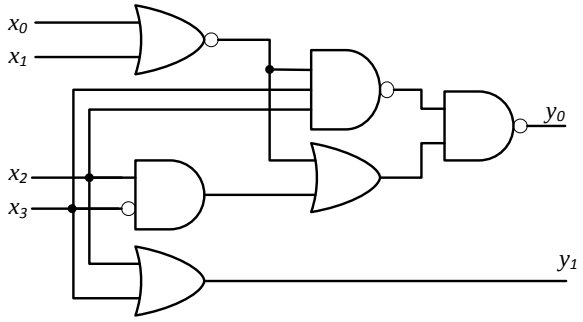


Figure 4.8: Schematic for 4-bit Square-root unit.

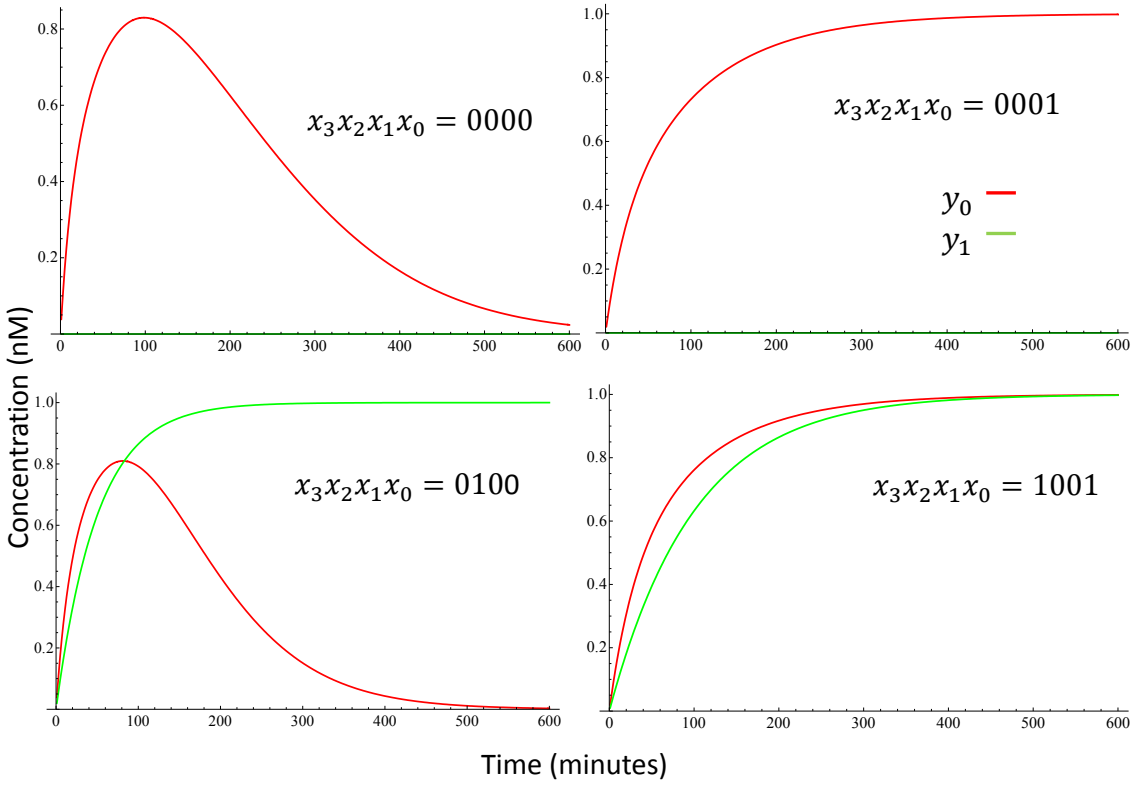


Figure 4.9: Kinetics simulations that compute the Square-root of 0, 1, 4, and 9 using the molecular implementation of unit shown in Figure 4.8.

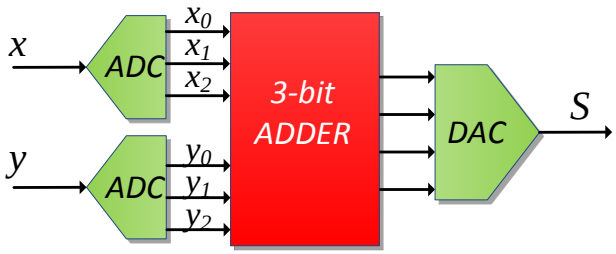


Figure 4.10: Block diagram of a simple prototype developed and verified in this research.

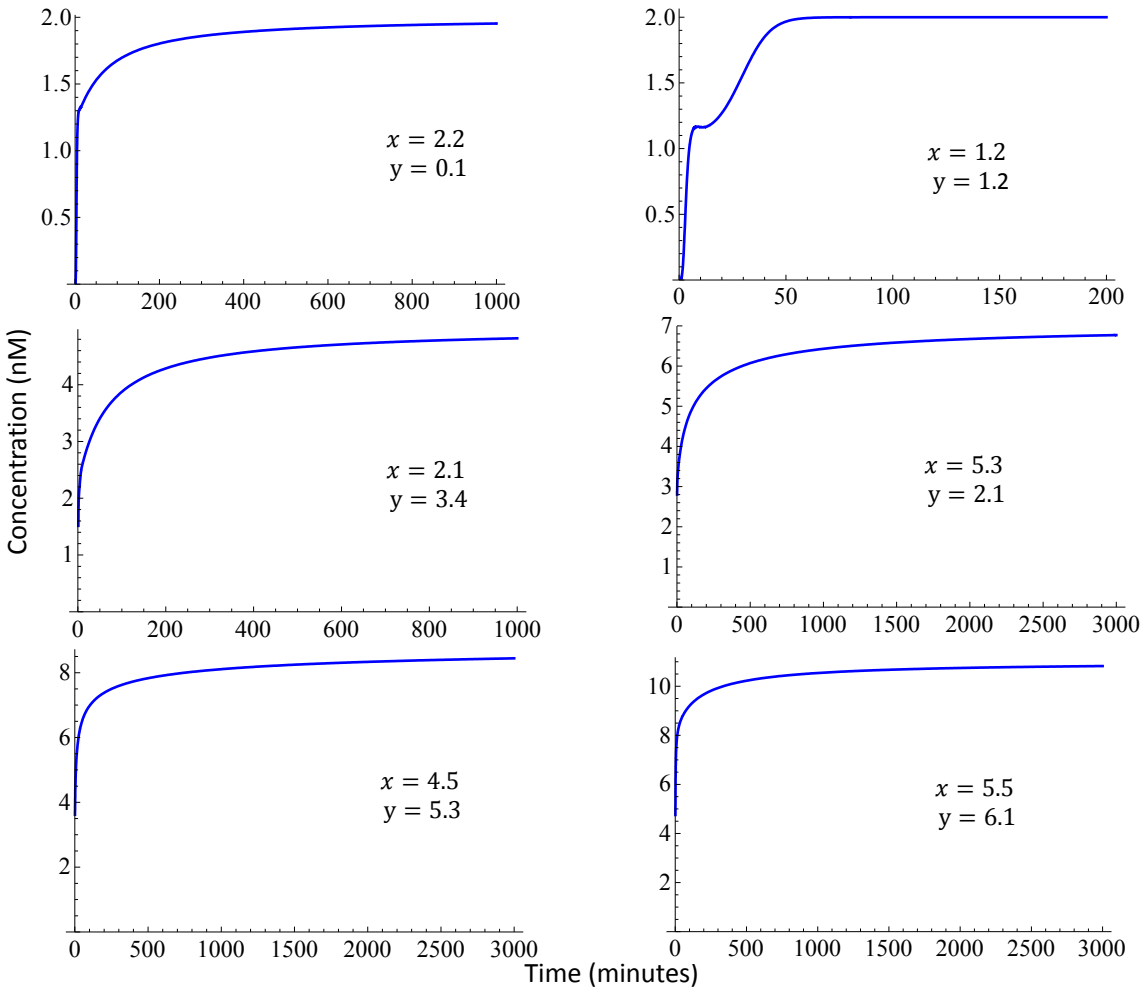


Figure 4.11: Simulation results for the system shown in Figure 6.1.

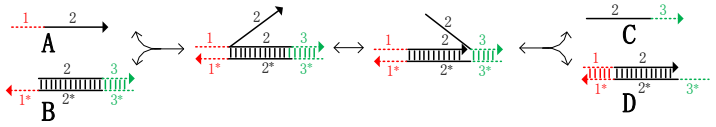


Figure 4.12: Implementation of $A + B \underset{r}{\overset{f}{\rightleftharpoons}} C + D$ using DNA strand-displacement mechanism.

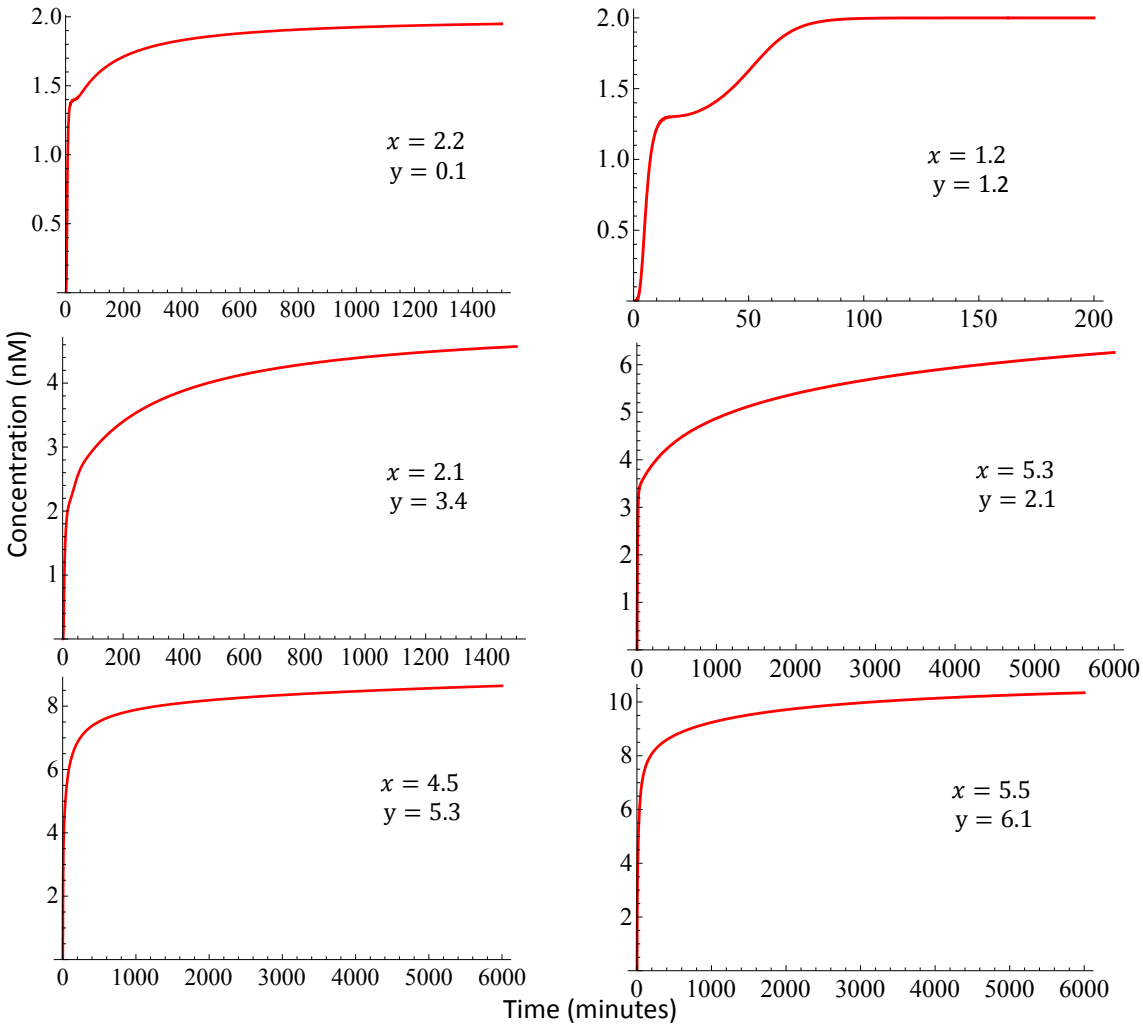


Figure 4.13: Simulation results for the DNA implementation of the system shown in Figure 6.1.

Chapter 5

Markov Chain Computations using Molecular Reactions

5.1 Introduction

The Markov chain has been frequently used for modeling and analyzing systems of chemical reactions [43],[44],[8]. However, this chapter addresses the reverse problem, i.e., modeling the Markov chain and computing its steady-state probabilities by a system of chemical reactions. Since Markov processes are commonly used in numerous processing and statistical modeling applications, a systematic method for synthesizing Markov chains with DNA strand displacement reactions leads to a systematic method for implementing these applications using DNA.

This research, for the first time, presents a systematic method of implementing first-order Markov chain processes using molecular reactions. Each state in the Markov chain is modeled by a unique *data* molecular type and each state transition is modeled by a molecular reaction and a unique *control* molecule. *Data* molecule for each state or *control* molecule for each state transition is distinguishable from molecules corresponding to other states or state transitions. All the reactions have the form of $C_{ij} + D_i \rightarrow C_{ij} + D_j$,

where C_{ij} is the *control* molecule that facilitates transition from state i to j, and D_i and D_j are *data* molecules for states i and j, respectively. The final concentration of *data* molecules related to each state determines the probability of that state. Since all of the reactions are bimolecular, the model can be mapped to a set of toehold-mediated DNA strand displacement reactions according to the second approach described in Chapter 2.

5.2 Modeling by Molecular reactions

This section describes the methodology of constructing a model for Markov chain process using molecular reactions. This model can be used to compute the steady-state probability of each state in the Markov chain diagram. The methodology has two parts: *initialization* and *transition reactions*.

Initialization: This stage consists of initializing two groups of molecules: *data* molecules and *control* molecules.

Data molecule for each state of Markov chain is a unique type of molecule assigned to that state. The initial quantity for each data molecule, except the start state, is zero. For the start state the initial value can be any large nonzero number; however, larger the initial value, more accurate the probability estimates are.

Control molecules are used to control transformation of data molecules of one state to data molecules of other states according to the transition probabilities in the Markov chain diagram. A unique type of molecule is devoted for each state transition in the chain. The quantities of control molecules are time invariant and can be determined according to the probabilities related to their corresponding transition in the chain; the ratio of quantity of a control molecule over total quantities of all control molecules in a state equals the probability of corresponding transition.

In general, the number of unique molecular types in our model is the sum of the number of states and the number of transitions in the Markov chain.

Transition Reactions: The transition reactions determine how data molecules transfer in order to implement the desired Markov chain. There is a transition reaction for each transition in the chain. This reaction transfers data molecules in the source state of transition to the data molecules in the destination state. Each transition reaction uses a control molecule for transferring data molecules. However, transition reactions should not change the concentration of control molecules. Therefore, if a control molecule is used as a reactant in a reaction, it should be also be a product of the reaction.

To illustrate our methodology we explain the molecular model for gambler problem as an instance of Markov chain[23]; a gambler starts with i dollars and plays game of chance in each step, either increasing his money by \$1 or decreasing by \$1. He stops when money is gone, *RUIN*, or when he has N dollars, *WIN*. Assuming the chances of winning, w, and loosing, l, for all states to be identical, what's the probability of ruin?

Figure 5.1 shows a 4-state (N=3) gambler problem with $w = 0.3$ and $l = 0.7$. Theoretical ruin and win probabilities for this example are 0.886076 and 0.113924, respectively [23].

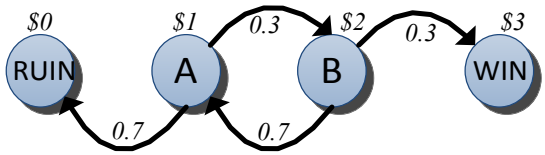


Figure 5.1: State diagram for the gambler problem with N=3.

In order to design its molecular reactions, first we devote a *data* molecular type to each state: Molecule $RUIN$ for ruin state, A and B for intermediate states, and WIN for win state. Suppose we want to compute P1, i.e., the probability of ruin if gambler starts the game at state A with \$1. Therefore, the initial value of *data* molecule A is nonzero, while the other states have *data* molecules with zero initial values. We consider 100 as the initial value of A.

Control molecules $A1$ and $A2$ are assigned to the output transitions of state A. Similarly, $B1$ and $B2$ are assigned to the transitions from state B. Because w=0.3 and l=0.7 for this example, we choose initial values as $[A1] = [B1] = 30$ and $[A2] = [B2] = 70$. One should notice that despite the exact concentrations for the control molecules, they need to conform to (5.1).

$$w = \frac{[A1]}{[A1] + [A2]} = \frac{[B1]}{[B1] + [B2]}$$

$$l = \frac{[A2]}{[A1] + [A2]} = \frac{[B2]}{[B1] + [B2]} \tag{5.1}$$

The final step is to write the molecular reactions related to each state transition. Reactions (5.2) and (5.3) represent output transitions for states A and B, respectively. These reactions with the initial concentrations for each molecular type are the proposed molecular model for the gambler problem in Figure 5.1.

$$R1: \qquad A + A1 \rightarrow B + A1$$

$$R2: \qquad A + A2 \rightarrow RUIN + A2 \tag{5.2}$$

$$R3: \qquad B + B1 \rightarrow WIN + B1$$

$$R4: \qquad B + B2 \rightarrow A + B2 \tag{5.3}$$

Thus, the gambler problem with N=3 can be modeled by eight types of molecules and four molecular reactions. Here the transition probabilities for states A and B are similar and control molecules $A1$ and $A2$ can be used for both states and $B1$ and $B2$ can be omitted.

5.3 Analysis of the Proposed Molecular Model

According to both stochastic chemical kinetics [20],[21] and mass-action kinetics [22], in this section the proposed molecular model is analyzed. We analyze the molecular model for the 4-state gambler problem shown in Figure fig:markov1.

5.3.1 Stochastic Model

If we only consider state A, there are two ways for data molecules A to transfer from this state; they can participate either in reaction $R1$, or $R2$. Based on the stochastic kinetics the probabilities of participating in reactions $R1$ and $R2$ can be computed as (5.4) and (5.5), respectively. We use lowercase letter to represent quantities for related molecular types; e.g., a_1 and a_2 stand for quantities of $A1$ and $A2$ respectively. Since the quantities of $A1$ and $A2$ are time invariant, the probabilities remain constant.

$$P(R1) = \frac{\begin{pmatrix} a_1 \\ 1 \end{pmatrix}\begin{pmatrix} a \\ 1 \end{pmatrix}}{\begin{pmatrix} a_1 \\ 1 \end{pmatrix}\begin{pmatrix} a \\ 1 \end{pmatrix}+\begin{pmatrix} a_2 \\ 1 \end{pmatrix}\begin{pmatrix} a \\ 1 \end{pmatrix}} = \frac{a_1}{a_1+a_2} \tag{5.4}$$

$$P(R2) = \frac{\begin{pmatrix} a_2 \\ 1 \end{pmatrix}\begin{pmatrix} a \\ 1 \end{pmatrix}}{\begin{pmatrix} a_1 \\ 1 \end{pmatrix}\begin{pmatrix} a \\ 1 \end{pmatrix}+\begin{pmatrix} a_2 \\ 1 \end{pmatrix}\begin{pmatrix} a \\ 1 \end{pmatrix}} = \frac{a_2}{a_1+a_2} \tag{5.5}$$

If all the states are considered, all of the four reactions can be fired and their probabilities are computed as (5.6).

$$
\begin{aligned}
P(R1) &= \frac{\binom{a_1}{1}\binom{a}{1}}{\binom{a_1}{1}\binom{a}{1}+\binom{a_2}{1}\binom{a}{1}+\binom{b_1}{1}\binom{b}{1}+\binom{b_2}{1}\binom{b}{1}} = \frac{a_1.a}{a(a_1+a_2)+b(b_1+b_1)} \\
P(R2) &= \frac{\binom{a_2}{1}\binom{a}{1}}{\binom{a_1}{1}\binom{a}{1}+\binom{a_2}{1}\binom{a}{1}+\binom{b_1}{1}\binom{b}{1}+\binom{b_2}{1}\binom{b}{1}} = \frac{a_2.a}{a(a_1+a_2)+b(b_1+b_1)} \\
P(R3) &= \frac{\binom{b_1}{1}\binom{b}{1}}{\binom{a_1}{1}\binom{a}{1}+\binom{a_2}{1}\binom{a}{1}+\binom{b_1}{1}\binom{b}{1}+\binom{b_2}{1}\binom{b}{1}} = \frac{b_1.b}{a(a_1+a_2)+b(b_1+b_1)} \\
P(R4) &= \frac{\binom{b_2}{1}\binom{b}{1}}{\binom{a_1}{1}\binom{a}{1}+\binom{a_2}{1}\binom{a}{1}+\binom{b_1}{1}\binom{b}{1}+\binom{b_2}{1}\binom{b}{1}} = \frac{b_2.b}{a(a_1+a_2)+b(b_1+b_1)}.
\end{aligned}
\tag{5.6}
$$

For the four probabilities in (5.6) we assume that at each step at least one reaction can be fired. In other words, $a\,(a_1+a_2)+b\,(b_1+b_2)\neq 0$. The quantities of molecules $RUIN$, A, B, and WIN denote the elements for the states of the system, $S=(ruin,a,b,win)$. Depending on which reaction is fired, S changes after each step.

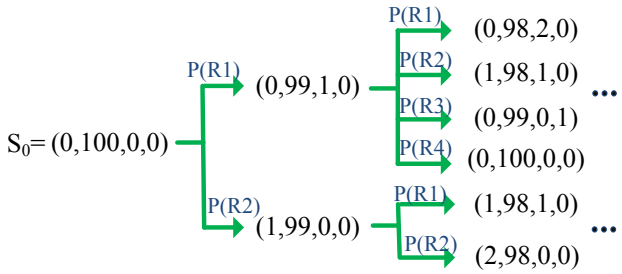


Figure 5.2: First two steps of updating the state of molecular model for Figure 5.1.

Figure 5.2 shows the graph for the first two steps of the example in Figure 5.1. One should keep in mind that the total number of *data* molecules in each state is constant.

As another interpretation for the model we consider each molecule in the system. The molecule transforms to a molecule either in left state or right state with the probabilities of 0.3 or 0.7, respectively. Therefore, we can interpret each single molecule in the system as an instance of the gambler's play.

The *Monte Carlo* simulation is used for validating the model. The goal is to compute the ruin probability if gambler arrives to play with \$1. Therefore, the simulation starts

with the initial state $S = (0, 100, 0, 0)$ and stops whenever no more reaction can be fired. The simulation is repeated 10^6 times. Figure 5.3 shows the simulation results. The horizontal axis represents the number of molecules and the blue (red) line represents the number of times the simulation ends up with those numbers of molecules in ruin (win) state. Ruin probability can be calculated as formulated in (5.7). The mean values of the ruin and win distributions in Figure 5.3 are used as the number of molecules. If we simulate with a larger initial value of *data* molecule, the probabilities can be computed more accurately. Table 5.1 shows the probabilities obtained using different initial values for *data* molecule A. Note that the accuracy improves with increase in the initial value of A.

$$P_1 = \frac{number\ of\ data\ molecules\ in\ ruin\ state}{total\ number\ of\ data\ molecules\ in\ ruin\ and\ win\ states} \tag{5.7}$$

Table 5.1: Simulation vs theoretical computation of ruin probability for example in Figure 5.1

Initial value for A	**Computed ruin probability**	**Error**
100	0.89	0.003
1000	0.887	0.0009
10000	0.8862	0.0001

5.3.2 Mass-action Kinetics

Based on the mass-action law, time variation of data molecules can be represented by the ODEs (5.8).

$$\begin{aligned}
\frac{d[A]}{dt} &= -k.\,[A_1]\,[A] - k.\,[A_2]\,[A] + k.\,[B_2]\,[B] \\
\frac{d\,[B]}{dt} &= -k.\,[B_1]\,[B] - k.\,[B_2]\,[B] + k.\,[A_1]\,[A] \\
\frac{d[S]}{dt} &= k.\,[A_2]\,[A]
\end{aligned} \tag{5.8}$$

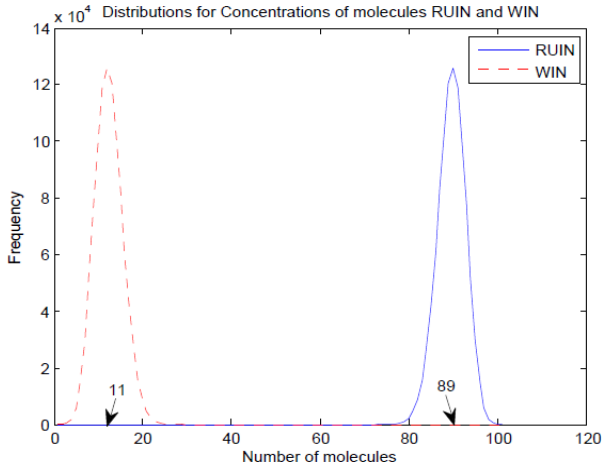


Figure 5.3: Stochastic simulation results for molecular model of Figure 5.1.

$$\frac{d[E]}{dt} = k.\,[B_1]\,[B]$$

Solving these ODEs using the initial values of molecules, we can obtain the time variation for each molecule. The final concentration of data molecule related to each state can be used to determine the probability of that state.

We used MATLAB to solve the ODEs and plot them as shown in Figure 5.4(a). The final concentration for ruin and win molecules are 88.61 (nM) and 11.39 (nM), respectively. Figure 5.4(b) illustrates the ratio $[RUIN]/([RUIN]+[WIN])$ which is the ruin probability and perfectly matches with the theoretical value.

5.4 DNA implementation

To implement the proposed model with a real molecular system we used DNA strand displacement reactions. By properly designing the toeholds in DNA molecules, an arbitrary rate of binding can be achieved. Our model consists of bimolecular reactions and it can be implemented by DNA strand displacements using both approaches presented in Section 2.3 of Chapter 2 . We choose the approach 1. For this purpose each molecule needs to be identified by two toeholds and two domains as depicted in Figure 5.5 for

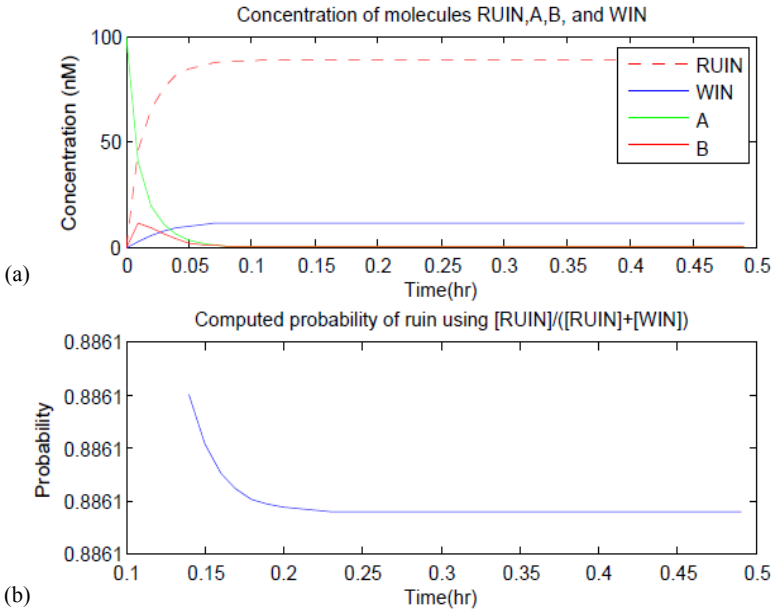


Figure 5.4: a) ODE simulation for molecular model of Markov chain in Figure 5.1, b) The computed [RUIN]/([RUIN]+[WIN]) ratio.

molecule A. In this representation continuous and dotted lines are used for domain and toehold parts, respectively.

Figure 5.5: DNA representation of molecule A.

To evaluate the DNA implementation of the proposed model, we implement the model for the example shown in Figure 5.1. All the molecules are mapped to the DNA strands as described above. We use the *Mathematica* tool of Soloveichik et al [11] to simulate the designed DNA system. The similar initial parameters as [11] are used for simulation. Figure 5.6 illustrates the dynamic concentrations of each *data* molecular type. The simulation results match with the simulation results of ODE model as shown in Figure fig:markov4(a). The ruin probability is computed as the ratio of the final concentration of $RUIN$ molecule over the summation of the final concentrations of $RUIN$ and WIN molecules.

We next use our DNA construction for a more complex instance of a gambler problem with N=9 and similar transition probabilities. We compute ruin probabilities when the gambler starts with \$5 and \$8. For the first case, we initialize the *data* molecule of the 5th state, E, to 100nM and the other *data* molecules to zero. While for the second case, we initialize the *data* molecule of the 8th state, H, to 100nM and the other *data* molecules to zero. Figure 5.7 demonstrates the simulation results. Note that as tabulated in Table 5.2, the ruin probabilities computed using the final concentrations shown in Figure 5.7 match with the theoretical probabilities.

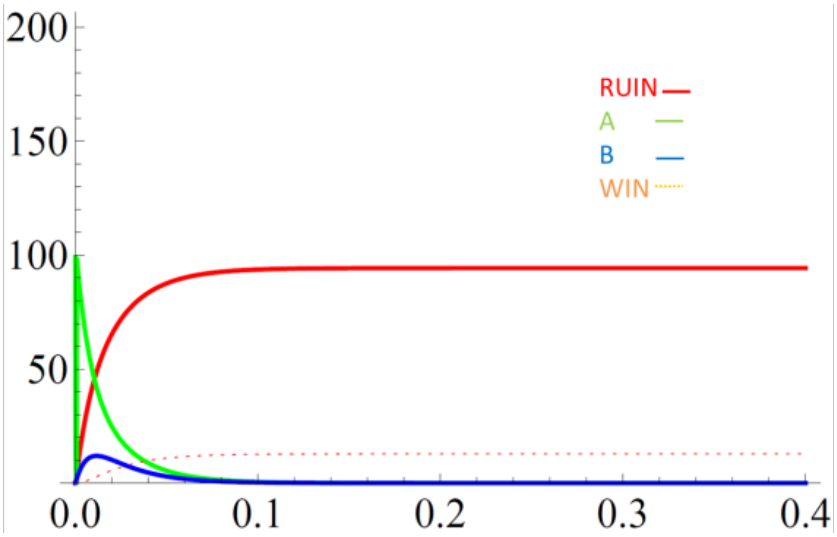


Figure 5.6: Simulation results of DNA implementation for the proposed molecular model for Figure 5.1.

Table 5.2: Simulation vs theoretical computation of ruin probabilities for A 9-state gambler Ruin Problem

Start state	**[ruin]/([ruin]+[win])**	**Theoretical probability of ruin**
$5	0.962	0.9667
$8	0.569	0.5717

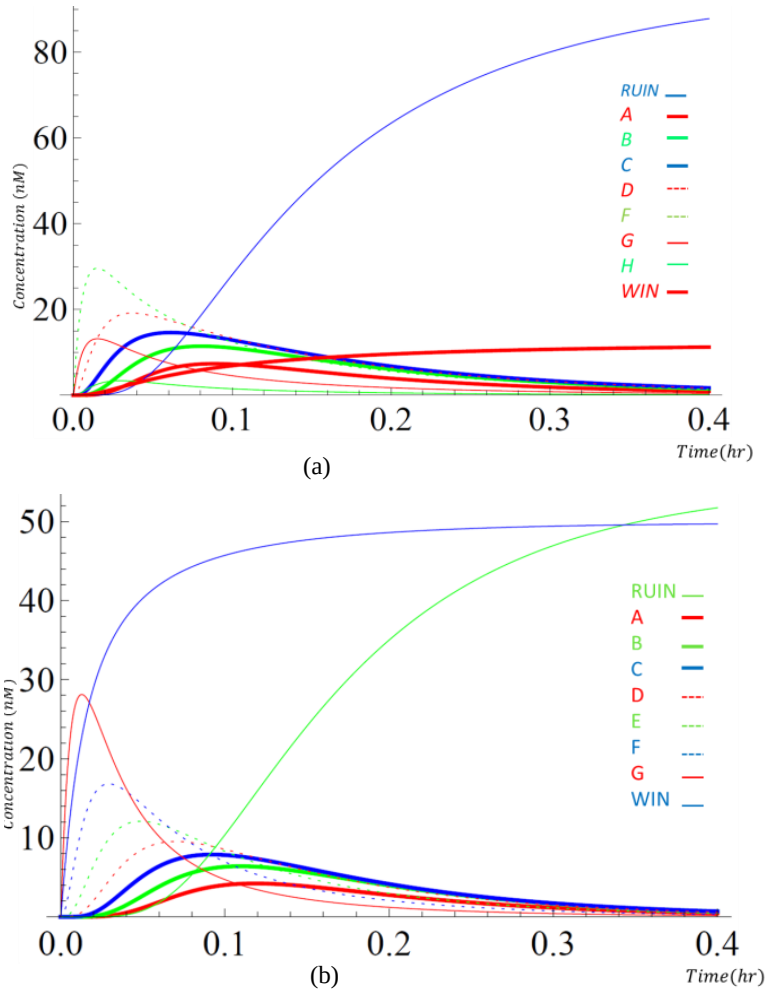


Figure 5.7: Simulation results of the DNA implementation for the gambler problem with N=9 and starting with a) \$5, b) \$8.

5.5 Discussion

Molecular systems have been used for modeling different applications. This chapter demonstrates a method for modeling the stochastic behavior of Markov chain processes using molecular reactions. Both stochastic and ODE simulation results validate our model. Although we describe the modeling of a gambler ruin problem; i.e., a first-order Markov chain with identical transition probabilities in each state, the method can be used for modeling any Markov chain process. A first-order Markov process with different transition probabilities for each state can be easily modeled by adjusting the initial quantities for control molecules of each state. Future work will be directed towards modeling of higher order Markov processes and generalizing the method for different types of random processes.

Chapter 6

CRNs for Computing Polynomials Using Fractional Coding

6.1 Fractional Coding

It has long been recognized that, viewed from a mathematical standpoint, a set of chemical reactions can exhibit rich dynamical behavior [45]. On the computational front, there has been a wealth of research into efficient methods for simulating chemical reactions, ranging from ordinary differential equations (ODEs) [46] to stochastic simulation [47]. On the mathematical front, entirely new branches of theory have been developed to characterize chemical dynamics [48]. As opposed to writing computer programs to analyze chemical systems, in the nascent field of molecular computing, the goal is computation directly with chemical reactions. In this context, a CRN transforms *input* concentrations of molecular types into *output* concentrations and so performs computation.

The question of the computational power of chemical reactions has been considered by several authors. Magnasco demonstrated that chemical reactions can compute anything that digital circuits can compute [49]. Soloveichik *et al.* demonstrated that chemical reactions are *Turing Universal*, meaning that they can compute anything that a computer algorithm can compute [43]. This work was applicable to a discrete, stochastic model of chemical kinetics. The computation is probabilistic; the total probability of error of the computation can be made arbitrarily small (but not zero).

Either explicitly or implicitly, prior work has considered two types of *encodings* for the input and output variables of CRNs [50, 51]:

1. The value of each variable corresponds to the concentration of a specific molecular type; we will call this the **direct** representation.

2. The value of each variable is represented by the difference between the concentrations of a pair of molecular types; we will call this the **dual-rail** representation [51].

In this chapter we introduce a new representation that we call the **fractional** representation. 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 following ratio:

$$x = \frac{[X_1]}{[X_0] + [X_1]}. \tag{6.1}$$

Evidently, the value is confined to the unit interval, $[0, 1]$. The proposed encoding method is inspired by prior work in designing stochastic circuits [52, 53, 54, 55]. Such circuits operate on randomized bit streams, with the values of variables represented as the fraction of 1's versus 0's in the streams. In a sense, the main contribution of this chapter is the application of this theory from stochastic circuit design to CRNs.

6.2 CRNs for Computing Polynomials

Based on the fractional representation in Eq. 6.1, we propose a CRN framework for computing univariate polynomials that map the unit interval [0,1] to itself. We

demonstrate that a CRN exists that computes any such polynomial. The full system consists of an *encoder*, the *computation CRNs* and a *decoder*, as shown in Fig. 6.1. The encoder converts the input molecular type, X (for $0 \leq [X] \leq 1$), into two molecular

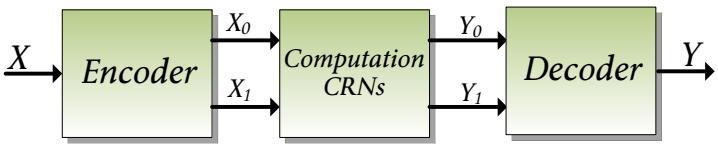


Figure 6.1: Whole system performing computation in fractional representation.

types, X_0 and X_1, such that

$$[X] = \frac{[X_1]}{[X_0] + [X_1]}.$$

The decoder converts the ratio of two molecular types, Y_0 and Y_1, into a single molecular type, Y, as the final output such that

$$[Y] = \frac{[Y_1]}{[Y_0] + [Y_1]}.$$

We describe the design of the Encoder and Decoder in Section 6.2.4, "Encoding and Decoding".

We first illustrate the Computation CRN block with a simple example. Consider the following CRN:

$$
\begin{aligned}
X_0 + X_0 &\rightarrow S_0 + X_0 + X_0 \\
X_0 + X_1 &\rightarrow 2S_1 + X_0 + X_1 \\
X_1 + X_1 &\rightarrow S_2 + X_1 + X_1
\end{aligned}
$$

(a)

$$
\begin{aligned}
S_0 + B_{0,0} &\rightarrow Y_0 + B_{0,0} \\
S_0 + B_{0,1} &\rightarrow Y_1 + B_{0,1} \\
S_1 + B_{1,0} &\rightarrow Y_0 + B_{1,0} \\
S_1 + B_{1,1} &\rightarrow Y_1 + B_{1,1} \\
S_2 + B_{2,0} &\rightarrow Y_0 + B_{2,0} \\
S_2 + B_{2,1} &\rightarrow Y_1 + B_{2,1} \\
Y_0 &\rightarrow \varnothing \\
Y_1 &\rightarrow \varnothing
\end{aligned}
$$

(b)

Set the initial concentrations as follows:

$$
\left.\begin{aligned}[B_{0,0}] &= 0.25\ nM \\ [B_{0,1}] &= 0.75\ nM\end{aligned}\right\} \quad \Rightarrow \quad b_0 = \frac{[B_{0,1}]}{[B_{0,0}] + [B_{0,1}]} = \frac{0.75}{0.25 + 0.75} = \frac{3}{4}
$$

$$
\left.\begin{aligned}[B_{1,0}] &= 0.75\ nM \\ [B_{1,1}] &= 0.25\ nM\end{aligned}\right\} \quad \Rightarrow \quad b_1 = \frac{[B_{1,1}]}{[B_{1,0}] + [B_{1,1}]} = \frac{0.25}{0.75 + 0.25} = \frac{1}{4}
$$

$$
\left.\begin{aligned}[B_{2,0}] &= 0.50\ nM \\ [B_{2,1}] &= 0.50\ nM\end{aligned}\right\} \quad \Rightarrow \quad b_2 = \frac{[B_{2,1}]}{[B_{2,0}] + [B_{2,1}]} = \frac{0.50}{0.50 + 0.50} = \frac{1}{2}
$$

Although not obvious, it may be shown that this CRN computes the function

$$
y(x) = \frac{3}{4}x^2 - x + \frac{3}{4}, \tag{6.2}
$$

where $0 \leq x \leq 1$.

Note that any unit could have been used in this chapter for the molecular concentrations. nM has been used due to the practical utility.

The CRN is composed of two sets of reactions: the three reactions in group (a) are referred as *control generating reactions* and the six reactions in group (b) represent the *transferring reactions*. The control generating reactions generate the molecules that control the transferring reactions (similar to the way that the control bits select outputs from inputs with multiplexors in electronic circuits). However, the control molecules represent analog values and transfer inputs to outputs proportionally. We note that the transferring reactions are conceptually similar to the molecular reactions proposed in Chapter 5 for implementing Markov Chains [56].

We provide details regarding the synthesis method in Section "Synthesizing CRNs for Computing Polynomials" 6.2.2. Here we simply note that, given a polynomial $y(x)$, the first step is to convert it to its *Bernstein polynomial* equivalent, $g(x)$. For the polynomial $y(x)$ in Equation (6.2),

$$g(x) = \frac{3}{4}[(1-x)^2] + \frac{1}{4}[2x(1-x)] + \frac{1}{2}x^2. \tag{6.3}$$

(A discussion of the math behind this is given in Section "Proof Based on the Mass-Action Kinetics" 6.2.3.)

Note that the coefficients of the Bernstein polynomial correspond to the values of b_i for i=0,1,2. These values are used to initialize the molecular types $B_{i,0}$ and $B_{i,1}$ for $i = 0, 1, 2$. In fact, computing with chemical reaction networks consists of two parts. First, choose a CRN as a means of building the dynamical system. Second, simulate a purposefully chosen dynamical system to equilibrium. By introducing the $B_{i,0}$ and $B_{i,1}$ species, the concentrations of which are time-invariant and fixed to what would have been rate constants, we propose changes to the first part that result in the same dynamical system simulated in the second part.

Suppose we want to evaluate $y(x)$ at x=0.5. We would initialize $X_0 = X_1$=0.5 nM such that

$$x = \frac{[X_1]}{[X_0] + [X_1]} = 0.5. \tag{6.4}$$

We would set the initial concentration of the other types to zero. The control generating reactions use X_0 and X_1 to produce the control molecules, S_0, S_1, and S_2 and transferring reactions use control molecules to compute the output. The output value, $y(x)$, is computed as the ratio of the final concentrations of Y_0 and Y_1, i.e.,

$$y(x) = \frac{[Y_1]}{[Y_0] + [Y_1]}. \tag{6.5}$$

The simulation results for evaluating this example at x=0.5 using a continuous mass-action kinetics model are shown in Fig. 6.2. As the time $t \to \infty$, the ratio

$$\frac{[Y_1(t)]}{[Y_0(t)] + [Y_1(t)]} \tag{6.6}$$

approaches the correct value of $y(0.5)$=0.4375.

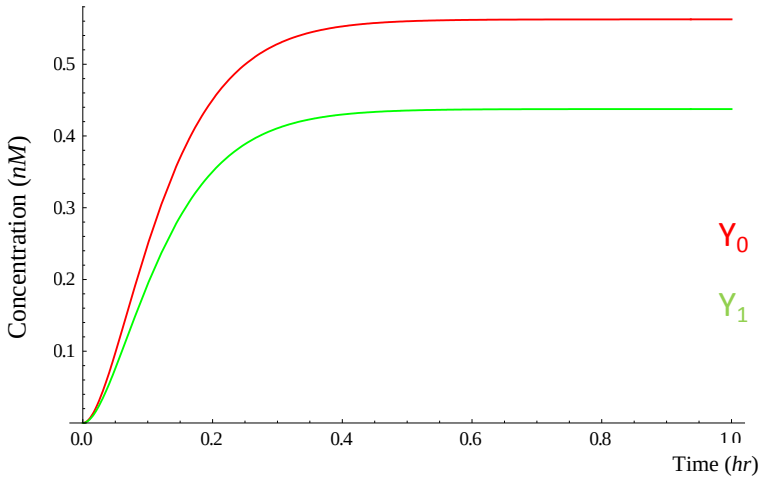


Figure 6.2: Simulation results for the CRN implementing the polynomial $y(x) = \frac{3}{4}x^2 - x + \frac{3}{4}$ at $x = 0.5$. These were obtained from an ODE simulation of the mass-action kinetics.

6.2.1 Representation by Bernstein Polynomials

In our method, the **Bernstein representation** of a polynomial is a key element. We briefly describe the relevant mathematics. The family of $n + 1$ polynomials of the

form

$$B_{i,n}(x) = \binom{n}{i} x^i (1-x)^{n-i}, \qquad i = 0, \ldots, n \tag{6.7}$$

are called Bernstein basis polynomials of degree n. A linear combination of Bernstein basis polynomials of degree n,

$$g(x) = \sum_{i=0}^{n} b_{i,n} B_{i,n}(x), \tag{6.8}$$

is a Bernstein polynomial of degree n. The $b_{i,n}$'s are called Bernstein coefficients.

Polynomials are usually represented in power form, i.e.,

$$y(x) = \sum_{i=0}^{n} a_{i,n} x^i. \tag{6.9}$$

We can convert such a power-form polynomial of degree n into a Bernstein polynomial of degree n. The conversion from the power-form coefficients, $a_{i,n}$, to the Bernstein coefficients, $b_{i,n}$, is a closed-form expression:

$$b_{i,n} = \sum_{j=0}^{i} \frac{\binom{i}{j}}{\binom{n}{j}} a_{j,n}, \qquad 0 \le i \le n. \tag{6.10}$$

For a proof of this, the reader is referred to [57].

Generally speaking, a power-form polynomial of degree n can be converted into an equivalent Bernstein polynomial of degree greater than or equal to n. The coefficients of a Bernstein polynomial of degree $m+1$ ($m \ge n$) can be derived from the Bernstein coefficients of an equivalent Bernstein polynomial of degree m as

$$b_{i,m+1} = \begin{cases} b_{0,m} & i = 0 \\ (1 - \frac{i}{m+1}) b_{i,m} + \frac{i}{m+1} b_{i-1,m} & 1 \le i \le m \\ b_{m,m} & i = m+1. \end{cases} \tag{6.11}$$

Again, for a proof the reader is referred to [57].

By encoding the values of variables as the ratio of the concentrations of two molecular types,

$$x = \frac{[X_1]}{[X_0] + [X_1]},$$

we can only represent numbers between 0 and 1. Accordingly, our method synthesizes functions that map the unit interval [0,1] onto itself. The method can also synthesize functions that map the unit interval to the negative unit interval [-1,0]. This computes the negative of a function that maps the unit interval to itself. As was shown in Example 1, the coefficients of the polynomials that we compute are also represented in this fractional form. Fortunately, it has been shown that polynomials that maps the unit interval [0,1] onto the interval (0,1) can be converted into a Bernstein polynomial *with all coefficients in the unit interval* [58]. Note, that the value of polynomial should not reach 0 or 1 in the unit interval, otherwise, it can't be converted into a Bernstein polynomial; however, it can be approximated by a Bernstein polynomial.

6.2.2 Synthesizing CRNs for Computing Polynomials

In this section we present a systematic methodology for synthesizing CRNs that can compute polynomials. As discussed in the previous section, we assume that the target polynomial is given in Bernstein form, with all coefficients in the unit interval. The method is composed of two parts, designing the CRN and initializing certain types to specific values, as discussed in the following section.

Designing the CRN

The CRN reactions consist of two sets of reactions that we call the *control generating reactions* and the *transferring reactions*.

First consider the control generating reactions. When our proposed CRN is computing a polynomial of degree m, each control generating reaction should have m reactants. The reactions consist of all possible combinations of m molecules chosen from X_0 and X_1. These $(m+1)$ reactions are listed in (6.12). In the first reaction of (6.12), all reactants are chosen from molecules of X_0 and produce molecules of S_0. In the second, $(m-1)$ molecules of X_0 and one molecule of X_1 are combined to produce molecules of

S_1. Similarly, the $(i+1)$st reaction contains i molecules of X_1 and $(m-i)$ molecule of X_0. The total number of possible reactions, as shown in (6.12), is $(m+1)$.

$$\begin{aligned}
mX_0 &\rightarrow S_0 + mX_0 \\
X_1 + (m-1)X_0 &\rightarrow mS_1 + X_1 + (m-1)X_0 \\
2X_1 + (m-2)X_0 &\rightarrow \binom{m}{2} S_2 + 2X_1 + (m-2)X_0 \\
&\vdots \\
iX_1 + (m-i)X_0 &\rightarrow \binom{m}{i} S_i + X_{0,1} + iX_1 + (m-i)X_0 \\
&\vdots \\
mX_1 &\rightarrow S_m + mX_1. \qquad (6.12)
\end{aligned}$$

A degree m Bernstein polynomial has $(m+1)$ Bernstein coefficients. We consider $(m+1)$ pairs of types $(B_{j,0}, B_{j,1})$ for $j = 0, 1, \ldots, m$, to represent these coefficients. The transferring reactions produce the final output, Y_0 or Y_1, from the products of the control generating reactions, the S_j's. They do so proportionally to the Bernstein coefficients. S_j goes to Y_0 if it combines with $B_{j,0}$ and goes to Y_1 if it combines with

$B_{j,1}$. Accordingly, there are $2(m+1)$ transferring reactions as listed in Equation (6.13).

$$\begin{aligned} S_0 + B_{0,0} &\rightarrow Y_0 + B_{0,0} \\ S_0 + B_{0,1} &\rightarrow Y_1 + B_{0,1} \\ S_1 + B_{1,0} &\rightarrow Y_0 + B_{1,0} \\ S_1 + B_{1,1} &\rightarrow Y_1 + B_{1,1} \\ &\vdots \\ S_m + B_{m,0} &\rightarrow Y_0 + B_{m,0} \\ S_m + B_{m,1} &\rightarrow Y_1 + B_{m,1} \\ Y_0 &\rightarrow \varnothing \\ Y_1 &\rightarrow \varnothing. \end{aligned} \tag{6.13}$$

The number of required reactions for the implementation of a Bernstein polynomial of degree m is equal to $3m+5$. We also need $3m+7$ molecular types listed in Table 6.1.

Table 6.1: The number of required molecular types in the proposed CRN for a polynomial of degree m.

Represented molecular type	Number of molecular types
X_0, X_1	2
S_j	$m+1$
$B_{i,0}, B_{i,1}$	$2m+2$
Y_0, Y_1	2
Total	$3m+7$

Initialization

We initialize the pair $(B_{j,0}, B_{j,1})$ according to the Bernstein coefficients $b_{j,m}$, i.e., we have

$$b_{j,m} = \frac{[B_{j,1}]}{[B_{j,0}] + [B_{j,1}]}. \tag{6.14}$$

For simplicity we initialize $B_{j,0}$ and $B_{j,1}$ such that the sum $[B_{j,0}]+[B_{j,1}]$ is the same arbitrary value for all j's. Call the sum $[B_{j,0}]+[B_{j,1}]=B$ for all j's. In fact, first we calculate the values of Bernstein coefficients using (6.10) and then initialize $B_{j,1}$ and $B_{j,0}$ as $[B_{j,1}]=B\times b_{j,m}$ and $[B_{j,0}]=B-[B_{j,1}]$. (For the example in the introduction, we considered $B=1\ nM$.)

We initialize the corresponding molecular type in the input pair (X_0, X_1) based on the value x_{in} at which the polynomial is to be evaluated, i.e.,

$$x_{in}=\frac{[X_1]}{[X_0]+[X_1]}. \tag{6.15}$$

All the other intermediate types, i.e., the S_j's as well as the output types Y_0 and Y_1, are initialized to zero.

6.2.3 Proof Based on the Mass-Action Kinetics

We use an ordinary differential model of the mass-action kinetics to prove the correctness of our proposed CRN design.

The control generating reactions (6.12) produce types S_j while the transferring reactions (6.13) consume them. Therefore the ODEs for the types S_j are:

$$\begin{aligned}
\frac{d[S_0]}{dt}&=[X_0]^m-[B_{0,0}][S_0]-[B_{0,1}][S_0]=[X_0]^m-[S_0]([B_{0,0}]+[B_{0,1}])\\
\frac{d[S_1]}{dt}&=m[X_0]^{m-1}[X_1]-[B_{1,0}][S_1]-[B_{1,1}][S_1]=m[X_0]^{m-1}[X_1]-[S_1]([B_{1,0}]+[B_{1,1}])\\
&\vdots\\
\frac{d[S_k]}{dt}&=\binom{m}{k}[X_0]^{m-k}[X_1]^k-[B_{k,0}][S_k]-[B_{k,1}][S_k]=\binom{m}{k}[X_0]^{m-k}[X_1]^k-[S_k]([B_{k,0}]+[B_{k,1}])\\
&\vdots\\
\frac{d[S_m]}{dt}&=[X_1]^m-[B_{m,0}][S_m]-[B_{m,1}][S_m]=[X_1]^m-[S_m]([B_{m,0}]+[B_{m,1}]).
\end{aligned}$$

At equilibrium $\frac{d[S_j]}{dt}$=0 for all j's. Accordingly, we can compute the S_j's as:

$$[S_j]=\frac{\binom{m}{j}[X_0]^{m-j}[X_1]^j}{[B_{j,0}]+[B_{j,1}]}\qquad 0\le j\le m. \tag{6.16}$$

Now we write the ODEs for the output types Y_0 and Y_1. Based on the transferring reactions (6.13), we have:

$$\begin{aligned}\frac{d[Y_0]}{dt} &= [B_{0,0}][S_0] + [B_{1,0}][S_1] + \cdots + [B_{m,0}][S_m] - [Y_0] \\ \frac{d[Y_1]}{dt} &= [B_{0,1}][S_0] + [B_{1,1}][S_1] + \cdots + [B_{m,1}][S_m] - [Y_1] \end{aligned} \tag{6.17}$$

At equilibrium $\frac{d[Y_0]}{dt} = \frac{d[Y_1]}{dt} = 0$ and

$$\begin{aligned}[Y_0] &= [B_{0,0}][S_0] + [B_{1,0}][S_1] + \cdots + [B_{m,0}][S_m] \\ [Y_1] &= [B_{0,1}][S_0] + [B_{1,1}][S_1] + \cdots + [B_{m,1}][S_m]. \end{aligned} \tag{6.18}$$

According to the fractional encoding, the output value, y, is calculated as follows.

$$y = \frac{[Y_1]}{[Y_0]+[Y_1]} = \tag{6.19}$$
$$\frac{[B_{0,1}][S_0]+[B_{1,1}][S_1]+\ldots+[B_{m,1}][S_m]}{([B_{0,0}][S_0]+[B_{1,0}][S_1]+\ldots+[B_{m,0}][S_m])+([B_{0,1}][S_0]+[B_{1,1}][S_1]+\cdots+[B_{m,1}][S_m])} \quad .$$

With the assumption that $([B_{j,0}] + [B_{j,1}]) = B$ for all j's, we have:

$$\begin{aligned} y &= \frac{[B_{0,1}][S_0] + [B_{1,1}][S_1] + \cdots + [B_{m,1}][S_m]}{([B_{0,0}] + [B_{0,1}])[S_0] + ([B_{1,0}] + [B_{1,1}])[S_1] + \cdots + ([B_{m,0}] + [B_{m,1}])[S_m]} \\ &= \frac{[B_{0,1}][S_0] + [B_{1,1}][S_1] + \cdots + [B_{m,1}][S_m]}{B([S_0] + [S_1] + \cdots + [S_m])} \\ &= \frac{\sum_{j=0}^{m} [B_{j,1}][S_j]}{B(\sum_{j=0}^{m} [S_j])}. \end{aligned} \tag{6.20}$$

By substituting $[S_i]$ from Eq. (6.16)

$$y = \frac{\sum_{j=0}^{m} [B_{j,1}] \frac{\binom{m}{j}[X_0]^{m-j}[X_1]^j}{B}}{B(\sum_{j=0}^{m} \frac{\binom{m}{j}[X_0]^{m-j}[X_1]^j}{B})} \tag{6.21}$$

We know that $\sum_{j=0}^{m}\binom{m}{j}[X_0]^{m-j}[X_1]^j = ([X_0]+[X_1])^m$, due to binomial theorem; therefore, the denominator can be replaced by $([X_0]+[X_1])^m$.

$$\begin{aligned} y &= \frac{\sum_{j=0}^{m}[B_{j,1}]\frac{\binom{m}{j}[X_0]^{m-j}[X_1]^j}{B}}{([X_0]+[X_1])^m} \\ &= \sum_{j=0}^{m}\frac{[B_{j,1}]}{B}\binom{m}{j}\frac{[X_0]^{m-j}[X_1]^j}{([X_0]+[X_1])^m} \\ &= \sum_{j=0}^{m} b_{j,m}\binom{m}{j}(1-x)^{m-j}x^j \end{aligned} \tag{6.22}$$

Equation (6.22) is exactly the expression for a Bernstein polynomial representation of degree m for $y(x)$. Thus, this CRN computes $y(x)$. Note that y is finite since $0 \leq [X_0] \leq 1$ and $0 \leq [X_1] \leq 1$. Therefore, for every initial state of interest our proposed CRN computes a stable equilibrium state.

Note that, in general, all the rate constants in our CRNs are assumed to be equal to each other. More precisely, based on the proof, there are three categories of reactions with respect to the rate constants: the control generating reactions, the transferring reactions, and the last two annihilation reactions of the transferring reactions. All reactions in each of these *categories* are required to have the same rate constant.

6.2.4 Encoding and Decoding

Our proposed CRNs perform computations on the fractional representation in Eq. 6.1. In this section we present chemical reactions that convert between this representation and a "direct representation", where the value of each variable is represented directly the concentration of a molecular type.

Encoding

Let a molecular type X denote the direct representation of the input value x and (X_0, X_1) denote the molecular pair for its fractional representation. Assume that the

total concentration of X_0 and X_1 is 1 nM. Then we have

$$\left.\begin{array}{l}[X] = \frac{[X_1]}{[X_0]+[X_1]} \\ [X_0] + [X_1] = 1nM\end{array}\right\} \quad \Rightarrow \quad \left\{\begin{array}{l}[X_1] = [X] \\ [X_0] = 1 - [X_1]\end{array}\right. \tag{6.23}$$

Since the concentration values for X_1 and X are the same and subsequent stages do not consume them, type X can be directly used as type X_1 in the fractional representation.

For generating X_0, we must implement subtraction. This is a little tricky. We designed the following reactions (6.24) for this task. T is initialized to 1 nM and B is an intermediate molecular type with initial value of zero.

$$\begin{array}{rcl} T & \rightarrow & X_0 + T \\ B + X_0 & \rightarrow & \varnothing \\ X_1 & \rightarrow & X_1 + B \\ X_0 & \rightarrow & \varnothing \end{array} \tag{6.24}$$

For these reactions the ODEs are

$$\begin{array}{rl} \frac{d[X_0]}{dt} = & [T] - [B][X_0] - [X_0] \\ \frac{d[B]}{dt} = & [X_1] - [B][X_0] \end{array} \tag{6.25}$$

and at equilibrium we have

$$\frac{d[X_0]}{dt} = 0 \Rightarrow [X_0] = [T] - [B][X_0] \tag{6.26}$$

$$\frac{d[B]}{dt} = 0 \Rightarrow [X_1] = [B][X_0]. \tag{6.27}$$

By substituting $[B][X_0]$ from Equation (6.27) to (6.26) we have

$$[X_0] = [T] - [X_1]. \tag{6.28}$$

Equation (6.28) is valid when $[T] \geq [X_1]$. Since $[X_0]$ cannot be negative, for $[T] \leq [X_1]$, $[X_0] = 0$. Thus, the equilibrium ODE solution for these reactions is

$$[X_0] = \begin{cases} [T] - [X_1] & \text{if} \quad [T] \geq [X_1] \\ 0 & \text{if} \quad [T] \leq [X_1]. \end{cases} \tag{6.29}$$

If T is initialized to 1 nM, Reactions (6.24) compute $[X_0] = 1 - [X_1]$.

So reactions (6.24) encode the input concentration of X as a pair of concentrations (X_0, X_1) in a fractional representation. Here, in fact, X_1 can substitute for X, as discussed above. Note that the concentration of X_0 is initialized to zero at the outset.

Decoding

For the output of our molecular computing system, we convert the fractional representation back to a direct representation. If the fractional output is represented by the pair of molecules (Y_0, Y_1) and the direct output by Y, we have

$$[Y] = \frac{[Y_1]}{[Y_0] + [Y_1]}. \tag{6.30}$$

In other words, we need to compute the summation of $[Y_0]$ and $[Y_1]$ and then the ratio of $[Y_1]$ over this summation. For this computation, we use the reactions proposed in [59]. We will show that Reactions (6.31) compute $[Y'] = [Y_0] + [Y_1]$ and Reactions (6.32) compute the final output $[Y] = \frac{[Y_1]}{[Y']} = \frac{[Y_1]}{[Y_0]+[Y_1]}$.

$$\begin{aligned} Y_0 &\rightarrow Y_0 + Y' \\ Y_1 &\rightarrow Y_1 + Y' \\ Y' &\rightarrow \varnothing \end{aligned} \tag{6.31}$$

$$\begin{aligned} Y_1 &\rightarrow Y_1 + Y \\ Y' + Y &\rightarrow Y' \end{aligned} \tag{6.32}$$

According to the ODEs of the Reactions (6.31) we have

$$\frac{d[Y']}{dt} = [Y_0] + [Y_1] - [Y']$$

and at equilibrium

$$\frac{d[Y']}{dt} = 0 \Rightarrow [Y'] = [Y_0] + [Y_1]. \tag{6.33}$$

Similarly for Reactions (6.32) we have

$$\frac{d[Y]}{dt} = [Y_1] - [Y][Y']$$

and the equilibrium value of $[Y]$ is

$$\frac{d[Y]}{dt} = 0 \Rightarrow [Y] = \frac{[Y_1]}{[Y']} = \frac{[Y_1]}{[Y_0] + [Y_1]}. \tag{6.34}$$

Therefore the set of reactions in (6.31) and (6.32) implement the decoding of the output.

6.2.5 DNA Implementation

The proposed CRN for computing polynomials is general in the sense that it can be implemented by any chemical or biochemical system with mass-action kinetics. As a practical medium, we choose DNA strand-displacement reactions. Indeed, we used the first approach, presented in Chapter 2, to map CRNs to DNA reactions.

We illustrate with the following target function:

$$y(x) = \frac{1}{4} + \frac{9}{8}x - \frac{15}{8}x^2 + \frac{5}{4}x^3 \tag{6.35}$$

The CRN includes reactions for the encoder, computation, and decoder parts. The Bernstein polynomial for $y(x)$ is

$$g(x) = \frac{2}{8}[(1-x)^3] + \frac{5}{8}[3x(1-x)^2] + \frac{3}{8}[3x^2(1-x)] + \frac{6}{8}x^3. \tag{6.36}$$

From the Bernstein coefficients, we initialize the types $(B_{i,0}, B_{i,1})$ for $i = 0, 1, 2, 3$ as follows:

$$\left.\begin{aligned}[B_{0,0}] &= 0.6\ nM \\ [B_{0,1}] &= 0.2\ nM\end{aligned}\right\} \quad \Rightarrow \quad \frac{0.2}{0.6+0.2} = \frac{2}{8}$$

$$\left.\begin{aligned}[B_{1,0}] &= 0.3\ nM \\ [B_{1,1}] &= 0.5\ nM\end{aligned}\right\} \quad \Rightarrow \quad \frac{0.5}{0.3+0.5} = \frac{5}{8}$$

$$\left.\begin{aligned}[B_{2,0}] &= 0.5\ nM \\ [B_{2,1}] &= 0.3\ nM\end{aligned}\right\} \quad \Rightarrow \quad \frac{0.3}{0.5+0.3} = \frac{3}{8}$$

$$\left.\begin{aligned}[B_{3,0}] &= 0.2\ nM \\ [B_{3,1}] &= 0.6\ nM\end{aligned}\right\} \quad \Rightarrow \quad \frac{0.6}{0.2+0.6} = \frac{6}{8}$$

We map our design to DNA strand-displacement reactions and evaluate it for 11 different input values between 0 and 1. The values of y computed by these CRN are plotted against x and shown with the target polynomial $y(x)$ in Fig. 6.3. Table 6.2 tabulates the computed values of $y(x)$ and the corresponding errors.

Table 6.2: Accuracy of a DNA strand displacement implementation of a CRN computing $y(x) = \frac{1}{4} + \frac{9}{8}x - \frac{15}{8}x^2 + \frac{5}{4}x^3$ using the proposed method.

x_{in}	Computed y(x)	Error (%)
0	0.261	4.4
0.1	0.3626	5
0.2	0.4207	2.5
0.3	0.4588	1.4
0.4	0.4838	0.8
0.5	0.5010	0.2
0.6	0.5180	0.4
0.7	0.5426	0.9
0.8	0.5823	1.3
0.9	0.6356	3
1	0.723	4

For the DNA implementation we used the parameters based on the examples in [11]. The maximum strand displacement rate constant is $q_{max} = 10^6 M^{-1} s^{-1}$, and the

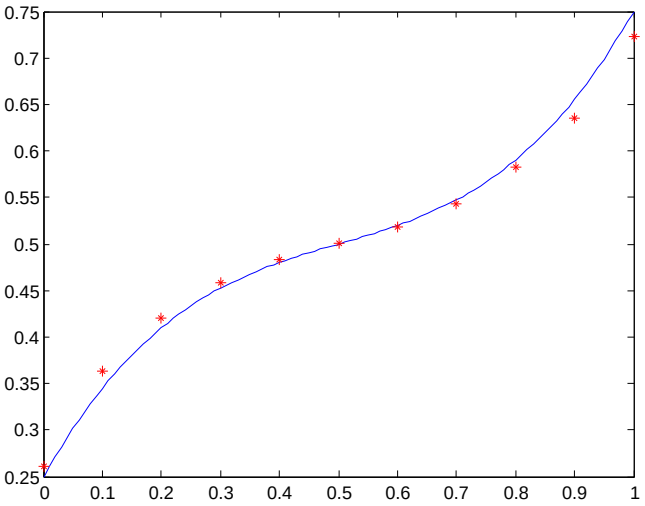


Figure 6.3: The values of $y(x)$ computed by a DNA implementation of proposed CRN. Blue line: target $y(x)$. Red stars: computed by DNA reactions.

initial concentrations of auxiliary complexes is set to $C_{max} = 10^{-5}M$. If the concentration of auxiliary species, C_{max}, is much larger than the maximum concentration of other species, (i.e., in proposed CRNs $C_{max} \gg 1nM$) then, as described in [11], we can assume that over the simulation time the auxiliary concentrations remain effectively constant. Therefore, DNA reactions correctly emulate the CRN independent of the auxiliary concentrations. Note that, for this assumption, the simulation time and reaction rates should not be very large values [11]. Although these requirements have been met in our simulations, errors exist.

As we describe later, the error stems from the fact that each molecular reaction is implemented by a sequence of DNA strand displacement reactions; the concentrations of auxiliary molecules, C_{max}, is bounded. In fact, if $C_{max} \to \infty$ the DNA simulation results converge to ODE simulation results. Further details concerning the analysis of errors when implementing CRNs with DNA strand displacement reactions, as well as a

proof of convergence of a DNA implementation to the target CRN, can be found in the Supplementary Information of [11] and [1].

Using the method presented in [11], each chemical reaction with m reactants and nonzero products can be emulated by $m+1$ DNA strand displacement reactions. For example, bimolecular reactions are mapped to 3 DNA strand displacement reactions. To illustrate this, we present a sequence of DNA strand displacement reactions that are used to simulate a bimolecular reaction with three products.

As described in [11], three DNA reactions, $R1, R2$, and $R3$, shown in Fig. 6.4 implement the molecular reaction $A + B \xrightarrow{k_i} A + B + C$.

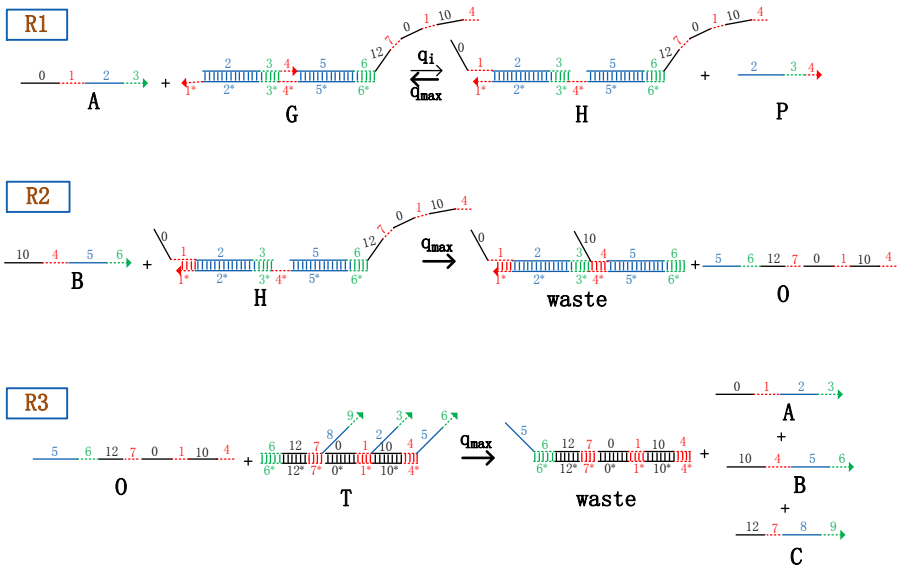


Figure 6.4: DNA strand displacement reactions that emulates reaction $A + B \xrightarrow{k_i} A + B + C$.

Unimolecular reactions without product, e.g., $Y \rightarrow \varnothing$, can be implemented by a single DNA strand displacement reaction. The DNA reaction shown in Fig. 6.5 emulates the reaction $A \xrightarrow{k_i} \varnothing$. The toehold of strand A binds to its complementary part of gate molecule G and produces double strand W_1 and single strand W_2. Since W_1 and W_2 cannot bind together, the reaction is unidirectional.

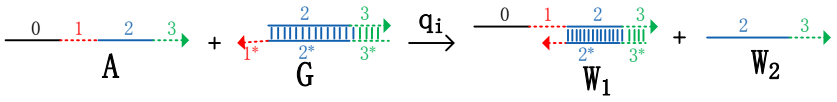


Figure 6.5: DNA strand displacement reaction that emulates reaction $A \xrightarrow{k_i} \varnothing$.

Table 6.3 summarizes the number of chemical and DNA strand displacement reactions for each group in our proposed method for computing polynomial of degree m.

Table 6.3: Number of chemical and DNA Strand-Displacement reactions for each group of the proposed CRN for computation of a Bernstein polynomial of degree m.

Group of reactions	**Type of chemical reaction**	**Number of chemical reactions**	**Number of DNA reactions**
Control generating	reactions with m reactants	$(m+1)$	$(m+1)\times(m+1)$
Transferring	bimolecular	$2m+2$	$(2m+2)\times 3$
	unimolecular without product	2	2×1
Total		$3m+5$	m^2+8m+9

6.3 Discussion

In this chapter, we have introduced a new encoding for computation with CRNs: the value corresponding to each variable consists of the ratio of the concentration of a molecular type to the sum of two types. Based on this fractional representation, we proposed a method for computing arbitrary polynomials that map the unit interval [0,1] to (-1,0). This is a rich class of functions.

Computation of polynomials with chemical kinetics has been attempted before by Buisman *et al.* [59]. Compared to our method, their method requires fewer molecular types and fewer reactions (m molecular types and $3m$ molecular reactions for a complete polynomial of degree m). However, unlike our approach, their CRNs are dependent on reaction rates. In fact, for each coefficient of the desired polynomial, they need a distinct reaction rate. This is unrealistic. Note that our approach only requires a single rate.

Soloveichik et al. [43], as well as earlier work [60, 61, 49], attempted to achieve Turing universality with chemical reactions. Although it is possible to compute polynomials with their CRNs, they did not provide a systematic framework for doing so.

The fractional representation that we propose is a nonstandard representation. However, we note that it is similar to encodings found in nature. Many biological systems have species with two distinct *states*. For example, it is common for an enzyme to have *active* and *inactive* states. The ratio of the concentrations of the two states is a meaningful value. This is quite analogous to our representation.

Clearly, the primary interest of this work is theoretical. CRNs are a fundamental model of computation, abstract yet conforming to the physical behavior of chemical systems. Delineating the range of behaviors of such systems has intellectual merit. These results may also have practical applications.

Control theory has played a remarkable role in mathematical biology, providing a framework for modeling, designing, and improving the dynamic behavior of systems such biological oscillators [62, 63, 64, 65]. Polynomials play a central role in control and oscillation. In fact, the transfer function of a control system, that is the ratio of its output to its input in the Laplace domain, is the ratio of two polynomials, i.e., $H(z) = \frac{A(z)}{B(z)} = \frac{a_0+a_1z++a_nz^n}{b_0+b_1z++b_mz^m}$ [66]. Furthermore, nonlinear feedback in oscillators can be implemented by polynomials [67].

Practitioners in synthetic biology are striving to create "embedded controllers" – viruses and bacteria that are engineered to perform useful molecular computation in situ where it is needed, for instance for drug delivery and biochemical sensing. Such embedded controllers may be called upon to perform computation such as filtering or signal processing. Computing polynomial functions is at the core of many of these computational tasks.

In the next chapter, we will attempt to generalize the method to compute a wider class of functions.

Chapter 7

CRNs for Computing Mathematical Functions using Fractional Coding

As yet, there is no *systematic* way to design molecular systems capable of computing mathematical functions. This chapter presents a systematic methodology to design CRNs for this goal. Using the fractional coding presented in Chapter 6 and expanding it for bipolar fractional coding, we propose a framework for design and implementation common mathematical functions.

7.1 Prior work

Synthetic biology in general, and molecular computing in particular, hold promises for not only monitoring proteins that have been identified as disease-specific biomarkers, but also for delivering drugs and systematically altering the interactions among molecules. Since early work on DNA computing [6], the field has evolved significantly

and various applications have been considered, some of which we point out in the following paragraphs. There has been noticeable interest in activating and inhibiting pathways by filtering proteins in different bands [18][68][69]. Furthermore, it has been demonstrated that DNA and other biological systems can be used to implement simple circuits such as AND, OR, NAND etc [33]– [70][71][72][73] [74][35] [36] [29]. These circuits have been used as building blocks for both digital signal processing [12][2][21][75][76], and mixed-signal (analog and digital) computation [75] [77]. Using these simple circuits, complex genetic circuits have been constructed to perform computation in cells [78]. To automate the design of genetic circuits, recently a computer-aided design system has been presented [78].

Additionally, as a non-conventional design language, chemical reaction networks (CRNs) have been used to design mathematical functions. Prior work has presented molecular reactions designed to compute different functions such as polynomials [59] [79], $log_a(x)$[80], and $log(1+x)$ [81]. However, no systematic method for molecular implementation of complex mathematical functions, such as exponential and sigmoid, has been presented before. This chapter presents a systematic method for designing CRNs that are able to compute a wide range of common mathematical functions. The building blocks of the proposed CRNs are simple units composed of four chemical reactions. All chemical reactions in the proposed system have two reactants. It has been shown that bimolecular chemical reactions, i.e., reactions with two reactants, can be implemented by DNA in a robust way [1]. Thus, our method provides a systematic way for DNA implementation of molecular systems that are able to compute mathematical functions.

Molecular computation of mathematical functions may have applications in the field of machine learning. Machine learning classifiers are becoming increasingly ubiquitous and their physical realization using different technologies has been considered [82][83]. Due to the remarkable advances in the field of synthetic biology, it is possible to implement biological machine learning systems in vitro and in vivo. For example cell classifier genetic logic circuits can sense features of molecules (miRNAs) in living cells, detect their

expression patterns, and selectively respond to specific cell types [72] [84] [73][85] [86]. These circuits can potentially lead to the production of personalized smart drugs that provide therapeutic medicine tailored to specific disease for specific patients[87].

Machine learning classifiers based on neural networks are commonly used today in many applications where sensor data are collected, features are computed and fed as input to a neural network [82][83]. In the biology realm, two types of neural networks have been studied in the literature: first, biological sensory neurons that convert external stimuli (light, surface electron density, etc.), coming from environment, into internal responses [88]– [89]; second, biological neural networks whose inputs and outputs are both molecular concentrations [90]-[91]. The second type is more attractive because it can work in homogeneous systems like living cells with no need of outside influence. This chapter considers the second group.

As an early theoretical research, [90] has presented chemical reactions that, based on the ordinary differential equations of mass action kinetics model, can imitate simple McClulloch-Pitts neurons. These chemical neurons can be coupled together in order to build a chemical neural network or finite state machine [92]. Practical implementation of neurons has not been considered in [90] and contemporaneous work until DNA emerged in the community as the silicon in the electronics community. Fortunately, DNA nanotechnology based on strand displacement reactions has provided a promising medium for physical implementation of neural networks and encouraged scientists to consider the realization of DNA neural networks both theoretically and experimentally. For example, in the theoretical aspect, [93] described a DNA Hopfield neural network and a DNA multi-layer perceptron. According to its proposed DNA system, [93] speculated that networks containing as many as 10^9 neurons might be feasible. In a later work, [94] described theoretical DNA implementation of a linear classifier. Beside theoretical research, experimental work for DNA neural networks has been proposed by researchers [95]-[96]. However, experimental attempts were not able to completely implement even

a single neuron till, for the first time, [58] successfully implemented artificial neural networks (ANNs) experimentally, using DNA strand displacement.

In general an ANN 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 or linear transfer function as an activation function. The DNA sigmoid function proposed in this chapter can be used to construct ANNs with nonlinear activation functions.

The contribution of this chapter is developing a framework based on a novel fractional coding approach that is able to synthesize simple bimolecular reactions to implement complex mathematical functions such as exponential, sigmoid, and tangent hyperbolic. This chapter also demonstrates a DNA implementation of a nonlinear ANN using the proposed framework, as an application.

In chapter 6 we presented a nontraditional molecular coding, referred to as *fractional representation*. In fractional representation 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,

$$x = \frac{[X_1]}{[X_0] + [X_1]} \tag{7.1}$$

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, $[0, 1]$. We refer to this representation as *unipolar fractional* coding.

Variables with values in the range $[-1, 1]$ can be represented by a different coding using two molecular types, given by:

$$x = \frac{[X_1] - [X_0]}{[X_0] + [X_1]}. \tag{7.2}$$

We refer to this representation as *bipolar fractional* coding. In this representation, the value of x lies between -1 and 1.

The novel contribution of this chapter is twofold. First, biomolecular reactions are proposed to compute operations such as ab, $1-ab$, and $sa+(1-s)b$ using unipolar and bipolar fractional coding. These molecular circuits are, respectively, referred to as Mult, NMult, and MUX. Second, this chapter 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 $\text{sigmoid}(x)$. The proposed CRNs can be implemented by biomolecular systems such as DNA.

The unipolar and biploar fractional coding approaches are inspired by digital computing using unipolar and bipolar stochastic logic circuits where numbers are represented by a bit stream of 0's and 1's [97], [98]. In molecular computing X_0 and X_1 molecules, respectively, correspond to the grouping of all 0's and all 1's. The knowledge of existing stochastic logic circuits form the basis of proposed new CRNs.

7.2 CRNs for Multiplication Units

Based on the fractional coding we propose novel CRNs for computing multiplication. These CRNs serve as fundamental units for computing desired functions described in Section 2. The fundamental multiplication units are referred to as Mult and NMult. The module Mult computes $c = a \times b$, and the module NMult computes $c = 1 - a \times b$ where a, b, and c are in unipolar fractional representation. The modules are described below.

7.2.1 Mult unit:

The Mult module shown by the symbol in Fig. 7.1(a) computes c as the multiplication of two inputs a and b all in unipolar fractional representation. In other words if $a = \frac{[A_1]}{[A_0]+[A_1]}$ and $b = \frac{[B_1]}{[B_0]+[B_1]}$ then $c = \frac{[C_1]}{[C_0]+[C_1]} = a \times b$. The set of four reactions in Fig. 7.1(a) shows the CRN for a multiplication unit, Mult.

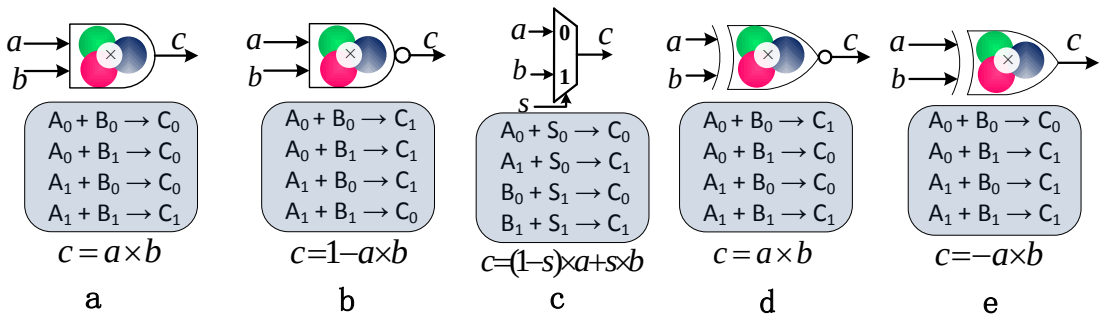


Figure 7.1: **Basic molecular modules.** **a,** Multiplication module, Mult, calculates $c = a \times b$, the multiplication of two input variables a and b in unipolar fractional representation. The module is implemented by four molecular reactions and represented by the presented symbol. **b,** The four molecular reactions and the symbol for Nmult unit. This module computes $c = 1 - a \times b$ in unipolar fractional representation. **c,** The MUX unit that performs scaled addition. a, b and c can be unipolar or bipolar, whereas s is in unipolar representation. **d,** The bipolar Mult unit that performs multiplication in bipolar fractional representation and its molecular reactions. **e,** The molecular reactions and the symbol for bipolar NMult unit. This module computes $c = -a \times b$ in bipolar fractional representation

On the basis of both stochastic and ordinary differential equations, we theoretically prove in Supplementary Section S.1 that these reactions compute $c = a \times b$.

7.2.2 NMult unit:

If we switch C_0 and C_1 in the molecular reactions of the Mult unit, we obtain the so called NMult unit which computes $1 - a \times b$. Fig. 7.1(b) shows the symbol and the set of reactions for the NMult unit.

Similar to the method we used for Multiplication module, it is easy to show that the reactions listed in 7.1(b) compute $c = 1 - a \times b$ in unipolar fractional coding. The details for the proof are in Supplementary Section S1.

The chemical reactions presented in Fig. 7.1 do not save the initial values of the input molecules of each Mult or NMult units. The reactions can be changed such that they preserve the values of either one or both of the input molecular pairs, (A_0, A_1) and (B_0, B_1). The details for these alternative Mult and NMult units are presented in Section S.2 of the Supplementary Information.

Note that for some functions we use another molecular unit, so called MUX, shown in Fig. 7.1(c). Furthermore, to perform multiplication in bipolar fractional coding, two different molecular units, shown in Fig. 7.1(d) and (e), are used. These three units are described in detail and used to compute the bipolar sigmoid function in Section 3.

7.3 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. 7.2.

7.3.1 Methodology

In the proposed methodology, the functions are approximated by truncating their Maclaurin series expansions. Note that other expansion methods such as Taylor series

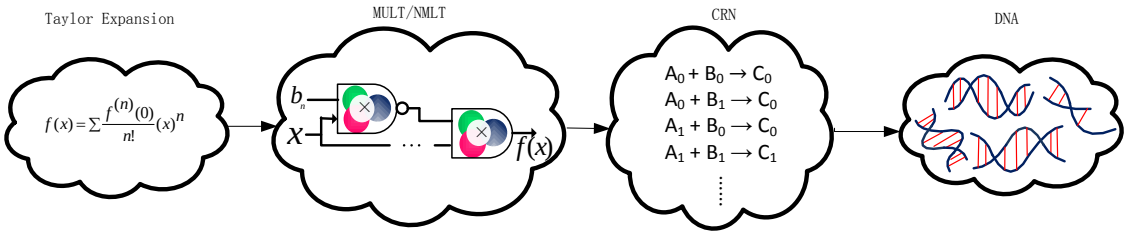


Figure 7.2: **The proposed methodology.** This figure shows the required steps for computing functions based on the proposed methodology. It starts with the approximation of the desired function as a polynomial using a series expansion method. The polynomial is then expressed in an equivalent form that only contains Mult and NMult units. The structure of Mult and NMult elements are mapped to their equivalent chemical reactions and finally the CRN is implemented by DNA strand displacement reactions.

can also be used. The approximated polynomials are then mapped into equivalent forms such that they can be implemented using Mult and NMult units. The Mult/NMult structure is then mapped to chemical reactions and then implemented by DNA. 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 the point a, corresponds to the power series

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n. \tag{7.3}$$

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:

$$e^{-x} = \sum_{n=0}^{\infty} \frac{(-x)^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - .... \tag{7.4}$$

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:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!}. \tag{7.5}$$

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 such that they can be implemented using Mult and NMult units. The Mult and NMult units are analogous to AND and NAND gates in stochastic logic; the AND and NAND gates perform the same operations for stochastic bit streams as Mult and NMult, respectively, do for molecular concentrations in unipolar fractional encoding. Recent work in stochastic logic [99] has shown that the form of such polynomials can be changed in a way that they can be mapped to a cascade of AND and NAND logic gates. The approach presented in [99] 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. This approach can be used for Maclaurin series of e^{-x}, $\sin(x)$, $\cos(x)$, $\log(1 + x)$, $\tanh(x)$, and $\text{sigmoid}(x)$. We use the approach proposed in [99] 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.

Horner's rule:

Consider a polynomial $P(x)$ of degree n given in its power form as

$$P(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + ... + a_nx^n. \tag{7.6}$$

According to the details in [99], (7.6) can be rewritten as

$$P(x) = b_0(1 - b_1x(1 - b_2x(1 - b_3x...(1 - b_{n-1}x(1 - b_nx))))...) \tag{7.7}$$

where $b_0 = a_0$ and $b_i = -\frac{a_i}{a_{i-1}}$ for $i = 1, 2, ..., n$. Provided $0 \leq b_i \leq 1$ for $i = 0, 1, ..., n$, this representation can be easily mapped to a regular cascade of molecular Mult and NMult units as described in [99].

In order to guarantee $0 \leq b_i \leq 1$ these requirements must be satisfied:

First, the coefficients of the original polynomial, i.e., the a_i's, should be alternatively positive and negative. Second, absolute values for all 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 $\text{sigmoid}(x)$, listed in equations (41) to (46) of the Supplementary Information, meet these requirements and can be represented using Equation (7.7).

For example if we apply the Horner's rule for the fifth order Maclaurin series of $f(x) = e^{-x}$, shown in (7.5), we obtain

$$e^{-x} = 1 - x(1 - \frac{x}{2}(1 - \frac{x}{3}(1 - \frac{x}{4}(1 - \frac{x}{5})))). \tag{7.8}$$

Equation (7.8) can be implemented using Mult and NMult units as shown in Table 7.1.

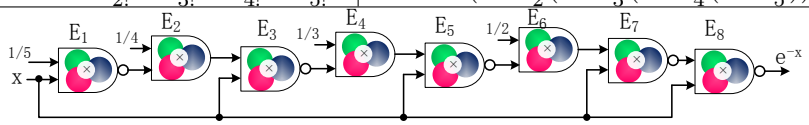
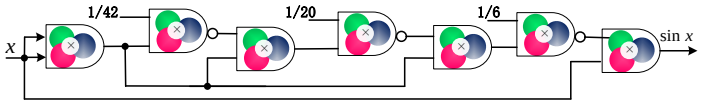
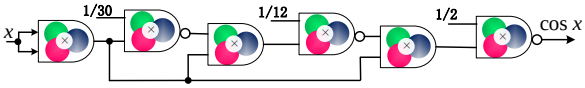
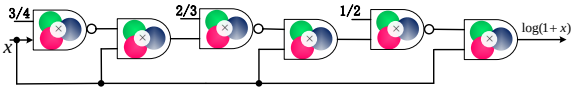
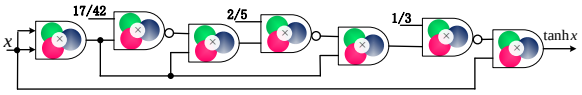
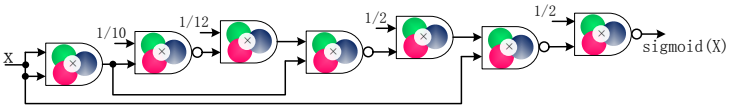
Elements, E_i, of the structure shown in Table 7.1 compute intermediate outputs, t_i in order to progressively compute e^{-x} function using the Equation (7.8). For this example we list the computation related to each element as follows:

E_1:	$t_1 = (1 - \frac{x}{5})$	E_2:	$t_2 = \frac{1}{4}t_1$	E_3:	$t_3 = (1 - \frac{x}{4}t_1)$
E_4:	$t_4 = \frac{1}{3}t_3$	E_5:	$t_5 = (1 - \frac{x}{3}t_3)$	E_6:	$t_6 = \frac{1}{2}t_5$
E_7:	$t_7 = (1 - \frac{x}{2}t_5)$	E_8:	$f(x) = 1 - xt_7 = e^{-x}$.		

Table 7.1 summarizes the truncated Maclaurin series, reformatted Maclaurin series using Horner's rule, and Mult/NMult structure for several other desired functions.

Step3- Synthesize the Chemical Reactions

Table 7.1: Truncated Maclaurin series, reformatted Maclaurin series using Horner's rule, and Mult/NMult structure for functions in equations (41)-(46) of the Supplementary Information.

Function	Truncated Maclaurin series	Reformatted using Equation (7.7)
e^{-x}	$1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!}$	$1 - x(1 - \frac{x}{2}(1 - \frac{x}{3}(1 - \frac{x}{4}(1 - \frac{x}{5}))))$
		
$\sin(x)$	$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$	$x(1 - \frac{x^2}{6}(1 - \frac{x^2}{20}(1 - \frac{x^2}{42})))$
		
$\cos(x)$	$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!}$	$1 - \frac{x^2}{2}(1 - \frac{x^2}{12}(1 - \frac{x^2}{30}))$
		
$\log(1+x)$	$x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4}$	$x(1 - \frac{x}{2}(1 - \frac{2}{3}x(1 - \frac{3}{4}x)))$
		
$\tanh(x)$	$x - \frac{1}{3}x^3 + \frac{2}{15}x^5 - \frac{17}{315}x^7$	$x(1 - \frac{x^2}{3}(1 - \frac{2}{5}x^2(1 - \frac{17}{42}x^2)))$
		
sigmoid(x)	$\frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + \frac{x^5}{480}$	$1 - \frac{1}{2}(1 - \frac{x}{2}(1 - \frac{x^2}{12}(1 - \frac{x^2}{10})))$
		

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. 7.1 (a) or (b) is used.

After designing chemical reactions the final step is to map them to DNA reactions as described in Section 7.5.

7.4 Molecular Perceptron

This section describes implementation of a single-layered neural network, also called a perceptron, by molecular reactions. As it is shown in Fig. 7.3(a), the system first computes the inner product of an input vector and a coefficient vector as $y = \sum_{i=1}^{N} w_i x_i + w_0$ and then it uses the sigmoid function to compute the final output z as $z =$sigmoid(y) for the 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 sigmoid function using bipolar fractional coding. Note that prior biomolecular implementations of artificial neural networks (ANNs) have considered either hard limit or linear activation functions [58][94]. No prior publication has considered molecular ANNs using sigmoid activation function. In this section we describe the implementation of bipolar MUX unit and bipolar Mult and NMult units.

7.4.1 MUX unit:

The MUX module shown by the symbol in Fig. 7.1(c) computes c as the weighted addition of two inputs a and b as $c = a \times (1-s) + b \times s$, where $0 \leq s \leq 1$. a, b, and c can be in unipolar or bipolar fractional representation while the weight s is always considered as

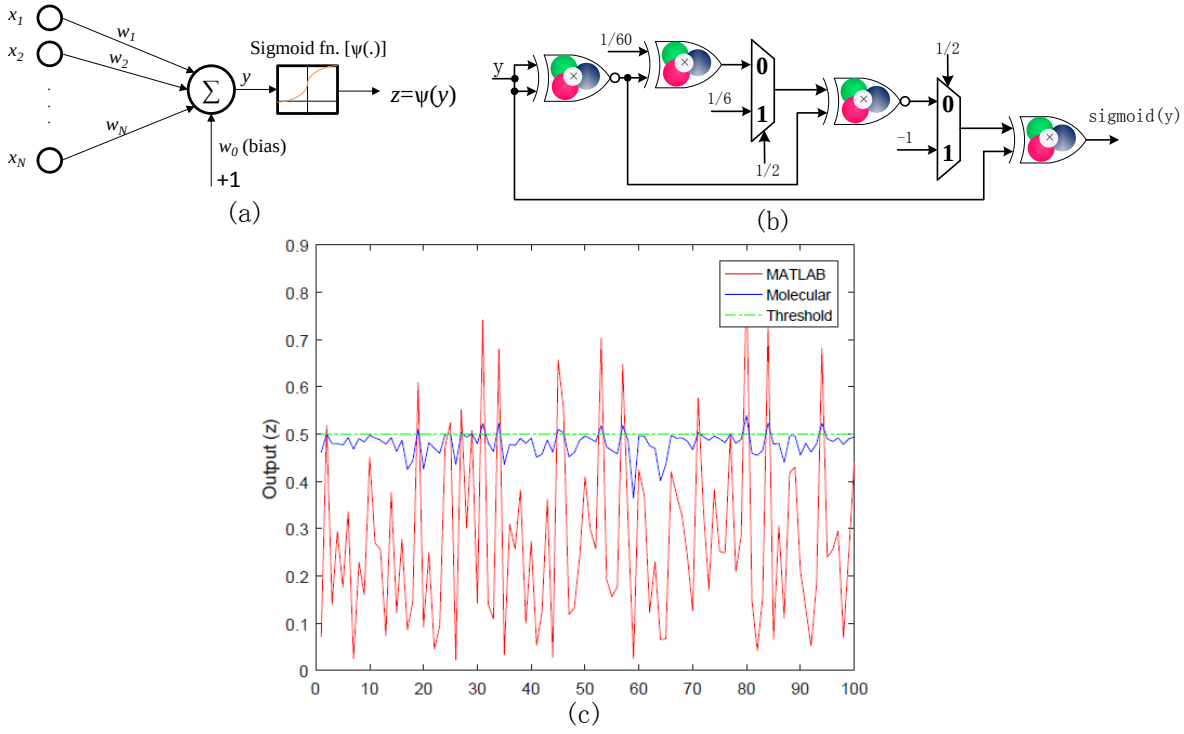


Figure 7.3: **Molecular Perceptron. a,** A perceptron system with 32 binary inputs and 1 output between 0 and 1. **b,** Molecular implementation of bipolar sigmoid function using bipolar Mult, NMult and MUX units. **c,** Results for the molecular simulation and MATLAB simulation of the perceptron system. Considering 0.5 as the threshold for decision, the results show that the molecular and MATLAB simulation agree with respect to the final decision.

unipolar. The set of four reactions in Fig. 7.1(c) shows the CRN for a MUX unit for both unipolar and bipolar fractional coding. Mass-action kinetic equations for both unipolar and bipolar fractional coding are discussed in Supplementary Information Section S.4.

7.4.2 Bipolar Mult unit:

The bipolar Mult module shown by the symbol in Fig. 7.1(d) computes c as the multiplication of two inputs a and b, where a, b and c are represented in bipolar fractional representation. In other words 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. 7.1(d) represents the CRN for a multiplication unit in bipolar fractional coding. In Supplementary Information Section S.3 we prove that these reactions compute $c = a \times b$.

7.4.3 Bipolar NMult unit:

Analogous to the way that we obtained NMult from Mult unit in unipolar fractional coding, if we switch C_0 and C_1 in the reactions of the bipolar Mult unit, we obtain the so called bipolar NMult unit which computes $-a \times b$. Fig. 7.1(e) shows the symbol and the set of reactions for the bipolar NMult unit. Similar to the method we used for Mult unit, it is easy to show that the reactions listed in 7.1(e) compute $c = -a \times b$ in bipolar fractional coding. The details for the proof are in Supplementary Information Section S.3.

7.4.4 Bipolar sigmoid function

The bipolar fractional representation can be used to implement the sigmoid function, presented in Section 7.3.1 for 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 unipolar fractional representation. In fact, for $x \in [-1, 1]$ the corresponding output

range is $[0.2689, 0.7311]$ As it is shown in [99], the sigmoid function for bipolar input and unipolar output in stochastic logic can be implemented by electronic 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. 7.1 perform for molecular systems. Accordingly, we map the circuit to the cascade of proposed molecular units as shown in Fig. 7.3(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 the desired perceptron system as it is shown in Fig. 7.3(a). We map this molecular circuit to DNA strand-displacement reactions and simulate it for $N = 32$ with the bias value of zero, i.e., $w_0 = 0$. We repeat the simulation for 100 different sets of input vectors. The results are compared to the theoretical results obtained by MATLAB simulation in Fig. 7.3(c). 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 less than that of the MATLAB output. Although the DNA computed outputs do not perfectly match with MATLAB simulation, if we consider 0.5 as the threshold for a binary decision, the DNA results and MATLAB results agree with respect to the final decision. Next section describes the details for DNA implementation of the proposed molecular systems.

7.5 DNA Implementation

In order to validate our proposed method using a biological medium, we implement the Mult/NMult circuits by DNA strand displacement (DSD) reactions. With biological origin, the DSD reactions can closely emulate mass-action kinetics of CRNs. Indeed, we use the second approach described in Chapter 2 for DNA implementation of our designs. We choose the second SNA implementation approach because recently Chen, *et al.*, [1]

showed that, using this approach, bimolecular reactions, such as $A + B \rightarrow C$, can be implemented by linear, double-stranded DNA complexes that are compatible with natural DNA. Our computational units are constructed from bimolecular reactions and can be biologically realized in a highly pure form using bacterial cloning as proposed in [1]. This means that the experimental limitations in the length of synthetic DNA strands can be bypassed. With longer strands the larger number of distinct molecular types can be designed and more complex CRNs can be realized by DNA molecules. Furthermore, as the experimental results in [1] show, for bimolecular chemical reactions, the kinetics of DNA implementation matches the mass-action kinetics model precisely. Since our designed CRNs are composed of bimolecular reactions, these can be implemented using the framework developed in [1].

Table 7.2 presents the accuracy of the proposed method, by listing the computed values of functions at eleven equally separated points in the interval [0,1]. The computed result, for each function, is reported 50 hrs after the simulation starts.The table also lists the mean square error for computation of each function at the eleven points. The error maybe due to several factors: the approximation of the function with their truncated series expansion, the implementation of the related CRNs by DSD reactions, and the limited simulation time, i.e., 50 hrs. As the results show the error is less than 1×10^{-3}.

Table 7.2: Computed values of functions with the proposed CRNs compared to their exact values.

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	Error
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	5.02e-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	4.63e-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	3.16e-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	1.8e-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	7.35e-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	2.5e-4
	exact	0.5000	0.5250	0.5498	0.5744	0.5987	0.6225	0.6457	0.6682	0.6900	0.7109	0.7311	

For each of the six target functions in this chapter we perform the DNA simulation based on the template in Fig. 2.2. For a visual comparison, each function is computed

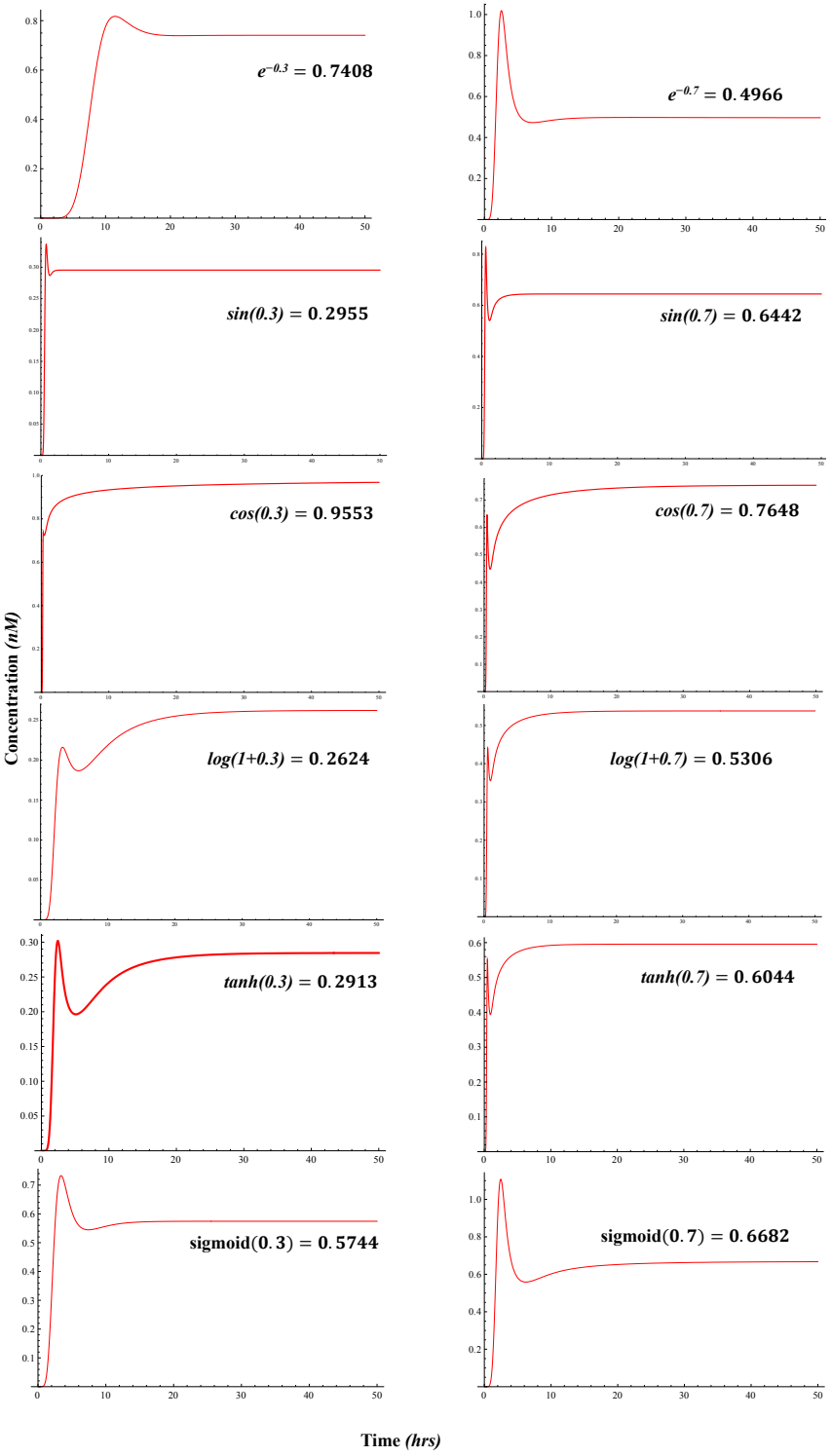


Figure 7.4: **DNA simulation results.** The DNA reaction kinetics for computation of e^{-x}, $\sin(x)$, $\cos(x)$, $\log(1+x)$, $\tanh(x)$, and $\text{sigmoid}(x)$ for x=0.3, and x=0.7. Each row is related to one function. The details for DNA implementation are listed in Supplementary Information Section S.7

for 11 different inputs 0:0.1:1 and the results are demonstrated in Fig. 7.5. The DNA computed outputs are shown by red stars and the exact values of functions are shown as blue lines. The DNA computed values follow the exact values with an acceptable accuracy.

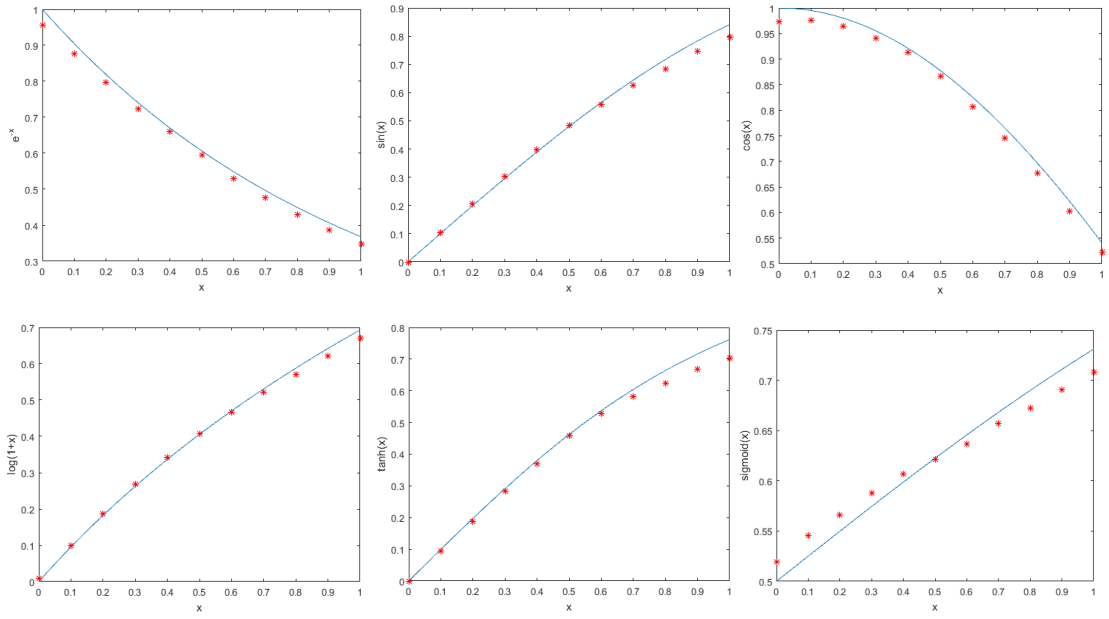


Figure 7.5: **Exact and computed values of the functions.** Computed values of functions using our proposed molecular systems along their exact graphs for e^{-x}, $\sin(x)$, $\cos(x)$, $\log(1+x)$, $\tanh(x)$, and $\text{sigmoid}(x)$. Blue lines: exact values, red stars: computed values.

7.6 Discussion

As yet, there is no *systematic* way to design molecular systems capable of computing mathematical functions. This chapter presents a systematic methodology to design CRNs for this goal. Furthermore, the proposed method is unique in that it relies exclusively on bimolecular reactions. According to recent work [1], bimolecular reactions are compatible with natural DNA. This means that, the computational elements of the

proposed CRNs can be biologically *realized* in a highly pure form by bacterial cloning, and can potentially be used for *in vivo* applications.

The Computation of polynomials has been presented in prior work [79]. Based on [79], a polynomial of degree n is converted to the equivalent Bernstein polynomial of degree m, where $n \leq m$, and is then mapped to a CRN. Although the method presented in [79] can be used to compute truncated Maclaurin series of desired functions, it uses complex molecular reactions with m reactants and at least $m+1$ products, with $m \geq 2$. The basic issue for having molecular reactions with more than two reactants is that they require large complexes. The trouble with using large complexes is that these can lead to DNA synthesis errors and are harder to be purified. The proposed systems, however, are only composed of bimolecular reactions and can be experimentally synthesized with a high level of purity.

Although the proposed molecular circuits are compatible with the experimental framework presented in [1], the proposed molecular circuits need to be experimentally demonstrated. Future work needs to be directed towards experimental validation of the theoretical framework presented in this chapter. Future work also needs to be directed towards extending the framework to complex genetic circuits where computing is carried out in cell.

Chapter 8

Conclusions and Future Directions

8.1 Conclusion

In this research, we explore the molecular implementation of several forms of computation: Signal processing, Markov chains, polynomials, and mathematical functions. In molecular systems signals are represented by time-varying *concentrations of different molecular types*, in contrast to electronic systems where signals are represented by time-varying voltage values. Although our designs are based on CRNs, a general and technology-independent programming language, we validate them by DNA as a fast growing biological technology.

We have presented a cross-disciplinary research framework that combines signal processing, analog and digital electronic circuit design, and synthetic biology to address the development of molecular computing circuits. Our research benefits from the well-established knowledge and techniques of very large scale integration (VLSI) implementation of DSP algorithms. We adjust and employ these techniques to design scalable molecular circuits with the same functionality.

Molecular systems are not substitutes for electronic computers. Indeed, the applications and challenges for these systems are different. In terms of applications, molecular systems will never be useful for fast number crunching. Rather, they are designed for *in vivo*/*in vitro* environments where, compared to electronic circuits, molecular systems are more compatible with the environment.

Table 8.1: Comparison between molecular (DNA) and electronics (silicon) computing systems.

Aspect	**DNA-based**			**Silicon-based**
	analog	Discrete-time	digital	
Addition	free	free	expensive	expensive
Multiplication	less expensive	less expensive	expensive	expensive
Fanout	expensive	expensive	free	free
Delay/Signal Transfer	very expensive			almost free
Bound on Performance	communication bounded			computation bounded
Speed	ultra-slow			very fast
Area	$\sim$ nm			$\sim$ nm
Parallelism	highly-parallel			less-parallel
Level of Integration	no integration			highly-integrated
Application Area	*in vivo*/*in vitro*			industrial/consumer

In addition to application, we point out several other fundamental differences in characteristics of molecular and electronic circuits. These are summarized in Table 8.1. Fanout operations in electronic circuits are free but are expensive in molecular implementations. Addition operations are free in molecular systems but are expensive in electronic circuits. The critical path of an electronic circuit is typically bounded by computation time; the delay elements enable reduction of critical path and faster computation. However, molecular implementations of delay elements require inherently slow transfer reactions. In fact, in contrast to electronic circuits, the most challenging part of molecular systems is delay unit. The speed of molecular systems is bounded by communication as opposed to computation. The computations in molecular systems are inherently highly parallel unlike in electronic systems where parallelism requires a significant increase in hardware resources. Finally, the electronic circuits are highly

integrated while the molecular systems are not suitable for highly integrated implementations. DNA and electronic systems also differ fundamentally with respect to storage properties. DNA systems can hold their concentrations indefinitely while the charge or stored value in an electronic system can leak and needs to be refreshed periodically.

8.2 Future Directions

For the molecular signal processing, future research direction would be a detailed study of the characteristics of continuous-time, discrete-time and digital processing molecular systems including noise analysis. For instance, the study would address how the precision correlates with changing the molecular concentrations and how robust the designs are with respect to parametric variations. In addition, the impact of specific DSP techniques, used in VLSI circuits, such as pipelining, retiming, folding, and unfolding on biomolecular designs would be investigated.

The main bottleneck in current implementations is computational speed. Unlike in electronic systems, where the speed is limited by changes in electric charge, the speed in molecular systems is limited by changes in molecular concentrations, which are inherently slow. A second future direction will be the development of faster molecular computing systems. New scheduling approaches where multiple computations are mapped into different phases of transfer will be investigated. Reducing currently achievable sample periods from hundreds of hours to a few hours, or even a few minutes, will enable experimental demonstration of some example signal processing functions using DNA. Furthermore, other biomolecular mediums such as enzymatic reactions will be considered to speed up the computational performance.

For the Markov chain computation, future work will be directed toward modeling of higher order Markov processes and generalizing the method for different types of random processes.

Although in this dissertation we used Maclaurin series expansion of mathematical functions, future research would be the investigation of other expansions such as the Lagrange expansion in order to implement other functions or to achieve more efficient implementations. What kinds of other molecular computations can be performed with fractional representation would be the direction of a future work. In this context, implementation of artificial neural networks can be investigated.

References

[1] Dalchau N. Srinivas N. Phillips A. Cardelli L. Soloveichik D. Chen, Y. and G. Seelig. Programmable chemical controllers made from dna. *Nature Nanotechnology*, 8:755–762, 2013.

[2] H. Jiang, S. Salehi, M. Riedel, and K. Parhi. Discrete-time signal processing with DNA. *ACS Synthetic Biology*, 2(5):245–254, 2013.

[3] R. H. Carlson. *Biology Is Technology: The Promise, Peril, and New Business of Engineering Life*. Cambridge, MA: Harvard UP, 2010.

[4] R. Carlson. The pace and proliferation of biological technologies. *Biosecurity and Bioterrorism: Biodefense Strategy, Practice, and Science*, 1:203–214, Sep. 2003.

[5] Michael Conrad. Molecular computing. *Advances in Computers*, 31:235–324, 1990.

[6] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(11):1021–1024, 1994.

[7] J. Chen and D. H. Wood. Computation with biomolecules. *Proceedings of the National Academy of Sciences*, 97(4):1328–1330, 2000.

[8] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[9] D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, 2006.

[10] B. Yurke, A. J. Turberfield, A. P. Mills, Jr, F. C. Simmel, and J. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406:605–608, 2000.

[11] D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

[12] H. Jiang, M. D. Riedel, and K. K. Parhi. Digital logic with molecular reactions. In *IEEE International Conference on Computer Aided Design (ICCAD)*, pages 721–727. IEEE, 2013.

[13] H. Jiang, M. Riedel, and K.K. Parhi. Synchronous sequential computation with molecular reactions. *Proc. of 2011 ACM/IEEE Design Automation Conference*, pages 836–841, 2011.

[14] H. Jiang, M.D. Riedel, and K.K. Parhi. Digital signal processing with molecular reactions. *IEEE Design and Test Magazine, (Special Issue on Bio-Design Automation in Synthetic Biology)*, 29(3):21–31, 2012.

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

[16] I. R. Epstein and J. A. Pojman. *An Introduction to Nonlinear Chemical Dynamics: Oscillations, Waves, Patterns, and Chaos*. Oxford Univ Press, 1998.

[17] P. De Kepper, I. R. Epstein, and K. Kustin. A systematically designed homogeneous oscillating reaction: the arsenite-iodate-chlorite system. *Journal of the American Chemical Society*, 103(8):2133–2134, 2008.

[18] M. Samoilov, A. Arkin, and J. Ross. Signal processing by simple chemical systems. *Journal of Physical Chemistry A*, 106(43):10205–10221, 2002.

[19] K. Oishi and E. Klavins. Biomolecular implementation of linear i/o systems. *IET Syst. Biol.*, 5:252–260, 2011.

[20] H. Jiang, M. Riedel, and K.K. Parhi. Asynchronous computations with molecular reactions. *Asilomar Conf. on Signals, Systems and Computers*, pages 493–497, 2011.

[21] S. A. Salehi, M. D. Riedel, and K. K. Parhi. Asynchronous discrete-time signal processing with molecular reactions. *Proc. of Asilomar Conference on Signals, Systems, and Computers*, pages 1767–1772, 2014.

[22] H. M. Sauro and K. Kim. Synthetic biology: It's an analog world. *Nature*, 497:572–573, 2013.

[23] R. Sarpeshkar. Analog synthetic biology. *Philos. Transact. A. Math. Phys. Eng. Sci.*, 372(20130110), 2014.

[24] R. Sarpeshkar. Analog versus digital: extrapolating from electronics to neurobiology. *Neural Comput.*, 10:1601–1638, 1998.

[25] R. Daniel, J. R. Rubens, R. Sarpeshkar, and T. K. Lu. Synthetic analog computation in living cells. *Nature*, 497:619–623, 2013.

[26] S. Hayat and T. Hinze. Toward integration of in vivo molecular computing devices: successes and challenges. *HFSP Journal*, 5(2):239–243, 2008.

[27] T. Kailath. *Linear Systems*. Prentice Hall, Englewood Cliffs, N.J., 1980.

[28] V. V. Kulkarni, H. Jiang, T. Chanyaswad, and M. Riedel. A biomolecular implementation of systems described by linear and nonlinear ODE's. In *International Workshop on Biodesign Automation*, 2013.

[29] A. Tamsir, J. J. Tabor, and C. A. Voigt. Robust multicellular computing using genetically encoded nor gates and chemical 'wires'. *Nature*, 469:212–215, 2011.

[30] L. Qian and E. Winfree. Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332:1196–1201, 2011.

[31] P. Siuti, J. Yazbek, and T.K. Lu. Synthetic circuits integrating logic and memory in living cells. *Nature Biotechnology*, 31:448–452, 2013.

[32] R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, and I. Netravali. Genetic circuit building blocks for cellular computation, communications, and signal processing. *Natural Computing*, 2:47–84, 2003.

[33] T. S. Gardner, C. R. Cantor, and J. J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(2000):339–342, 2000.

[34] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429(6990):423–429, 2004.

[35] D. Endy. Foundations for engineering biology. *Nature*, 438:449–453, 2005.

[36] K. Ramalingam, J. R. Tomshine, J. A. Maynard, and Y. N. Kaznessis. Forward engineering of synthetic bio-logical AND gates. *Biochemical Engineering Journal*, 47(1–3):38–47, 2009.

[37] T.S. Moon, C. Lou, A. Tamsi, B.C. Stanton, and C. A. Voigt. Genetic programs constructed from layered logic gates in single cells. *Nature*, 491:249–253, 2012.

[38] 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*, 355(4):619–627, 2006.

[39] D. Ro, E. Paradise, M. Ouellet, K. Fisher, K. Newman, J. Ndungu, K. Ho, R. Eachus, T. Ham, M. Chang, S. Withers, Y. Shiba, R. Sarpong, , and J. Keasling. Production of the antimalarial drug precursor artemisinic acid in engineered yeast. *Nature*, 440:940–943, 2006.

[40] 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*, 2010 (in press).

[41] D. M. Widmaier, D. Tullman-Ercek, E. A. Mirsky, R. Hill, S. Govindarajan, J. Minshull, and C. A. Voigt. Engineering the Salmonella type III secretion system to export spider silk monomers. *Molecular Systems Biology*, 5(309):1–9, 2009.

[42] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* 3rd ed. Prentice Hall Press Upper Saddle River, NJ, USA, 2009.

[43] D. Soloveichik, M. Cook, E. Winfree, and J. Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4), 2008.

[44] Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.

[45] F. Horn and R. Jackson. General mass action kinetics. *Archive for Rational Mechanics and Analysis*, 47:81–116, 1972.

[46] P. Érdi and J. Tóth. *Mathematical Models of Chemical Reactions: Theory and Applications of Deterministic and Stochastic Models.* Manchester University Press, 1989.

[47] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[48] S. Strogatz. *Nonlinear Dynamics and Chaos with Applications to Physics, Biology, Chemistry, and Engineering.* Perseus Books, 1994.

[49] M. O. Magnasco. Chemical kinetics is turing universal. *Phys. Rev. Lett.*, 78(6):1190–1193, Feb 1997.

[50] Doty D. Chen, H. and D. Soloveichik. Deterministic function computation with chemical reaction networks. *DNA Computing and Molecular Programming, LNCS, Springer*, 7433:24–42, 2012.

[51] Doty D. Chen, H. and D. Soloveichik. Rate-independent computation in continuous chemical reaction networks. *Conference on Innovations in Theoretical Computer Science*, pages 313–326, 2014.

[52] B. R. Gaines. Stochastic computing. *In proceedings of AFIP spring join computer conference, ACM*, pages 149–156, 1967.

[53] W. Qian and M. D. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *Design Automation Conference*, pages 648–653, 2008.

[54] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60(1):93–105, 2011.

[55] W. Qian, M. D. Riedel, and I. Rosenberg. Uniform approximation and Bernstein polynomials with coefficients in the unit interval. *European Journal of Combinatorics*, 32(3):448–463, 2011.

[56] S. A. Salehi, M. D. Riedel, and K. K. Parhi. Markov chain computations using moleular reactions. *IEEE International Conference on Digital Signal Processing (DSP)*, pages 689–693, 2015.

[57] R. Farouki and V. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.

[58] L. Qian and E. Winfree. Neural network computation with dna strand displacement cascades. *Nature*, 475:368372, 2011.

[59] ten Eikelder H. M. M. Hilbers P. A. J. Buisman, H. J. and A. M. L. Liekens. Computing algebraic functions with biochemical reaction networks. *Artif. Life*, 15(1):5–19, 2009.

[60] A. M. L. Liekens and C. T. Fernando. Turing complete catalytic particle computers. In *Lecture Notes in Computer Science – Advances in Artificial Life*, volume 4648, pages 1202–1211. Springer, 2007.

[61] Aspnes J. Angluin, D. and D. Eisenstat. Fast computation by population protocols with a leader. *Technical Report YALEU/DCS/TR-1358, Yale University Department of Computer Science*, 2006.

[62] Workshop:Biological Systems and Networks. Ima thematic year on control theory and its applications. http://www.ima.umn.edu/2015-2016/W11.16-20.15/abstracts.html, 2015.

[63] P. A. Iglesias and P. Ingalls, B. Control theory and systems biology. *MIT Press*, 2010.

[64] Buzi G. Doyle J. C. Chandra, F. A. Glycolytic oscillations and limits on robust efficiency. *Science*, 333(6039):187–192, 2013.

[65] Franco E. Agrawal, D. K. and R. Schulman. A self-regulating biomolecular comparator for processing oscillatory signals. *Journal of The Royal Society Interface*, 12(111), 2015.

[66] R. C. Dorf and R. H. Bishop. *Modern Control Systems 9th ed.* Prentice Hall, 2001.

[67] G. Stan and R. Sepulchre. Analysis of interconnected oscillators by dissipativity theory. *IEEE Trans. Autom. Control*, 52(2):256270, 2007.

[68] S. C. Moenke G. Prince V. L. Meena A. Thomas A. P. Skupin A. Taylor C. W. Thurley, K. Tovey and M. Falcke. Reliable encoding of stimulus intensities within

random sequences of intracellular ca^{2+} spikes. *Science Signaling*, 7((331): ra59 DOI: 10.1126/scisignal.2005237), 2014.

[69] R. R. Takayama S. Sumit, M. Neubig and J. J. Linderman. Band-pass processing in a gpcr signaling pathway selects for nfat transcription factor activation. *Integr. Biol.*, 7:1378–1386, 20.

[70] R. Weiss, G. E. Homsy, and T. F. Knight. Toward in vivo digital circuits. In *DIMACS Workshop on Evolution as Computation*, pages 1–18, 1999.

[71] K. Rinaudo, L. Bleris, R. Maddamsetti, S. Subramanian, R. Weiss, and Y. Benenson. A universal rnai-based logic evaluator that operates in mammalian cells. *Nature biotechnology*, 25(7):795–801, 2007.

[72] Z. Xie, L. Wroblewska, L. Prochazka, R. Weiss, and Y. Benenson. Multi-input rnai-based logic circuit for identification of specific cancer cells. *Science*, 333(6047):1307–1311, 2011.

[73] Jiang Y. Chen H. Liao W. Li Z. Weiss R. Li, Y. and Z. Modular Xie. construction of mammalian gene circuits using tale transcriptional repressors. *Nat. Chem. Biol.*, 11(207-2013), 2015.

[74] Gil B. Ben-Dor U. Adar R. Beneson, Y. and E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429:423–429, 1987.

[75] Jiang H. Riedel-M.D. Salehi, S.A. and K.K. Parhi. Molecular sensing and computing systems (invited paper). *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications*, 1(3):249–264, 2015.

[76] P. Senum and M. D. Riedel. Rate-independent constructs for chemical computation. *PLoS ONE*, 6(6), 2011.

[77] Selvaggio G. Rubens, J.R. and T.K. Lu. Synthetic mixed-signal computation in living cells. *Nat. Commun.*, 7:11658, 2016.

[78] Shin J. Vaidyanathan P. Paralanov V. Strychalski E.A. Ross D. Densmore D. Nielsen A.A.K., Der B.S. and Voigt C.A. Genetic circuit design automation. *Science*, (DOI: 10.1126/science.aac7341), 2016.

[79] Parhi K. K. Salehi, S. A. and M. D. Riedel. Chemical reaction networks for computing polynomials. *ACS Synthetic Biology*, 6(1):76–83, 2017.

[80] Sawlekar R. Foo, M. and 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.

[81] C.T. Chou. Chemical reaction networks for computing logarithm. *submitted to Synthetic Biology.*

[82] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2013.

[83] E. Alpaydin. *Introduction to Machine Learning*, volume 3rd Edition. MIT press, 2014.

[84] Y. Benenson. Biomolecular computing systems: principles, progress and potential. *Nature Reviews Genetics*, 13:455–468, 2012.

[85] Endo K. Takahashi S. Funakoshi S. Takei I. Katayama S. Toyoda T. Kotaka M. Takaki T. Umeda M. et al. Miki, K. Efficient detection and purification of cell populations using synthetic microrna switches. *Cell Stem Cell*, 16:699–711, 2015.

[86] Weinberg B.H. Cha S.S. Goodloe M. Wong W.W. Sayeg, M.K. and X. Han. Rationally designed microrna-based genetic classifiers target specific neurons in the brain. *ACS Synth. Biol.*, 4:788–799, 2015.

[87] Beerenwinkel N. Mohammadi, P. and Y. Benenson. Automated design of synthetic cell classifier circuits using a two-step optimization strategy. *Cell Systems*, 4(2):207–21, 2017.

[88] Sahu S. Bandyopadhyay, A. and D. Fujita. Smallest artificial molecular neural-net for collective and emergent information processing. *Applied physics letters*, 95(11):113702, 2009.

[89] Chen L. X. Lei J. L. Luo H. Q. Huang, W. T. and N. B. Li. Molecular neuron: From sensing to logic computation, information encoding, and encryption. *Sensors and Actuators: B. Chemical*, 239:704–710, 2017.

[90] Weinberger E. D. Hjelmfelt, A. and J. Ross. Chemical implementation of neural networks and turing machines. *Proc. Natl Acad. Sci. USA*, 88:10983–10987, 1991.

[91] Jr Turberfield M. Turberfield A. J. Yurke B. Mills, A. P. and P. M. Platzman. Experimental aspects of dna neural network computation. *Soft Comput.*, 5:1018, 2001.

[92] Weinberger E. D. Hjelmfelt, A. and J. Ross. Chemical implementation of finite-state machines. *Proc. Natl. Acad. Sci. U.S.A.*, 89:383, 1992.

[93] Yurke B. Mills, A. P. and P. M. Platzman. Article for analog vector algebra computation. *Biosystems*, 52:175–180, 1999.

[94] W. Zhang, M. Ha, D. Braga, M. Renn, C.D. Frisbie, and C.H. Kim. A 1v printed organic dram cell based on ion-gel gated transistors with a sub-10nw-per-cell refresh power. In *International Solid-State Circuits Conference Digest*, pages 326–328, 2011.

[95] Pemberton M. Hjelmfelt A. Laplante, J. P. and J. Ross. Experiments on pattern recognition by chemical kinetics. *J. Phys. Chem.*, 99:1006310065, 1995.

[96] H. W. et al. Lim. In vitro molecular pattern classification via dna-based weighted-sum operation. *Biosystems*, 100:1–7, 2010.

[97] B. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, volume 2, chapter 2, pages 37–172. Plenum Press, 1969.

[98] A. Alaghi and Hayes j. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12:92, 2013.

[99] K. K. Parhi and Y. Liu. Computing arithmetic functions using stochastic logic by series expansion. *IEEE Transactions on Emerging Technologies in Computing (TETC)*, (DOI: 10.1109/TETC.2016.2618750), 2016.

Appendix A

List of molecular Reactions

In this chapter, we list chemical reaction networks and DNA-level reactions for the molecular circuits presented in this thesis. Each chemical reaction discussed in this thesis is mapped to DNA level using the method described in [11].

A.1 Molecular Reactions

A.1.1 molecular perceptron

$$
\begin{aligned}
X10 + W10 &\xrightarrow{k} X10 + W10 + C1 \\
X10 + W11 &\xrightarrow{k} X10 + W11 + C0 \\
X11 + W10 &\xrightarrow{k} X11 + W10 + C0 \\
X11 + W11 &\xrightarrow{k} X11 + W11 + C1 \\
\\
X20 + W20 &\xrightarrow{k} X20 + W20 + C1 \\
X20 + W21 &\xrightarrow{k} X20 + W21 + C0 \\
X21 + W20 &\xrightarrow{k} X21 + W20 + C0 \\
X21 + W21 &\xrightarrow{k} X21 + W21 + C1
\end{aligned}
$$

$$
\begin{aligned}
X30 + W30 &\xrightarrow{\mathrm{k}} X30 + W30 + C1 \\
X30 + W31 &\xrightarrow{\mathrm{k}} X30 + W31 + C0 \\
X31 + W30 &\xrightarrow{\mathrm{k}} X31 + W30 + C0 \\
X31 + W31 &\xrightarrow{\mathrm{k}} X31 + W31 + C1
\end{aligned}
$$

$$
\begin{aligned}
X40 + W40 &\xrightarrow{\mathrm{k}} X40 + W40 + C1 \\
X40 + W41 &\xrightarrow{\mathrm{k}} X40 + W41 + C0 \\
X41 + W40 &\xrightarrow{\mathrm{k}} X41 + W40 + C0 \\
X41 + W41 &\xrightarrow{\mathrm{k}} X41 + W41 + C1
\end{aligned}
$$

$$
\begin{aligned}
X50 + W50 &\xrightarrow{\mathrm{k}} X50 + W50 + C1 \\
X50 + W51 &\xrightarrow{\mathrm{k}} X50 + W51 + C0 \\
X51 + W50 &\xrightarrow{\mathrm{k}} X51 + W50 + C0 \\
X51 + W51 &\xrightarrow{\mathrm{k}} X51 + W51 + C1
\end{aligned}
$$

$$
\begin{aligned}
X60 + W60 &\xrightarrow{\mathrm{k}} X60 + W60 + C1 \\
X60 + W61 &\xrightarrow{\mathrm{k}} X60 + W61 + C0 \\
X61 + W60 &\xrightarrow{\mathrm{k}} X61 + W60 + C0 \\
X61 + W61 &\xrightarrow{\mathrm{k}} X61 + W61 + C1
\end{aligned}
$$

$$
\begin{aligned}
X70 + W70 &\xrightarrow{\mathrm{k}} X70 + W70 + C1 \\
X70 + W71 &\xrightarrow{\mathrm{k}} X70 + W71 + C0 \\
X71 + W70 &\xrightarrow{\mathrm{k}} X71 + W70 + C0 \\
X71 + W71 &\xrightarrow{\mathrm{k}} X71 + W71 + C1
\end{aligned}
$$

$$
\begin{aligned}
X80 + W80 &\xrightarrow{\mathrm{k}} X80 + W80 + C1 \\
X80 + W81 &\xrightarrow{\mathrm{k}} X80 + W81 + C0 \\
X81 + W80 &\xrightarrow{\mathrm{k}} X81 + W80 + C0 \\
X81 + W81 &\xrightarrow{\mathrm{k}} X81 + W81 + C1
\end{aligned}
$$

$$
\begin{aligned}
X90 + W90 &\xrightarrow{k} X90 + W90 + C1 \\
X90 + W91 &\xrightarrow{k} X90 + W91 + C0 \\
X91 + W90 &\xrightarrow{k} X91 + W90 + C0 \\
X91 + W91 &\xrightarrow{k} X91 + W91 + C1
\end{aligned}
$$

$$
\begin{aligned}
X100 + W100 &\xrightarrow{k} X100 + W100 + C1 \\
X100 + W101 &\xrightarrow{k} X100 + W101 + C0 \\
X101 + W100 &\xrightarrow{k} X101 + W100 + C0 \\
X101 + W101 &\xrightarrow{k} X101 + W101 + C1
\end{aligned}
$$

$$
\begin{aligned}
X110 + W110 &\xrightarrow{k} X110 + W110 + C1 \\
X110 + W111 &\xrightarrow{k} X110 + W111 + C0 \\
X111 + W110 &\xrightarrow{k} X111 + W110 + C0 \\
X111 + W111 &\xrightarrow{k} X111 + W111 + C1
\end{aligned}
$$

$$
\begin{aligned}
X120 + W120 &\xrightarrow{k} X120 + W120 + C1 \\
X120 + W121 &\xrightarrow{k} X120 + W121 + C0 \\
X121 + W120 &\xrightarrow{k} X121 + W120 + C0 \\
X121 + W121 &\xrightarrow{k} X121 + W121 + C1
\end{aligned}
$$

$$
\begin{aligned}
X130 + W130 &\xrightarrow{k} X130 + W130 + C1 \\
X130 + W131 &\xrightarrow{k} X130 + W131 + C0 \\
X131 + W130 &\xrightarrow{k} X131 + W130 + C0 \\
X131 + W131 &\xrightarrow{k} X131 + W131 + C1
\end{aligned}
$$

$$
\begin{aligned}
X140 + W140 &\xrightarrow{k} X140 + W140 + C1 \\
X140 + W141 &\xrightarrow{k} X140 + W141 + C0 \\
X141 + W140 &\xrightarrow{k} X141 + W140 + C0 \\
X141 + W141 &\xrightarrow{k} X141 + W141 + C1
\end{aligned}
$$

$$
\begin{aligned}
X150 + W150 &\xrightarrow{\mathrm{k}} X150 + W150 + C1 \\
X150 + W151 &\xrightarrow{\mathrm{k}} X150 + W151 + C0 \\
X151 + W150 &\xrightarrow{\mathrm{k}} X151 + W150 + C0 \\
X151 + W151 &\xrightarrow{\mathrm{k}} X151 + W151 + C1
\end{aligned}
$$

$$
\begin{aligned}
X160 + W160 &\xrightarrow{\mathrm{k}} X160 + W160 + C1 \\
X160 + W161 &\xrightarrow{\mathrm{k}} X160 + W161 + C0 \\
X161 + W160 &\xrightarrow{\mathrm{k}} X161 + W160 + C0 \\
X161 + W161 &\xrightarrow{\mathrm{k}} X161 + W161 + C1
\end{aligned}
$$

$$
\begin{aligned}
X170 + W170 &\xrightarrow{\mathrm{k}} X170 + W170 + C1 \\
X170 + W171 &\xrightarrow{\mathrm{k}} X170 + W171 + C0 \\
X171 + W170 &\xrightarrow{\mathrm{k}} X171 + W170 + C0 \\
X171 + W171 &\xrightarrow{\mathrm{k}} X171 + W171 + C1
\end{aligned}
$$

$$
\begin{aligned}
X180 + W180 &\xrightarrow{\mathrm{k}} X180 + W180 + C1 \\
X180 + W181 &\xrightarrow{\mathrm{k}} X180 + W181 + C0 \\
X181 + W180 &\xrightarrow{\mathrm{k}} X181 + W180 + C0 \\
X181 + W181 &\xrightarrow{\mathrm{k}} X181 + W181 + C1
\end{aligned}
$$

$$
\begin{aligned}
X190 + W190 &\xrightarrow{\mathrm{k}} X190 + W190 + C1 \\
X190 + W191 &\xrightarrow{\mathrm{k}} X190 + W191 + C0 \\
X191 + W190 &\xrightarrow{\mathrm{k}} X191 + W190 + C0 \\
X191 + W191 &\xrightarrow{\mathrm{k}} X191 + W191 + C1
\end{aligned}
$$

$$
\begin{aligned}
X200 + W200 &\xrightarrow{\mathrm{k}} X200 + W200 + C1 \\
X200 + W201 &\xrightarrow{\mathrm{k}} X200 + W201 + C0 \\
X201 + W200 &\xrightarrow{\mathrm{k}} X201 + W200 + C0 \\
X201 + W201 &\xrightarrow{\mathrm{k}} X201 + W201 + C1
\end{aligned}
$$

$$
\begin{aligned}
X210 + W210 &\xrightarrow{\mathrm{k}} X210 + W210 + C1 \\
X210 + W211 &\xrightarrow{\mathrm{k}} X210 + W211 + C0 \\
X211 + W210 &\xrightarrow{\mathrm{k}} X211 + W210 + C0 \\
X211 + W211 &\xrightarrow{\mathrm{k}} X211 + W211 + C1
\end{aligned}
$$

$$
\begin{aligned}
X220 + W220 &\xrightarrow{\mathrm{k}} X220 + W220 + C1 \\
X220 + W221 &\xrightarrow{\mathrm{k}} X220 + W221 + C0 \\
X221 + W220 &\xrightarrow{\mathrm{k}} X221 + W220 + C0 \\
X221 + W221 &\xrightarrow{\mathrm{k}} X221 + W221 + C1
\end{aligned}
$$

$$
\begin{aligned}
X230 + W230 &\xrightarrow{\mathrm{k}} X230 + W230 + C1 \\
X230 + W231 &\xrightarrow{\mathrm{k}} X230 + W231 + C0 \\
X231 + W230 &\xrightarrow{\mathrm{k}} X231 + W230 + C0 \\
X231 + W231 &\xrightarrow{\mathrm{k}} X231 + W231 + C1
\end{aligned}
$$

$$
\begin{aligned}
X240 + W240 &\xrightarrow{\mathrm{k}} X240 + W240 + C1 \\
X240 + W241 &\xrightarrow{\mathrm{k}} X240 + W241 + C0 \\
X241 + W240 &\xrightarrow{\mathrm{k}} X241 + W240 + C0 \\
X241 + W241 &\xrightarrow{\mathrm{k}} X241 + W241 + C1
\end{aligned}
$$

$$
\begin{aligned}
X250 + W250 &\xrightarrow{\mathrm{k}} X250 + W250 + C1 \\
X250 + W251 &\xrightarrow{\mathrm{k}} X250 + W251 + C0 \\
X251 + W250 &\xrightarrow{\mathrm{k}} X251 + W250 + C0 \\
X251 + W251 &\xrightarrow{\mathrm{k}} X251 + W251 + C1
\end{aligned}
$$

$$
\begin{aligned}
X260 + W260 &\xrightarrow{\mathrm{k}} X260 + W260 + C1 \\
X260 + W261 &\xrightarrow{\mathrm{k}} X260 + W261 + C0 \\
X261 + W260 &\xrightarrow{\mathrm{k}} X261 + W260 + C0 \\
X261 + W261 &\xrightarrow{\mathrm{k}} X261 + W261 + C1
\end{aligned}
$$

$$
\begin{aligned}
X270 + W270 &\xrightarrow{\mathrm{k}} X270 + W270 + C1 \\
X270 + W271 &\xrightarrow{\mathrm{k}} X270 + W271 + C0 \\
X271 + W270 &\xrightarrow{\mathrm{k}} X271 + W270 + C0 \\
X271 + W271 &\xrightarrow{\mathrm{k}} X271 + W271 + C1
\end{aligned}
$$

$$
\begin{aligned}
X280 + W280 &\xrightarrow{\mathrm{k}} X280 + W280 + C1 \\
X280 + W281 &\xrightarrow{\mathrm{k}} X280 + W281 + C0 \\
X281 + W280 &\xrightarrow{\mathrm{k}} X281 + W280 + C0 \\
X281 + W281 &\xrightarrow{\mathrm{k}} X281 + W281 + C1
\end{aligned}
$$

$$
\begin{aligned}
X290 + W290 &\xrightarrow{\mathrm{k}} X290 + W290 + C1 \\
X290 + W291 &\xrightarrow{\mathrm{k}} X290 + W291 + C0 \\
X291 + W290 &\xrightarrow{\mathrm{k}} X291 + W290 + C0 \\
X291 + W291 &\xrightarrow{\mathrm{k}} X291 + W291 + C1
\end{aligned}
$$

$$
\begin{aligned}
X300 + W300 &\xrightarrow{\mathrm{k}} X300 + W300 + C1 \\
X300 + W301 &\xrightarrow{\mathrm{k}} X300 + W301 + C0 \\
X301 + W300 &\xrightarrow{\mathrm{k}} X301 + W300 + C0 \\
X301 + W301 &\xrightarrow{\mathrm{k}} X301 + W301 + C1
\end{aligned}
$$

$$
\begin{aligned}
X310 + W310 &\xrightarrow{\mathrm{k}} X310 + W310 + C1 \\
X310 + W311 &\xrightarrow{\mathrm{k}} X310 + W311 + C0 \\
X311 + W310 &\xrightarrow{\mathrm{k}} X311 + W310 + C0 \\
X311 + W311 &\xrightarrow{\mathrm{k}} X311 + W311 + C1
\end{aligned}
$$

$$
\begin{aligned}
X320 + W320 &\xrightarrow{\mathrm{k}} X320 + W320 + C1 \\
X320 + W321 &\xrightarrow{\mathrm{k}} X320 + W321 + C0 \\
X321 + W320 &\xrightarrow{\mathrm{k}} X321 + W320 + C0 \\
X321 + W321 &\xrightarrow{\mathrm{k}} X321 + W321 + C1
\end{aligned}
$$

$$
\begin{aligned}
X330 + W330 &\xrightarrow{\mathrm{k}} X330 + W330 + C1 \\
X330 + W331 &\xrightarrow{\mathrm{k}} X330 + W331 + C0 \\
X331 + W330 &\xrightarrow{\mathrm{k}} X331 + W330 + C0 \\
X331 + W331 &\xrightarrow{\mathrm{k}} X331 + W331 + C1
\end{aligned}
$$

$$
\begin{aligned}
C0 &\xrightarrow{\mathrm{k}} nth \\
C1 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
2C0 &\xrightarrow{\mathrm{k}} C11 + C0 + C0 \\
C0 + C1 &\xrightarrow{\mathrm{k}} C10 + C0 + C1 \\
C1 + C0 &\xrightarrow{\mathrm{k}} C10 + C1 + C0 \\
2C1 &\xrightarrow{\mathrm{k}} C11 + C1 + C1 \\
C10 &\xrightarrow{\mathrm{k}} nth \\
C11 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
A20 + C10 &\xrightarrow{\mathrm{k}} C20 + A20 + C10 \\
A20 + C11 &\xrightarrow{\mathrm{k}} C21 + A20 + C11 \\
A21 + C10 &\xrightarrow{\mathrm{k}} C21 + A21 + C10 \\
A21 + C11 &\xrightarrow{\mathrm{k}} C20 + A21 + C11 \\
C20 &\xrightarrow{\mathrm{k}} nth \\
C21 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
A30 + C20 &\xrightarrow{\mathrm{k}} C30 + A30 + C20 \\
A30 + C21 &\xrightarrow{\mathrm{k}} C31 + A30 + C21 \\
A31 + B30 &\xrightarrow{\mathrm{k}} C30 + A31 + B30 \\
A31 + B31 &\xrightarrow{\mathrm{k}} C31 + A31 + B31 \\
C30 &\xrightarrow{\mathrm{k}} nth \\
C31 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
C10 + C30 &\xrightarrow{\mathrm{k}} C41 + C10 + C30 \\
C10 + C31 &\xrightarrow{\mathrm{k}} C40 + C10 + C31 \\
C11 + C30 &\xrightarrow{\mathrm{k}} C40 + C11 + C30 \\
C11 + C31 &\xrightarrow{\mathrm{k}} C41 + C11 + C31 \\
C40 &\xrightarrow{\mathrm{k}} nth \\
C41 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
A50 + C40 &\xrightarrow{\mathrm{k}} C50 + A50 + C40 \\
A50 + C41 &\xrightarrow{\mathrm{k}} C51 + A50 + C41 \\
A51 + B50 &\xrightarrow{\mathrm{k}} C50 + A51 + B50 \\
A51 + B51 &\xrightarrow{\mathrm{k}} C51 + A51 + B51 \\
C50 &\xrightarrow{\mathrm{k}} nth \\
C51 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
C0 + C50 &\xrightarrow{\mathrm{k}} C60 + C0 + C50 \\
C0 + C51 &\xrightarrow{\mathrm{k}} C61 + C0 + C51 \\
C1 + C50 &\xrightarrow{\mathrm{k}} C61 + C1 + C50 \\
C1 + C51 &\xrightarrow{\mathrm{k}} C60 + C1 + C51 \\
C60 &\xrightarrow{\mathrm{k}} nth \\
C61 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
C60 &\xrightarrow{\mathrm{k}} C60 + cp \\
C61 &\xrightarrow{\mathrm{k}} C61 + cp \\
cp &\xrightarrow{\mathrm{k}} nth \\
C61 &\xrightarrow{\mathrm{k}} C61 + c \\
cp + c &\xrightarrow{\mathrm{k}} cp
\end{aligned}
$$

A.1.2 molecular ADC 3bit

$$
\begin{aligned}
i1 + T1 &\xrightarrow{k} W1 \\
i1 + x2n &\xrightarrow{k} i1 + x2p \\
T1 + x2p &\xrightarrow{k} T1 + x2n \\
\\
x2p + i1 + T2 &\xrightarrow{k} W2 + x2p \\
x2n + W2 &\xrightarrow{k} T2 + x2n + i1 \\
x2p + i1 + x1n &\xrightarrow{k} x1p + i1 + x2p \\
x2p + x1p + T2 &\xrightarrow{k} T2 + x1n + x2p \\
x2n + W1 + Tp2 &\xrightarrow{k} Wp2 + x2n \\
x2p + Wp2 &\xrightarrow{k} Tp2 + x2p + W1 \\
x2n + W1 + x1n &\xrightarrow{k} x1p + W1 + x2n \\
x2n + x1p + Tp2 &\xrightarrow{k} Tp2 + x1n + x2n \\
\\
x2p + x1p + i1 + T3 &\xrightarrow{k} W3 + x2p + x1p \\
x1n + W3 &\xrightarrow{k} T3 + i1 + x1n \\
x2n + W3 &\xrightarrow{k} T3 + i1 + x2n \\
\\
x2p + x1p + i1 + x0n &\xrightarrow{k} x0p + x2p + x1p + i1 \\
x2p + x1p + T3 + x0p &\xrightarrow{k} x0n + x2p + x1p + T3 \\
x2p + x1n + W2 + Tp3 &\xrightarrow{k} Wp3 + x2p + x1n \\
\\
x2n + Wp3 &\xrightarrow{k} Tp3 + W2 + x2n \\
x1p + Wp3 &\xrightarrow{k} Tp3 + W2 + x1p \\
x2p + x1n + W2 + x0n &\xrightarrow{k} x0p + x2p + x1n + W2 \\
x2p + x1n + Tp3 + x0p &\xrightarrow{k} x0n + x2p + x1n + Tp3 \\
x2n + x1p + W1 + Tpp3 &\xrightarrow{k} Wpp3 + x2n + x1p \\
x2p + Wpp3 &\xrightarrow{k} Tpp3 + W1 + x2p \\
x1n + Wpp3 &\xrightarrow{k} Tpp3 + W1 + x1n
\end{aligned}
$$

$$
\begin{aligned}
x2n + x1p + W1 + x0n &\xrightarrow{k} x0p + x2n + x1p + W1 \\
x2n + x1p + Tpp3 + x0p &\xrightarrow{k} x0n + x2n + x1p + Tpp3 \\
x2n + x1n + Wp2 + Tppp3 &\xrightarrow{k} Wppp3 + x2n + x1n \\
x2p + Wppp3 &\xrightarrow{k} Tppp3 + Wp2 + x2p \\
x1p + Wppp3 &\xrightarrow{k} Tppp3 + Wp2 + x1p \\
x2n + x1n + Wp2 + x0n &\xrightarrow{k} x0p + x2n + x1n + Wp2 \\
x2n + x1n + Tppp3 + x0p &\xrightarrow{k} x0n + x2n + x1n + Tppp3
\end{aligned}
$$

A.1.3 molecular DAC 3bit

$$
\begin{aligned}
b3 + o3 &\xrightarrow{k} out + b3 + m3 \\
u3 + m3 + out &\xrightarrow{k} o3 + u3 \\
b2 + o2 &\xrightarrow{k} out + b2 + m2 \\
u2 + m2 + out &\xrightarrow{k} o2 + u2 \\
b1 + o1 &\xrightarrow{k} out + b1 + m1 \\
u1 + m1 + out &\xrightarrow{k} o1 + u1 \\
i1 + T1 &\xrightarrow{k} W1 \\
i1 + u1 &\xrightarrow{k} i1 + b1 \\
\\
T1 + b1 &\xrightarrow{k} T1 + u1 \\
b1 + i1 + T2 &\xrightarrow{k} W2 + b1 \\
u1 + W2 &\xrightarrow{k} T2 + u1 + i1 \\
b1 + i1 + u2 &\xrightarrow{k} b2 + i1 + b1 \\
b1 + b2 + T2 &\xrightarrow{k} T2 + u2 + b1
\end{aligned}
$$

$$
\begin{aligned}
u1 + W1 + Tp2 &\xrightarrow{\text{k}} Wp2 + u1 \\
b1 + Wp2 &\xrightarrow{\text{k}} Tp2 + b1 + W1 \\
u1 + W1 + u2 &\xrightarrow{\text{k}} b2 + W1 + u1 \\
u1 + b2 + Tp2 &\xrightarrow{\text{k}} Tp2 + u2 + u1 \\
\\
b1 + b2 + i1 + T3 &\xrightarrow{\text{k}} W3 + b1 + b2 \\
u2 + W3 &\xrightarrow{\text{k}} T3 + i1 + u2 \\
u1 + W3 &\xrightarrow{\text{k}} T3 + i1 + u1 \\
b1 + b2 + i1 + u3 &\xrightarrow{\text{k}} b3 + b1 + b2 + i1 \\
\\
b1 + b2 + T3 + b3 &\xrightarrow{\text{k}} u3 + b1 + b2 + T3 \\
b1 + u2 + W2 + Tp3 &\xrightarrow{\text{k}} Wp3 + b1 + u2 \\
u1 + Wp3 &\xrightarrow{\text{k}} Tp3 + W2 + u1 \\
b2 + Wp3 &\xrightarrow{\text{k}} Tp3 + W2 + b2 \\
b1 + u2 + W2 + u3 &\xrightarrow{\text{k}} b3 + b1 + u2 + W2 \\
\\
b1 + u2 + Tp3 + b3 &\xrightarrow{\text{k}} u3 + b1 + u2 + Tp3 \\
u1 + b2 + W1 + Tpp3 &\xrightarrow{\text{k}} Wpp3 + u1 + b2 \\
b1 + Wpp3 &\xrightarrow{\text{k}} Tpp3 + W1 + b1 \\
u2 + Wpp3 &\xrightarrow{\text{k}} Tpp3 + W1 + u2 \\
u1 + b2 + W1 + u3 &\xrightarrow{\text{k}} b3 + u1 + b2 + W1 \\
\\
u1 + b2 + Tpp3 + b3 &\xrightarrow{\text{k}} u3 + u1 + b2 + Tpp3 \\
u1 + u2 + Wp2 + Tppp3 &\xrightarrow{\text{k}} Wppp3 + u1 + u2 \\
b1 + Wppp3 &\xrightarrow{\text{k}} Tppp3 + Wp2 + b1 \\
\\
b2 + Wppp3 &\xrightarrow{\text{k}} Tppp3 + Wp2 + b2 \\
u1 + u2 + Wp2 + u3 &\xrightarrow{\text{k}} b3 + u1 + u2 + Wp2 \\
u1 + u2 + Tppp3 + b3 &\xrightarrow{\text{k}} u3 + u1 + u2 + Tppp3
\end{aligned}
$$

A.1.4 molecular Adder 3bit

$$
\begin{aligned}
i1 + T1 &\xrightarrow{k} W1 \\
i1 + x2n &\xrightarrow{k} i1 + x2p \\
T1 + x2p &\xrightarrow{k} T1 + x2n \\
x2p + i1 + T2 &\xrightarrow{k} W2 + x2p \\
x2n + W2 &\xrightarrow{k} T2 + x2n + i1 \\
x2p + i1 + x1n &\xrightarrow{k} x1p + i1 + x2p \\
\\
x2p + x1p + T2 &\xrightarrow{k} T2 + x1n + x2p \\
x2n + W1 + Tp2 &\xrightarrow{k} Wp2 + x2n \\
x2p + Wp2 &\xrightarrow{k} Tp2 + x2p + W1 \\
x2n + W1 + x1n &\xrightarrow{k} x1p + W1 + x2n \\
x2n + x1p + Tp2 &\xrightarrow{k} Tp2 + x1n + x2n \\
\\
x2p + x1p &\xrightarrow{k} q1 + x2p + x1p \\
2q1 &\xrightarrow{k} nth \\
i1 + T3 &\xrightarrow{k} q2 \\
q1 + q2 &\xrightarrow{k} W3 \\
W3 + x1n &\xrightarrow{k} i1 + T3 + x1n \\
W3 + x2n &\xrightarrow{k} i1 + T3 + x2n \\
\\
q2 + x1n &\xrightarrow{k} i1 + T3 + x1n \\
q2 + x2n &\xrightarrow{k} i1 + T3 + x2n \\
x1n + W3 &\xrightarrow{k} T3 + i1 + x1n \\
x2n + W3 &\xrightarrow{k} T3 + i1 + x2n \\
i1 + x0n &\xrightarrow{k} q3 + i1 \\
T3 + q3 &\xrightarrow{k} x0n + T3
\end{aligned}
$$

$$
\begin{aligned}
q1 + q3 &\xrightarrow{k} x0p \\
x2p + x1p + T3 + x0p &\xrightarrow{k} x0n + x2p + x1p + T3 \\
x2p + x1n + W2 + Tp3 &\xrightarrow{k} Wp3 + x2p + x1n \\
x2n + Wp3 &\xrightarrow{k} Tp3 + W2 + x2n \\
x1p + Wp3 &\xrightarrow{k} Tp3 + W2 + x1p \\
x2p + x1n + W2 + x0n &\xrightarrow{k} x0p + x2p + x1n + W2 \\
\\
x2p + x1n + Tp3 + x0p &\xrightarrow{k} x0n + x2p + x1n + Tp3 \\
x2n + x1p + W1 + Tpp3 &\xrightarrow{k} Wpp3 + x2n + x1p \\
x2p + Wpp3 &\xrightarrow{k} Tpp3 + W1 + x2p \\
x1n + Wpp3 &\xrightarrow{k} Tpp3 + W1 + x1n \\
x2n + x1p + W1 + x0n &\xrightarrow{k} x0p + x2n + x1p + W1 \\
\\
x2n + x1p + Tpp3 + x0p &\xrightarrow{k} x0n + x2n + x1p + Tpp3 \\
x2n + x1n + Wp2 + Tppp3 &\xrightarrow{k} Wppp3 + x2n + x1n \\
x2p + Wppp3 &\xrightarrow{k} Tppp3 + Wp2 + x2p \\
x1p + Wppp3 &\xrightarrow{k} Tppp3 + Wp2 + x1p \\
\\
x2n + x1n + Wp2 + x0n &\xrightarrow{k} x0p + x2n + x1n + Wp2 \\
x2n + x1n + Tppp3 + x0p &\xrightarrow{k} x0n + x2n + x1n + Tppp3 \\
\\
i2 + T1y &\xrightarrow{k} W1y \\
i2 + y2n &\xrightarrow{k} i2 + y2p \\
T1y + y2p &\xrightarrow{k} T1y + y2n \\
y2p + i2 + T2y &\xrightarrow{k} W2y + y2p \\
y2n + W2y &\xrightarrow{k} T2y + y2n + i2 \\
\\
y2p + i2 + y1n &\xrightarrow{k} y1p + i2 + y2p \\
y2p + y1p + T2y &\xrightarrow{k} T2y + y1n + y2p \\
y2n + W1y + Tp2y &\xrightarrow{k} Wp2y + y2n
\end{aligned}
$$

$$
\begin{aligned}
y2p + Wp2y &\xrightarrow{k} Tp2y + y2p + W1y \\
y2n + W1y + y1n &\xrightarrow{k} y1p + W1y + y2n \\
y2n + y1p + Tp2y &\xrightarrow{k} Tp2y + y1n + y2n \\
y2p + y1p + i2 + T3y &\xrightarrow{k} W3y + y2p + y1p \\
\\
y1n + W3y &\xrightarrow{k} T3y + i2 + y1n \\
y2n + W3y &\xrightarrow{k} T3y + i2 + y2n \\
y2p + y1p + i2 + y0n &\xrightarrow{k} y0p + y2p + y1p + i2 \\
\\
y2p + y1p + T3y + y0p &\xrightarrow{k} y0n + y2p + y1p + T3y \\
y2p + y1n + W2y + Tp3y &\xrightarrow{k} Wp3y + y2p + y1n \\
y2n + Wp3y &\xrightarrow{k} Tp3y + W2y + y2n \\
\\
y1p + Wp3y &\xrightarrow{k} Tp3y + W2y + y1p \\
y2p + y1n + W2y + y0n &\xrightarrow{k} y0p + y2p + y1n + W2y \\
y2p + y1n + Tp3y + y0p &\xrightarrow{k} y0n + y2p + y1n + Tp3y \\
y2n + y1p + W1y + Tpp3y &\xrightarrow{k} Wpp3y + y2n + y1p \\
\\
y2p + Wpp3y &\xrightarrow{k} Tpp3y + W1y + y2p \\
y1n + Wpp3y &\xrightarrow{k} Tpp3y + W1y + y1n \\
y2n + y1p + W1y + y0n &\xrightarrow{k} y0p + y2n + y1p + W1y \\
y2n + y1p + Tpp3y + y0p &\xrightarrow{k} y0n + y2n + y1p + Tpp3y \\
\\
y2n + y1n + Wp2y + Tppp3y &\xrightarrow{k} Wppp3y + y2n + y1n \\
y2p + Wppp3y &\xrightarrow{k} Tppp3y + Wp2y + y2p \\
y1p + Wppp3y &\xrightarrow{k} Tppp3y + Wp2y + y1p \\
y2n + y1n + Wp2y + y0n &\xrightarrow{k} y0p + y2n + y1n + Wp2y \\
\\
y2n + y1n + Tppp3y + y0p &\xrightarrow{k} y0n + y2n + y1n + Tppp3y
\end{aligned}
$$

$$
\begin{aligned}
c2p + o4 &\xrightarrow{k} out + c2p + m4 \\
c2n + m4 + out &\xrightarrow{k} o4 + c2n \\
s2p + o3 &\xrightarrow{k} out + s2p + m3 \\
s2n + m3 + out &\xrightarrow{k} o3 + s2n \\
s1p + o2 &\xrightarrow{k} out + s1p + m2 \\
s1n + m2 + out &\xrightarrow{k} o2 + s1n \\
s0p + o1 &\xrightarrow{k} out + s0p + m1 \\
s0n + m1 + out &\xrightarrow{k} o1 + s0n \\
x0n + y0p &\xrightarrow{k} x0n + y0p + G1p \\
x0p + y0n &\xrightarrow{k} x0p + y0n + G1p \\
2G1p &\xrightarrow{k} nth \\
G1p + s0n &\xrightarrow{k} s0p \\
x0n + y0n &\xrightarrow{k} x0n + y0n + G1n \\
x0p + y0p &\xrightarrow{k} x0p + y0p + G1n \\
2G1n &\xrightarrow{k} nth \\
G1n + s0p &\xrightarrow{k} s0n \\
x0n + c0p &\xrightarrow{k} x0n + c0n \\
y0n + c0p &\xrightarrow{k} y0n + c0n \\
x0p + y0p &\xrightarrow{k} x0p + y0p + G2 \\
2G2 &\xrightarrow{k} nth \\
G2 + c0n &\xrightarrow{k} c0p \\
x1n + y1p &\xrightarrow{k} x1n + y1p + G3p \\
x1p + y1n &\xrightarrow{k} x1p + y1n + G3p \\
2G3p &\xrightarrow{k} nth \\
G3p + z3n &\xrightarrow{k} z3p
\end{aligned}
$$

$$
\begin{aligned}
x1n + y1n &\xrightarrow{k} x1n + y1n + G3n \\
x1p + y1p &\xrightarrow{k} x1p + y1p + G3n \\
2G3n &\xrightarrow{k} nth \\
G3n + z3p &\xrightarrow{k} z3n
\end{aligned}
$$

$$
\begin{aligned}
z3n + c0p &\xrightarrow{k} z3n + c0p + G4p \\
z3p + c0n &\xrightarrow{k} z3p + c0n + G4p \\
2G4p &\xrightarrow{k} nth \\
G4p + s1n &\xrightarrow{k} s1p
\end{aligned}
$$

$$
\begin{aligned}
z3n + c0n &\xrightarrow{k} z3n + c0n + G4n \\
z3p + c0p &\xrightarrow{k} z3p + c0p + G4n \\
2G4n &\xrightarrow{k} nth \\
G4n + s1p &\xrightarrow{k} s1n
\end{aligned}
$$

$$
\begin{aligned}
z3n + z5p &\xrightarrow{k} z3n + z5n \\
c0n + z5p &\xrightarrow{k} c0n + z5n \\
z3p + c0p &\xrightarrow{k} z3p + c0p + G5 \\
2G5 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
G5 + z5n &\xrightarrow{k} z5p \\
x1n + z6p &\xrightarrow{k} x1n + z6n \\
y1n + z6p &\xrightarrow{k} y1n + z6n \\
x1p + y1p &\xrightarrow{k} x1p + y1p + G6
\end{aligned}
$$

$$
\begin{aligned}
2G6 &\xrightarrow{k} nth \\
G6 + z6n &\xrightarrow{k} z6p \\
z5p + c1n &\xrightarrow{k} z5p + c1p \\
z6p + c1n &\xrightarrow{k} z6p + c1p
\end{aligned}
$$

$$
\begin{aligned}
z5n + z6n &\xrightarrow{k} z5n + z6n + G7 \\
2G7 &\xrightarrow{k} nth \\
G7 + c1p &\xrightarrow{k} c1n \\
x2n + y2p &\xrightarrow{k} x2n + y2p + G8p \\
x2p + y2n &\xrightarrow{k} x2p + y2n + G8p
\end{aligned}
$$

$$
\begin{aligned}
2G8p &\xrightarrow{k} nth \\
G8p + z8n &\xrightarrow{k} z8p \\
x2n + y2n &\xrightarrow{k} x2n + y2n + G8n \\
x2p + y2p &\xrightarrow{k} x2p + y2p + G8n
\end{aligned}
$$

$$
\begin{aligned}
2G8n &\xrightarrow{k} nth \\
G8n + z8p &\xrightarrow{k} z8n \\
z8n + c1p &\xrightarrow{k} z8n + c1p + G9p \\
z8p + c1n &\xrightarrow{k} z8p + c1n + G9p
\end{aligned}
$$

$$
\begin{aligned}
2G9p &\xrightarrow{k} nth \\
G9p + s2n &\xrightarrow{k} s2p \\
z8n + c1n &\xrightarrow{k} z8n + c1n + G9n \\
z8p + c1p &\xrightarrow{k} z8p + c1p + G9n
\end{aligned}
$$

$$
\begin{aligned}
2G9n &\xrightarrow{k} nth \\
G9n + s2p &\xrightarrow{k} s2n \\
z8n + z10p &\xrightarrow{k} z8n + z10n \\
c1n + z10p &\xrightarrow{k} c1n + z10n
\end{aligned}
$$

$$
\begin{aligned}
z8p + c1p &\xrightarrow{k} z8p + c1p + G10 \\
2G10 &\xrightarrow{k} nth \\
G10 + z10n &\xrightarrow{k} z10p \\
x2n + z11p &\xrightarrow{k} x2n + z11n
\end{aligned}
$$

$$
\begin{aligned}
y2n + z11p &\xrightarrow{k} y2n + z11n \\
x2p + y2p &\xrightarrow{k} x2p + y2p + G11 \\
2G11 &\xrightarrow{k} nth \\
G11 + z11n &\xrightarrow{k} z11p \\
\\
z10p + c2n &\xrightarrow{k} z10p + c2p \\
z11p + c2n &\xrightarrow{k} z11p + c2p \\
z10n + z11n &\xrightarrow{k} z10n + z11n + G12 \\
2G12 &\xrightarrow{k} nth \\
G12 + c2p &\xrightarrow{k} c2n
\end{aligned}
$$

A.1.5 molecular Markov

$$
\begin{aligned}
AV + A1 &\xrightarrow{k} BV + A1 \\
AV + A2 &\xrightarrow{k} SV + A2 \\
BV + B1 &\xrightarrow{k} CV + B1 \\
\\
BV + B2 &\xrightarrow{k} AV + B2 \\
CV + C1 &\xrightarrow{k} DV + C1 \\
CV + C2 &\xrightarrow{k} BV + C2 \\
DV + D1 &\xrightarrow{k} EV + D1 \\
\\
DV + D2 &\xrightarrow{k} CV + D2 \\
EV + E1 &\xrightarrow{k} FV + E1 \\
EV + E2 &\xrightarrow{k} DV + E2 \\
FV + F1 &\xrightarrow{k} GV + F1
\end{aligned}
$$

$$
\begin{aligned}
FV + F2 &\xrightarrow{\mathrm{k}} EV + F2 \\
GV + G1 &\xrightarrow{\mathrm{k}} HV + G1 \\
GV + G2 &\xrightarrow{\mathrm{k}} FV + G2
\end{aligned}
$$

$$
\begin{aligned}
HV + H1 &\xrightarrow{\mathrm{k}} ENDV + H1 \\
HV + H2 &\xrightarrow{\mathrm{k}} GV + H2
\end{aligned}
$$

A.1.6 $y(x) = \frac{3}{4}x^2 - x + \frac{3}{4}$ Molecular

$$
\begin{aligned}
X10 + X00 &\xrightarrow{\mathrm{k_s}} S0 \\
X10 + X01 &\xrightarrow{\mathrm{k_s}} S1 \\
X11 + X00 &\xrightarrow{\mathrm{k_s}} S1 \\
X11 + X01 &\xrightarrow{\mathrm{k_s}} S2 \\
C00 + S0 &\xrightarrow{\mathrm{k_s}} Y0 \\
C01 + S0 &\xrightarrow{\mathrm{k_s}} Y1 \\
C10 + S1 &\xrightarrow{\mathrm{k_s}} Y0 \\
C11 + S1 &\xrightarrow{\mathrm{k_s}} Y1 \\
C20 + S2 &\xrightarrow{\mathrm{k_s}} Y0 \\
C21 + S2 &\xrightarrow{\mathrm{k_s}} Y1
\end{aligned}
$$

A.1.7 molecular encoder

$$
\begin{aligned}
X &\xrightarrow{\mathrm{k}} X1 + X \\
X1 &\xrightarrow{\mathrm{k}} nth \\
T &\xrightarrow{\mathrm{k}} X0 + T
\end{aligned}
$$

$$
\begin{aligned}
X1 &\xrightarrow{\mathrm{k}} X1 + Xp \\
X0 &\xrightarrow{\mathrm{k}} nth \\
X0 + Xp &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

A.1.8 molecular decoder

$$
\begin{aligned}
Y0 &\xrightarrow{\mathrm{k}} Y0 + Yp \\
Y1 &\xrightarrow{\mathrm{k}} Y1 + Yp
\end{aligned}
$$

$$
\begin{aligned}
Yp &\xrightarrow{\mathrm{k}} nth \\
Y1 &\xrightarrow{\mathrm{k}} Y1 + Y \\
Yp + Y &\xrightarrow{\mathrm{k}} Yp
\end{aligned}
$$

A.1.9 molecular e-x

$$
\begin{aligned}
A10 + Ap10 &\xrightarrow{\mathrm{k}} C11 + A10 + Ap10 \\
A10 + Ap11 &\xrightarrow{\mathrm{k}} C11 + A10 + Ap11 \\
A11 + Ap10 &\xrightarrow{\mathrm{k}} C11 + A11 + Ap10
\end{aligned}
$$

$$
\begin{aligned}
A11 + Ap11 &\xrightarrow{\mathrm{k}} C10 + A11 + Ap11 \\
C10 &\xrightarrow{\mathrm{k}} nth \\
C11 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
A20 + C10 &\xrightarrow{\mathrm{k}} C20 + A20 + C10 \\
A20 + C11 &\xrightarrow{\mathrm{k}} C20 + A20 + C11
\end{aligned}
$$

$$
\begin{aligned}
A21 + C10 &\xrightarrow{\mathrm{k}} C20 + A21 + C10 \\
A21 + C11 &\xrightarrow{\mathrm{k}} C21 + A21 + C11 \\
C20 &\xrightarrow{\mathrm{k}} nth \\
C21 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

$$
\begin{aligned}
A10 + C20 &\xrightarrow{k} C31 + A10 + C20 \\
A10 + C21 &\xrightarrow{k} C31 + A10 + C21 \\
\\
A11 + C20 &\xrightarrow{k} C31 + A11 + C20 \\
A11 + C21 &\xrightarrow{k} C30 + A11 + C21 \\
C30 &\xrightarrow{k} nth \\
C31 &\xrightarrow{k} nth \\
\\
A40 + C30 &\xrightarrow{k} C40 + A40 + C30 \\
\\
A40 + C31 &\xrightarrow{k} C40 + A40 + C31 \\
A41 + C30 &\xrightarrow{k} C40 + A41 + C30 \\
A41 + C31 &\xrightarrow{k} C41 + A41 + C31 \\
\\
C40 &\xrightarrow{k} nth \\
C41 &\xrightarrow{k} nth \\
\\
A10 + C40 &\xrightarrow{k} C51 + A10 + C40 \\
A10 + C41 &\xrightarrow{k} C51 + A10 + C41 \\
A11 + C40 &\xrightarrow{k} C51 + A11 + C40 \\
\\
A11 + C41 &\xrightarrow{k} C50 + A11 + C41 \\
C50 &\xrightarrow{k} nth \\
C51 &\xrightarrow{k} nth \\
\\
A50 + C50 &\xrightarrow{k} C60 + A50 + C50 \\
A50 + C51 &\xrightarrow{k} C60 + A50 + C51 \\
\\
A51 + C50 &\xrightarrow{k} C60 + A51 + C50 \\
A51 + C51 &\xrightarrow{k} C61 + A51 + C51 \\
C60 &\xrightarrow{k} nth \\
C61 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
A10 + C60 &\xrightarrow{k} C71 + A10 + C60 \\
A10 + C61 &\xrightarrow{k} C71 + A10 + C61 \\
\\
A11 + C60 &\xrightarrow{k} C71 + A11 + C60 \\
A11 + C61 &\xrightarrow{k} C70 + A11 + C61 \\
C70 &\xrightarrow{k} nth \\
C71 &\xrightarrow{k} nth \\
\\
A10 + C70 &\xrightarrow{k} C81 + A10 + C70 \\
\\
A10 + C71 &\xrightarrow{k} C81 + A10 + C71 \\
A11 + C70 &\xrightarrow{k} C81 + A11 + C70 \\
A11 + C71 &\xrightarrow{k} C80 + A11 + C71 \\
C80 &\xrightarrow{k} nth \\
\\
C81 &\xrightarrow{k} nth \\
\\
x &\xrightarrow{k} x + A11 \\
A11 &\xrightarrow{k} nth \\
T &\xrightarrow{k} A10 + T \\
A11 &\xrightarrow{k} A11 + yn \\
\\
A10 &\xrightarrow{k} nth \\
A10 + yn &\xrightarrow{k} nth \\
\\
C80 &\xrightarrow{k} C80 + cp \\
C81 &\xrightarrow{k} C81 + cp \\
cp &\xrightarrow{k} nth \\
\\
C81 &\xrightarrow{k} C81 + c \\
cp + c &\xrightarrow{k} cp
\end{aligned}
$$

A.1.10 molecular bipolar sigmoid

$$
\begin{aligned}
2X0 &\xrightarrow{k} C11 + X0 + X0 \\
X0 + X1 &\xrightarrow{k} C10 + X0 + X1 \\
X1 + X0 &\xrightarrow{k} C10 + X1 + X0 \\
2X1 &\xrightarrow{k} C11 + X1 + X1 \\
C10 &\xrightarrow{k} nth \\
C11 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
A20 + C10 &\xrightarrow{k} C20 + A20 + C10 \\
A20 + C11 &\xrightarrow{k} C21 + A20 + C11 \\
A21 + C10 &\xrightarrow{k} C21 + A21 + C10 \\
A21 + C11 &\xrightarrow{k} C20 + A21 + C11 \\
C20 &\xrightarrow{k} nth \\
C21 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
A30 + C20 &\xrightarrow{k} C30 + A30 + C20 \\
A30 + C21 &\xrightarrow{k} C31 + A30 + C21 \\
A31 + B30 &\xrightarrow{k} C30 + A31 + B30 \\
A31 + B31 &\xrightarrow{k} C31 + A31 + B31 \\
C30 &\xrightarrow{k} nth \\
C31 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
C10 + C30 &\xrightarrow{k} C41 + C10 + C30 \\
C10 + C31 &\xrightarrow{k} C40 + C10 + C31 \\
C11 + C30 &\xrightarrow{k} C40 + C11 + C30 \\
C11 + C31 &\xrightarrow{k} C41 + C11 + C31 \\
C40 &\xrightarrow{k} nth \\
C41 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
A50 + C40 &\xrightarrow{k} C50 + A50 + C40 \\
A50 + C41 &\xrightarrow{k} C51 + A50 + C41 \\
A51 + B50 &\xrightarrow{k} C50 + A51 + B50 \\
A51 + B51 &\xrightarrow{k} C51 + A51 + B51 \\
C50 &\xrightarrow{k} nth \\
C51 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
X0 + C50 &\xrightarrow{k} C60 + X0 + C50 \\
X0 + C51 &\xrightarrow{k} C61 + X0 + C51 \\
X1 + C50 &\xrightarrow{k} C61 + X1 + C50 \\
X1 + C51 &\xrightarrow{k} C60 + X1 + C51 \\
C60 &\xrightarrow{k} nth \\
C61 &\xrightarrow{k} nth
\end{aligned}
$$

A.1.11 molecular unipolar sigmoid

$$
\begin{aligned}
2A10 &\xrightarrow{k} C10 + A10 + A10 \\
A10 + A11 &\xrightarrow{k} C10 + A10 + A11 \\
A11 + A10 &\xrightarrow{k} C10 + A11 + A10 \\
2A11 &\xrightarrow{k} C11 + A11 + A11 \\
C10 &\xrightarrow{k} nth \\
C11 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
A20 + C10 &\xrightarrow{k} C21 + A20 + C10 \\
A20 + C11 &\xrightarrow{k} C21 + A20 + C11 \\
A21 + C10 &\xrightarrow{k} C21 + A21 + C10 \\
A21 + C11 &\xrightarrow{k} C20 + A21 + C11 \\
C20 &\xrightarrow{k} nth \\
C21 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
A30 + C20 &\xrightarrow{k} C30 + A30 + C20 \\
A30 + C21 &\xrightarrow{k} C30 + A30 + C21 \\
A31 + C20 &\xrightarrow{k} C30 + A31 + C20 \\
A31 + C21 &\xrightarrow{k} C31 + A31 + C21 \\
C30 &\xrightarrow{k} nth \\
C31 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
C10 + C30 &\xrightarrow{k} C41 + C10 + C30 \\
C10 + C31 &\xrightarrow{k} C41 + C10 + C31 \\
C11 + C30 &\xrightarrow{k} C41 + C11 + C30 \\
C11 + C31 &\xrightarrow{k} C40 + C11 + C31 \\
C40 &\xrightarrow{k} nth \\
C41 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
A50 + C40 &\xrightarrow{k} C50 + A50 + C40 \\
A50 + C41 &\xrightarrow{k} C50 + A50 + C41 \\
A51 + C40 &\xrightarrow{k} C50 + A51 + C40 \\
A51 + C41 &\xrightarrow{k} C51 + A51 + C41 \\
C50 &\xrightarrow{k} nth \\
C51 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
A10 + C50 &\xrightarrow{k} C61 + A10 + C50 \\
A10 + C51 &\xrightarrow{k} C61 + A10 + C51 \\
A11 + C50 &\xrightarrow{k} C61 + A11 + C50 \\
A11 + C51 &\xrightarrow{k} C60 + A11 + C51 \\
C60 &\xrightarrow{k} nth \\
C61 &\xrightarrow{k} nth
\end{aligned}
$$

$$
\begin{aligned}
C60 + A70 &\xrightarrow{\mathrm{k}} C71 + C60 + A70 \\
C60 + A71 &\xrightarrow{\mathrm{k}} C71 + C60 + A71 \\
C61 + A70 &\xrightarrow{\mathrm{k}} C71 + C61 + A70 \\
C61 + A71 &\xrightarrow{\mathrm{k}} C70 + C61 + A71 \\
C70 &\xrightarrow{\mathrm{k}} nth \\
C71 &\xrightarrow{\mathrm{k}} nth
\end{aligned}
$$

A.1.12 molecular Fully async FIR

$$
\begin{aligned}
X + a4 &\xrightarrow{\mathrm{k_s}} A + Cp \\
2A &\xrightarrow{\mathrm{k_f}} Dpp \\
2Cp &\xrightarrow{\mathrm{k_f}} yp \\
Dp + a4 &\xrightarrow{\mathrm{k_s}} yp \\
yp + a1 &\xrightarrow{\mathrm{k_s}} Y \\
Dpp + a2 &\xrightarrow{\mathrm{k_s}} Dp \\
2src &\xrightarrow{\mathrm{k_s}} 2src + a1 \\
2src &\xrightarrow{\mathrm{k_s}} 2src + a2 \\
2src &\xrightarrow{\mathrm{k_s}} 2src + a3 \\
2src &\xrightarrow{\mathrm{k_s}} 2src + a4 \\
Dp + a1 &\xrightarrow{\mathrm{k_f}} Dp \\
X + a1 &\xrightarrow{\mathrm{k_f}} X \\
yp + a2 &\xrightarrow{\mathrm{k_f}} yp \\
Dpp + a3 &\xrightarrow{\mathrm{k_f}} Dpp \\
Y + a4 &\xrightarrow{\mathrm{k_f}} y
\end{aligned}
$$

A.1.13 molecular Fully async IIR

$$
\begin{aligned}
D1 + a4 &\xrightarrow{\mathrm{k_s}} Dp2 + Cp + F \\
2Cp &\xrightarrow{\mathrm{k_f}} C2 \\
2C2 &\xrightarrow{\mathrm{k_f}} C4 \\
2C4 &\xrightarrow{\mathrm{k_f}} yp \\
2F &\xrightarrow{\mathrm{k_f}} F2 \\
2F2 &\xrightarrow{\mathrm{k_f}} F4 \\
2F4 &\xrightarrow{\mathrm{k_f}} x \\
D2 + a4 &\xrightarrow{\mathrm{k_s}} Ep + H \\
2Ep &\xrightarrow{\mathrm{k_f}} E2 \\
2E2 &\xrightarrow{\mathrm{k_f}} E4 \\
2E4 &\xrightarrow{\mathrm{k_f}} yp \\
2H &\xrightarrow{\mathrm{k_f}} H2 \\
2H2 &\xrightarrow{\mathrm{k_f}} H4 \\
2H4 &\xrightarrow{\mathrm{k_f}} x \\
x + a1 &\xrightarrow{\mathrm{k_s}} A + Dp1 \\
2A &\xrightarrow{\mathrm{k_f}} A2 \\
2A2 &\xrightarrow{\mathrm{k_f}} A4 \\
2A4 &\xrightarrow{\mathrm{k_f}} yp \\
Dp1 + a2 &\xrightarrow{\mathrm{k_s}} D1 \\
Dp2 + a2 &\xrightarrow{\mathrm{k_s}} D2
\end{aligned}
$$

$$
\begin{aligned}
2src &\xrightarrow{k_s} 2src + a1 \\
2src &\xrightarrow{k_s} 2src + a2 \\
2src &\xrightarrow{k_s} 2src + a3 \\
2src &\xrightarrow{k_s} 2src + a4 \\
D1 + a1 &\xrightarrow{k_f} D1 \\
D2 + a1 &\xrightarrow{k_f} D2
\end{aligned}
$$

$$
\begin{aligned}
x + a2 &\xrightarrow{k_f} x \\
Dp1 + a3 &\xrightarrow{k_f} Dp1 \\
Dp2 + a3 &\xrightarrow{k_f} Dp2 \\
yp + a4 &\xrightarrow{k_f} yp
\end{aligned}
$$

A.2 DNA Reactions

A.2.1 perceptron DNA

$$
\begin{aligned}
X10 + gateL[1] &\xrightarrow{k} gateH[1] + strandB[1] \\
gateH[1] + strandB[1] &\xrightarrow{q_{max}} X10 + gateL[1] \\
W10 + gateH[1] &\xrightarrow{q_{max}} strandO[1] \\
strandO[1] + gateT[1] &\xrightarrow{q_{max}} X10 + W10 + C1
\end{aligned}
$$

$$
\begin{aligned}
X10 + gateL[2] &\xrightarrow{k} gateH[2] + strandB[2] \\
gateH[2] + strandB[2] &\xrightarrow{q_{max}} X10 + gateL[2] \\
W11 + gateH[2] &\xrightarrow{q_{max}} strandO[2] \\
strandO[2] + gateT[2] &\xrightarrow{q_{max}} X10 + W11 + C0
\end{aligned}
$$

$$
\begin{aligned}
X11 + gateL[3] &\xrightarrow{\text{k}} gateH[3] + strandB[3] \\
gateH[3] + strandB[3] &\xrightarrow{\text{q}_{\max}} X11 + gateL[3] \\
W10 + gateH[3] &\xrightarrow{\text{q}_{\max}} strandO[3] \\
strandO[3] + gateT[3] &\xrightarrow{\text{q}_{\max}} X11 + W10 + C0 \\
\\
X11 + gateL[4] &\xrightarrow{\text{k}} gateH[4] + strandB[4] \\
gateH[4] + strandB[4] &\xrightarrow{\text{q}_{\max}} X11 + gateL[4] \\
W11 + gateH[4] &\xrightarrow{\text{q}_{\max}} strandO[4] \\
strandO[4] + gateT[4] &\xrightarrow{\text{q}_{\max}} X11 + W11 + C1 \\
\\
X20 + gateL[5] &\xrightarrow{\text{k}} gateH[5] + strandB[5] \\
gateH[5] + strandB[5] &\xrightarrow{\text{q}_{\max}} X20 + gateL[5] \\
W20 + gateH[5] &\xrightarrow{\text{q}_{\max}} strandO[5] \\
strandO[5] + gateT[5] &\xrightarrow{\text{q}_{\max}} X20 + W20 + C1 \\
\\
X20 + gateL[6] &\xrightarrow{\text{k}} gateH[6] + strandB[6] \\
gateH[6] + strandB[6] &\xrightarrow{\text{q}_{\max}} X20 + gateL[6] \\
W21 + gateH[6] &\xrightarrow{\text{q}_{\max}} strandO[6] \\
strandO[6] + gateT[6] &\xrightarrow{\text{q}_{\max}} X20 + W21 + C0 \\
\\
X21 + gateL[7] &\xrightarrow{\text{k}} gateH[7] + strandB[7] \\
gateH[7] + strandB[7] &\xrightarrow{\text{q}_{\max}} X21 + gateL[7] \\
W20 + gateH[7] &\xrightarrow{\text{q}_{\max}} strandO[7] \\
strandO[7] + gateT[7] &\xrightarrow{\text{q}_{\max}} X21 + W20 + C0 \\
\\
X21 + gateL[8] &\xrightarrow{\text{k}} gateH[8] + strandB[8] \\
gateH[8] + strandB[8] &\xrightarrow{\text{q}_{\max}} X21 + gateL[8] \\
W21 + gateH[8] &\xrightarrow{\text{q}_{\max}} strandO[8] \\
strandO[8] + gateT[8] &\xrightarrow{\text{q}_{\max}} X21 + W21 + C1
\end{aligned}
$$

$$
\begin{aligned}
X30 + gateL[9] &\xrightarrow{\text{k}} gateH[9] + strandB[9] \\
gateH[9] + strandB[9] &\xrightarrow{\text{q}_{\max}} X30 + gateL[9] \\
W30 + gateH[9] &\xrightarrow{\text{q}_{\max}} strandO[9] \\
strandO[9] + gateT[9] &\xrightarrow{\text{q}_{\max}} X30 + W30 + C1 \\
\\
X30 + gateL[10] &\xrightarrow{\text{k}} gateH[10] + strandB[10] \\
gateH[10] + strandB[10] &\xrightarrow{\text{q}_{\max}} X30 + gateL[10] \\
W31 + gateH[10] &\xrightarrow{\text{q}_{\max}} strandO[10] \\
strandO[10] + gateT[10] &\xrightarrow{\text{q}_{\max}} X30 + W31 + C0 \\
\\
X31 + gateL[11] &\xrightarrow{\text{k}} gateH[11] + strandB[11] \\
gateH[11] + strandB[11] &\xrightarrow{\text{q}_{\max}} X31 + gateL[11] \\
W30 + gateH[11] &\xrightarrow{\text{q}_{\max}} strandO[11] \\
strandO[11] + gateT[11] &\xrightarrow{\text{q}_{\max}} X31 + W30 + C0 \\
\\
X31 + gateL[12] &\xrightarrow{\text{k}} gateH[12] + strandB[12] \\
gateH[12] + strandB[12] &\xrightarrow{\text{q}_{\max}} X31 + gateL[12] \\
W31 + gateH[12] &\xrightarrow{\text{q}_{\max}} strandO[12] \\
strandO[12] + gateT[12] &\xrightarrow{\text{q}_{\max}} X31 + W31 + C1 \\
\\
X40 + gateL[13] &\xrightarrow{\text{k}} gateH[13] + strandB[13] \\
gateH[13] + strandB[13] &\xrightarrow{\text{q}_{\max}} X40 + gateL[13] \\
W40 + gateH[13] &\xrightarrow{\text{q}_{\max}} strandO[13] \\
strandO[13] + gateT[13] &\xrightarrow{\text{q}_{\max}} X40 + W40 + C1 \\
\\
X40 + gateL[14] &\xrightarrow{\text{k}} gateH[14] + strandB[14] \\
gateH[14] + strandB[14] &\xrightarrow{\text{q}_{\max}} X40 + gateL[14] \\
W41 + gateH[14] &\xrightarrow{\text{q}_{\max}} strandO[14] \\
strandO[14] + gateT[14] &\xrightarrow{\text{q}_{\max}} X40 + W41 + C0
\end{aligned}
$$

$$
\begin{aligned}
X41 + gateL[15] &\xrightarrow{\text{k}} gateH[15] + strandB[15] \\
gateH[15] + strandB[15] &\xrightarrow{\text{q}_{\max}} X41 + gateL[15] \\
W40 + gateH[15] &\xrightarrow{\text{q}_{\max}} strandO[15] \\
strandO[15] + gateT[15] &\xrightarrow{\text{q}_{\max}} X41 + W40 + C0 \\
\\
X41 + gateL[16] &\xrightarrow{\text{k}} gateH[16] + strandB[16] \\
gateH[16] + strandB[16] &\xrightarrow{\text{q}_{\max}} X41 + gateL[16] \\
W41 + gateH[16] &\xrightarrow{\text{q}_{\max}} strandO[16] \\
strandO[16] + gateT[16] &\xrightarrow{\text{q}_{\max}} X41 + W41 + C1 \\
\\
X50 + gateL[17] &\xrightarrow{\text{k}} gateH[17] + strandB[17] \\
gateH[17] + strandB[17] &\xrightarrow{\text{q}_{\max}} X50 + gateL[17] \\
W50 + gateH[17] &\xrightarrow{\text{q}_{\max}} strandO[17] \\
strandO[17] + gateT[17] &\xrightarrow{\text{q}_{\max}} X50 + W50 + C1 \\
\\
X50 + gateL[18] &\xrightarrow{\text{k}} gateH[18] + strandB[18] \\
gateH[18] + strandB[18] &\xrightarrow{\text{q}_{\max}} X50 + gateL[18] \\
W51 + gateH[18] &\xrightarrow{\text{q}_{\max}} strandO[18] \\
strandO[18] + gateT[18] &\xrightarrow{\text{q}_{\max}} X50 + W51 + C0 \\
\\
X51 + gateL[19] &\xrightarrow{\text{k}} gateH[19] + strandB[19] \\
gateH[19] + strandB[19] &\xrightarrow{\text{q}_{\max}} X51 + gateL[19] \\
W50 + gateH[19] &\xrightarrow{\text{q}_{\max}} strandO[19] \\
strandO[19] + gateT[19] &\xrightarrow{\text{q}_{\max}} X51 + W50 + C0 \\
\\
X51 + gateL[20] &\xrightarrow{\text{k}} gateH[20] + strandB[20] \\
gateH[20] + strandB[20] &\xrightarrow{\text{q}_{\max}} X51 + gateL[20] \\
W51 + gateH[20] &\xrightarrow{\text{q}_{\max}} strandO[20] \\
strandO[20] + gateT[20] &\xrightarrow{\text{q}_{\max}} X51 + W51 + C1
\end{aligned}
$$

$$
\begin{aligned}
X60 + gateL[21] &\xrightarrow{\mathrm{k}} gateH[21] + strandB[21] \\
gateH[21] + strandB[21] &\xrightarrow{\mathrm{q_{max}}} X60 + gateL[21] \\
W60 + gateH[21] &\xrightarrow{\mathrm{q_{max}}} strandO[21] \\
strandO[21] + gateT[21] &\xrightarrow{\mathrm{q_{max}}} X60 + W60 + C1 \\
\\
X60 + gateL[22] &\xrightarrow{\mathrm{k}} gateH[22] + strandB[22] \\
gateH[22] + strandB[22] &\xrightarrow{\mathrm{q_{max}}} X60 + gateL[22] \\
W61 + gateH[22] &\xrightarrow{\mathrm{q_{max}}} strandO[22] \\
strandO[22] + gateT[22] &\xrightarrow{\mathrm{q_{max}}} X60 + W61 + C0 \\
\\
X61 + gateL[23] &\xrightarrow{\mathrm{k}} gateH[23] + strandB[23] \\
gateH[23] + strandB[23] &\xrightarrow{\mathrm{q_{max}}} X61 + gateL[23] \\
W60 + gateH[23] &\xrightarrow{\mathrm{q_{max}}} strandO[23] \\
strandO[23] + gateT[23] &\xrightarrow{\mathrm{q_{max}}} X61 + W60 + C0 \\
\\
X61 + gateL[24] &\xrightarrow{\mathrm{k}} gateH[24] + strandB[24] \\
gateH[24] + strandB[24] &\xrightarrow{\mathrm{q_{max}}} X61 + gateL[24] \\
W61 + gateH[24] &\xrightarrow{\mathrm{q_{max}}} strandO[24] \\
strandO[24] + gateT[24] &\xrightarrow{\mathrm{q_{max}}} X61 + W61 + C1 \\
\\
X70 + gateL[25] &\xrightarrow{\mathrm{k}} gateH[25] + strandB[25] \\
gateH[25] + strandB[25] &\xrightarrow{\mathrm{q_{max}}} X70 + gateL[25] \\
W70 + gateH[25] &\xrightarrow{\mathrm{q_{max}}} strandO[25] \\
strandO[25] + gateT[25] &\xrightarrow{\mathrm{q_{max}}} X70 + W70 + C1 \\
\\
X70 + gateL[26] &\xrightarrow{\mathrm{k}} gateH[26] + strandB[26] \\
gateH[26] + strandB[26] &\xrightarrow{\mathrm{q_{max}}} X70 + gateL[26] \\
W71 + gateH[26] &\xrightarrow{\mathrm{q_{max}}} strandO[26] \\
strandO[26] + gateT[26] &\xrightarrow{\mathrm{q_{max}}} X70 + W71 + C0
\end{aligned}
$$

$$
\begin{aligned}
X71 + gateL[27] &\xrightarrow{\mathrm{k}} gateH[27] + strandB[27] \\
gateH[27] + strandB[27] &\xrightarrow{\mathrm{q_{max}}} X71 + gateL[27] \\
W70 + gateH[27] &\xrightarrow{\mathrm{q_{max}}} strandO[27] \\
strandO[27] + gateT[27] &\xrightarrow{\mathrm{q_{max}}} X71 + W70 + C0 \\
\\
X71 + gateL[28] &\xrightarrow{\mathrm{k}} gateH[28] + strandB[28] \\
gateH[28] + strandB[28] &\xrightarrow{\mathrm{q_{max}}} X71 + gateL[28] \\
W71 + gateH[28] &\xrightarrow{\mathrm{q_{max}}} strandO[28] \\
strandO[28] + gateT[28] &\xrightarrow{\mathrm{q_{max}}} X71 + W71 + C1 \\
\\
X80 + gateL[29] &\xrightarrow{\mathrm{k}} gateH[29] + strandB[29] \\
gateH[29] + strandB[29] &\xrightarrow{\mathrm{q_{max}}} X80 + gateL[29] \\
W80 + gateH[29] &\xrightarrow{\mathrm{q_{max}}} strandO[29] \\
strandO[29] + gateT[29] &\xrightarrow{\mathrm{q_{max}}} X80 + W80 + C1 \\
\\
X80 + gateL[30] &\xrightarrow{\mathrm{k}} gateH[30] + strandB[30] \\
gateH[30] + strandB[30] &\xrightarrow{\mathrm{q_{max}}} X80 + gateL[30] \\
W81 + gateH[30] &\xrightarrow{\mathrm{q_{max}}} strandO[30] \\
strandO[30] + gateT[30] &\xrightarrow{\mathrm{q_{max}}} X80 + W81 + C0 \\
\\
X81 + gateL[31] &\xrightarrow{\mathrm{k}} gateH[31] + strandB[31] \\
gateH[31] + strandB[31] &\xrightarrow{\mathrm{q_{max}}} X81 + gateL[31] \\
W80 + gateH[31] &\xrightarrow{\mathrm{q_{max}}} strandO[31] \\
strandO[31] + gateT[31] &\xrightarrow{\mathrm{q_{max}}} X81 + W80 + C0 \\
\\
X81 + gateL[32] &\xrightarrow{\mathrm{k}} gateH[32] + strandB[32] \\
gateH[32] + strandB[32] &\xrightarrow{\mathrm{q_{max}}} X81 + gateL[32] \\
W81 + gateH[32] &\xrightarrow{\mathrm{q_{max}}} strandO[32] \\
strandO[32] + gateT[32] &\xrightarrow{\mathrm{q_{max}}} X81 + W81 + C1
\end{aligned}
$$

$$
\begin{aligned}
X90 + gateL[33] &\xrightarrow{\text{k}} gateH[33] + strandB[33] \\
gateH[33] + strandB[33] &\xrightarrow{\text{q}_{\max}} X90 + gateL[33] \\
W90 + gateH[33] &\xrightarrow{\text{q}_{\max}} strandO[33] \\
strandO[33] + gateT[33] &\xrightarrow{\text{q}_{\max}} X90 + W90 + C1 \\
\\
X90 + gateL[34] &\xrightarrow{\text{k}} gateH[34] + strandB[34] \\
gateH[34] + strandB[34] &\xrightarrow{\text{q}_{\max}} X90 + gateL[34] \\
W91 + gateH[34] &\xrightarrow{\text{q}_{\max}} strandO[34] \\
strandO[34] + gateT[34] &\xrightarrow{\text{q}_{\max}} X90 + W91 + C0 \\
\\
X91 + gateL[35] &\xrightarrow{\text{k}} gateH[35] + strandB[35] \\
gateH[35] + strandB[35] &\xrightarrow{\text{q}_{\max}} X91 + gateL[35] \\
W90 + gateH[35] &\xrightarrow{\text{q}_{\max}} strandO[35] \\
strandO[35] + gateT[35] &\xrightarrow{\text{q}_{\max}} X91 + W90 + C0 \\
\\
X91 + gateL[36] &\xrightarrow{\text{k}} gateH[36] + strandB[36] \\
gateH[36] + strandB[36] &\xrightarrow{\text{q}_{\max}} X91 + gateL[36] \\
W91 + gateH[36] &\xrightarrow{\text{q}_{\max}} strandO[36] \\
strandO[36] + gateT[36] &\xrightarrow{\text{q}_{\max}} X91 + W91 + C1 \\
\\
X100 + gateL[37] &\xrightarrow{\text{k}} gateH[37] + strandB[37] \\
gateH[37] + strandB[37] &\xrightarrow{\text{q}_{\max}} X100 + gateL[37] \\
W100 + gateH[37] &\xrightarrow{\text{q}_{\max}} strandO[37] \\
strandO[37] + gateT[37] &\xrightarrow{\text{q}_{\max}} X100 + W100 + C1 \\
\\
X100 + gateL[38] &\xrightarrow{\text{k}} gateH[38] + strandB[38] \\
gateH[38] + strandB[38] &\xrightarrow{\text{q}_{\max}} X100 + gateL[38] \\
W101 + gateH[38] &\xrightarrow{\text{q}_{\max}} strandO[38] \\
strandO[38] + gateT[38] &\xrightarrow{\text{q}_{\max}} X100 + W101 + C0
\end{aligned}
$$

$$
\begin{aligned}
X101 + gateL[39] &\xrightarrow{\mathrm{k}} gateH[39] + strandB[39] \\
gateH[39] + strandB[39] &\xrightarrow{\mathrm{q_{max}}} X101 + gateL[39] \\
W100 + gateH[39] &\xrightarrow{\mathrm{q_{max}}} strandO[39] \\
strandO[39] + gateT[39] &\xrightarrow{\mathrm{q_{max}}} X101 + W100 + C0 \\
\\
X101 + gateL[40] &\xrightarrow{\mathrm{k}} gateH[40] + strandB[40] \\
gateH[40] + strandB[40] &\xrightarrow{\mathrm{q_{max}}} X101 + gateL[40] \\
W101 + gateH[40] &\xrightarrow{\mathrm{q_{max}}} strandO[40] \\
strandO[40] + gateT[40] &\xrightarrow{\mathrm{q_{max}}} X101 + W101 + C1 \\
\\
X110 + gateL[41] &\xrightarrow{\mathrm{k}} gateH[41] + strandB[41] \\
gateH[41] + strandB[41] &\xrightarrow{\mathrm{q_{max}}} X110 + gateL[41] \\
W110 + gateH[41] &\xrightarrow{\mathrm{q_{max}}} strandO[41] \\
strandO[41] + gateT[41] &\xrightarrow{\mathrm{q_{max}}} X110 + W110 + C1 \\
\\
X110 + gateL[42] &\xrightarrow{\mathrm{k}} gateH[42] + strandB[42] \\
gateH[42] + strandB[42] &\xrightarrow{\mathrm{q_{max}}} X110 + gateL[42] \\
W111 + gateH[42] &\xrightarrow{\mathrm{q_{max}}} strandO[42] \\
strandO[42] + gateT[42] &\xrightarrow{\mathrm{q_{max}}} X110 + W111 + C0 \\
\\
X111 + gateL[43] &\xrightarrow{\mathrm{k}} gateH[43] + strandB[43] \\
gateH[43] + strandB[43] &\xrightarrow{\mathrm{q_{max}}} X111 + gateL[43] \\
W110 + gateH[43] &\xrightarrow{\mathrm{q_{max}}} strandO[43] \\
strandO[43] + gateT[43] &\xrightarrow{\mathrm{q_{max}}} X111 + W110 + C0 \\
\\
X111 + gateL[44] &\xrightarrow{\mathrm{k}} gateH[44] + strandB[44] \\
gateH[44] + strandB[44] &\xrightarrow{\mathrm{q_{max}}} X111 + gateL[44] \\
W111 + gateH[44] &\xrightarrow{\mathrm{q_{max}}} strandO[44] \\
strandO[44] + gateT[44] &\xrightarrow{\mathrm{q_{max}}} X111 + W111 + C1
\end{aligned}
$$

$$
\begin{aligned}
X120 + gateL[45] &\xrightarrow{\text{k}} gateH[45] + strandB[45] \\
gateH[45] + strandB[45] &\xrightarrow{\text{q}_{\max}} X120 + gateL[45] \\
W120 + gateH[45] &\xrightarrow{\text{q}_{\max}} strandO[45] \\
strandO[45] + gateT[45] &\xrightarrow{\text{q}_{\max}} X120 + W120 + C1 \\
\\
X120 + gateL[46] &\xrightarrow{\text{k}} gateH[46] + strandB[46] \\
gateH[46] + strandB[46] &\xrightarrow{\text{q}_{\max}} X120 + gateL[46] \\
W121 + gateH[46] &\xrightarrow{\text{q}_{\max}} strandO[46] \\
strandO[46] + gateT[46] &\xrightarrow{\text{q}_{\max}} X120 + W121 + C0 \\
\\
X121 + gateL[47] &\xrightarrow{\text{k}} gateH[47] + strandB[47] \\
gateH[47] + strandB[47] &\xrightarrow{\text{q}_{\max}} X121 + gateL[47] \\
W120 + gateH[47] &\xrightarrow{\text{q}_{\max}} strandO[47] \\
strandO[47] + gateT[47] &\xrightarrow{\text{q}_{\max}} X121 + W120 + C0 \\
\\
X121 + gateL[48] &\xrightarrow{\text{k}} gateH[48] + strandB[48] \\
gateH[48] + strandB[48] &\xrightarrow{\text{q}_{\max}} X121 + gateL[48] \\
W121 + gateH[48] &\xrightarrow{\text{q}_{\max}} strandO[48] \\
strandO[48] + gateT[48] &\xrightarrow{\text{q}_{\max}} X121 + W121 + C1 \\
\\
X130 + gateL[49] &\xrightarrow{\text{k}} gateH[49] + strandB[49] \\
gateH[49] + strandB[49] &\xrightarrow{\text{q}_{\max}} X130 + gateL[49] \\
W130 + gateH[49] &\xrightarrow{\text{q}_{\max}} strandO[49] \\
strandO[49] + gateT[49] &\xrightarrow{\text{q}_{\max}} X130 + W130 + C1 \\
\\
X130 + gateL[50] &\xrightarrow{\text{k}} gateH[50] + strandB[50] \\
gateH[50] + strandB[50] &\xrightarrow{\text{q}_{\max}} X130 + gateL[50] \\
W131 + gateH[50] &\xrightarrow{\text{q}_{\max}} strandO[50] \\
strandO[50] + gateT[50] &\xrightarrow{\text{q}_{\max}} X130 + W131 + C0
\end{aligned}
$$

$$
\begin{aligned}
X131 + gateL[51] &\xrightarrow{\mathrm{k}} gateH[51] + strandB[51] \\
gateH[51] + strandB[51] &\xrightarrow{\mathrm{q_{max}}} X131 + gateL[51] \\
W130 + gateH[51] &\xrightarrow{\mathrm{q_{max}}} strandO[51] \\
strandO[51] + gateT[51] &\xrightarrow{\mathrm{q_{max}}} X131 + W130 + C0 \\
\\
X131 + gateL[52] &\xrightarrow{\mathrm{k}} gateH[52] + strandB[52] \\
gateH[52] + strandB[52] &\xrightarrow{\mathrm{q_{max}}} X131 + gateL[52] \\
W131 + gateH[52] &\xrightarrow{\mathrm{q_{max}}} strandO[52] \\
strandO[52] + gateT[52] &\xrightarrow{\mathrm{q_{max}}} X131 + W131 + C1 \\
\\
X140 + gateL[53] &\xrightarrow{\mathrm{k}} gateH[53] + strandB[53] \\
gateH[53] + strandB[53] &\xrightarrow{\mathrm{q_{max}}} X140 + gateL[53] \\
W140 + gateH[53] &\xrightarrow{\mathrm{q_{max}}} strandO[53] \\
strandO[53] + gateT[53] &\xrightarrow{\mathrm{q_{max}}} X140 + W140 + C1 \\
\\
X140 + gateL[54] &\xrightarrow{\mathrm{k}} gateH[54] + strandB[54] \\
gateH[54] + strandB[54] &\xrightarrow{\mathrm{q_{max}}} X140 + gateL[54] \\
W141 + gateH[54] &\xrightarrow{\mathrm{q_{max}}} strandO[54] \\
strandO[54] + gateT[54] &\xrightarrow{\mathrm{q_{max}}} X140 + W141 + C0 \\
\\
X141 + gateL[55] &\xrightarrow{\mathrm{k}} gateH[55] + strandB[55] \\
gateH[55] + strandB[55] &\xrightarrow{\mathrm{q_{max}}} X141 + gateL[55] \\
W140 + gateH[55] &\xrightarrow{\mathrm{q_{max}}} strandO[55] \\
strandO[55] + gateT[55] &\xrightarrow{\mathrm{q_{max}}} X141 + W140 + C0 \\
\\
X141 + gateL[56] &\xrightarrow{\mathrm{k}} gateH[56] + strandB[56] \\
gateH[56] + strandB[56] &\xrightarrow{\mathrm{q_{max}}} X141 + gateL[56] \\
W141 + gateH[56] &\xrightarrow{\mathrm{q_{max}}} strandO[56] \\
strandO[56] + gateT[56] &\xrightarrow{\mathrm{q_{max}}} X141 + W141 + C1
\end{aligned}
$$

$$
\begin{aligned}
X150 + gateL[57] &\xrightarrow{\mathrm{k}} gateH[57] + strandB[57] \\
gateH[57] + strandB[57] &\xrightarrow{\mathrm{q_{max}}} X150 + gateL[57] \\
W150 + gateH[57] &\xrightarrow{\mathrm{q_{max}}} strandO[57] \\
strandO[57] + gateT[57] &\xrightarrow{\mathrm{q_{max}}} X150 + W150 + C1 \\
\\
X150 + gateL[58] &\xrightarrow{\mathrm{k}} gateH[58] + strandB[58] \\
gateH[58] + strandB[58] &\xrightarrow{\mathrm{q_{max}}} X150 + gateL[58] \\
W151 + gateH[58] &\xrightarrow{\mathrm{q_{max}}} strandO[58] \\
strandO[58] + gateT[58] &\xrightarrow{\mathrm{q_{max}}} X150 + W151 + C0 \\
\\
X151 + gateL[59] &\xrightarrow{\mathrm{k}} gateH[59] + strandB[59] \\
gateH[59] + strandB[59] &\xrightarrow{\mathrm{q_{max}}} X151 + gateL[59] \\
W150 + gateH[59] &\xrightarrow{\mathrm{q_{max}}} strandO[59] \\
strandO[59] + gateT[59] &\xrightarrow{\mathrm{q_{max}}} X151 + W150 + C0 \\
\\
X151 + gateL[60] &\xrightarrow{\mathrm{k}} gateH[60] + strandB[60] \\
gateH[60] + strandB[60] &\xrightarrow{\mathrm{q_{max}}} X151 + gateL[60] \\
W151 + gateH[60] &\xrightarrow{\mathrm{q_{max}}} strandO[60] \\
strandO[60] + gateT[60] &\xrightarrow{\mathrm{q_{max}}} X151 + W151 + C1 \\
\\
X160 + gateL[61] &\xrightarrow{\mathrm{k}} gateH[61] + strandB[61] \\
gateH[61] + strandB[61] &\xrightarrow{\mathrm{q_{max}}} X160 + gateL[61] \\
W160 + gateH[61] &\xrightarrow{\mathrm{q_{max}}} strandO[61] \\
strandO[61] + gateT[61] &\xrightarrow{\mathrm{q_{max}}} X160 + W160 + C1 \\
\\
X160 + gateL[62] &\xrightarrow{\mathrm{k}} gateH[62] + strandB[62] \\
gateH[62] + strandB[62] &\xrightarrow{\mathrm{q_{max}}} X160 + gateL[62] \\
W161 + gateH[62] &\xrightarrow{\mathrm{q_{max}}} strandO[62] \\
strandO[62] + gateT[62] &\xrightarrow{\mathrm{q_{max}}} X160 + W161 + C0
\end{aligned}
$$

$$
\begin{aligned}
X161 + gateL[63] &\xrightarrow{\mathrm{k}} gateH[63] + strandB[63] \\
gateH[63] + strandB[63] &\xrightarrow{\mathrm{q_{max}}} X161 + gateL[63] \\
W160 + gateH[63] &\xrightarrow{\mathrm{q_{max}}} strandO[63] \\
strandO[63] + gateT[63] &\xrightarrow{\mathrm{q_{max}}} X161 + W160 + C0 \\
\\
X161 + gateL[64] &\xrightarrow{\mathrm{k}} gateH[64] + strandB[64] \\
gateH[64] + strandB[64] &\xrightarrow{\mathrm{q_{max}}} X161 + gateL[64] \\
W161 + gateH[64] &\xrightarrow{\mathrm{q_{max}}} strandO[64] \\
strandO[64] + gateT[64] &\xrightarrow{\mathrm{q_{max}}} X161 + W161 + C1 \\
\\
X170 + gateL[65] &\xrightarrow{\mathrm{k}} gateH[65] + strandB[65] \\
gateH[65] + strandB[65] &\xrightarrow{\mathrm{q_{max}}} X170 + gateL[65] \\
W170 + gateH[65] &\xrightarrow{\mathrm{q_{max}}} strandO[65] \\
strandO[65] + gateT[65] &\xrightarrow{\mathrm{q_{max}}} X170 + W170 + C1 \\
\\
X170 + gateL[66] &\xrightarrow{\mathrm{k}} gateH[66] + strandB[66] \\
gateH[66] + strandB[66] &\xrightarrow{\mathrm{q_{max}}} X170 + gateL[66] \\
W171 + gateH[66] &\xrightarrow{\mathrm{q_{max}}} strandO[66] \\
strandO[66] + gateT[66] &\xrightarrow{\mathrm{q_{max}}} X170 + W171 + C0 \\
\\
X171 + gateL[67] &\xrightarrow{\mathrm{k}} gateH[67] + strandB[67] \\
gateH[67] + strandB[67] &\xrightarrow{\mathrm{q_{max}}} X171 + gateL[67] \\
W170 + gateH[67] &\xrightarrow{\mathrm{q_{max}}} strandO[67] \\
strandO[67] + gateT[67] &\xrightarrow{\mathrm{q_{max}}} X171 + W170 + C0 \\
\\
X171 + gateL[68] &\xrightarrow{\mathrm{k}} gateH[68] + strandB[68] \\
gateH[68] + strandB[68] &\xrightarrow{\mathrm{q_{max}}} X171 + gateL[68] \\
W171 + gateH[68] &\xrightarrow{\mathrm{q_{max}}} strandO[68] \\
strandO[68] + gateT[68] &\xrightarrow{\mathrm{q_{max}}} X171 + W171 + C1
\end{aligned}
$$

$$
\begin{aligned}
X180 + gateL[69] &\xrightarrow{\mathrm{k}} gateH[69] + strandB[69] \\
gateH[69] + strandB[69] &\xrightarrow{\mathrm{q_{max}}} X180 + gateL[69] \\
W180 + gateH[69] &\xrightarrow{\mathrm{q_{max}}} strandO[69] \\
strandO[69] + gateT[69] &\xrightarrow{\mathrm{q_{max}}} X180 + W180 + C1 \\
\\
X180 + gateL[70] &\xrightarrow{\mathrm{k}} gateH[70] + strandB[70] \\
gateH[70] + strandB[70] &\xrightarrow{\mathrm{q_{max}}} X180 + gateL[70] \\
W181 + gateH[70] &\xrightarrow{\mathrm{q_{max}}} strandO[70] \\
strandO[70] + gateT[70] &\xrightarrow{\mathrm{q_{max}}} X180 + W181 + C0 \\
\\
X181 + gateL[71] &\xrightarrow{\mathrm{k}} gateH[71] + strandB[71] \\
gateH[71] + strandB[71] &\xrightarrow{\mathrm{q_{max}}} X181 + gateL[71] \\
W180 + gateH[71] &\xrightarrow{\mathrm{q_{max}}} strandO[71] \\
strandO[71] + gateT[71] &\xrightarrow{\mathrm{q_{max}}} X181 + W180 + C0 \\
\\
X181 + gateL[72] &\xrightarrow{\mathrm{k}} gateH[72] + strandB[72] \\
gateH[72] + strandB[72] &\xrightarrow{\mathrm{q_{max}}} X181 + gateL[72] \\
W181 + gateH[72] &\xrightarrow{\mathrm{q_{max}}} strandO[72] \\
strandO[72] + gateT[72] &\xrightarrow{\mathrm{q_{max}}} X181 + W181 + C1 \\
\\
X190 + gateL[73] &\xrightarrow{\mathrm{k}} gateH[73] + strandB[73] \\
gateH[73] + strandB[73] &\xrightarrow{\mathrm{q_{max}}} X190 + gateL[73] \\
W190 + gateH[73] &\xrightarrow{\mathrm{q_{max}}} strandO[73] \\
strandO[73] + gateT[73] &\xrightarrow{\mathrm{q_{max}}} X190 + W190 + C1 \\
\\
X190 + gateL[74] &\xrightarrow{\mathrm{k}} gateH[74] + strandB[74] \\
gateH[74] + strandB[74] &\xrightarrow{\mathrm{q_{max}}} X190 + gateL[74] \\
W191 + gateH[74] &\xrightarrow{\mathrm{q_{max}}} strandO[74] \\
strandO[74] + gateT[74] &\xrightarrow{\mathrm{q_{max}}} X190 + W191 + C0
\end{aligned}
$$

$$
\begin{aligned}
X191 + gateL[75] &\xrightarrow{\mathrm{k}} gateH[75] + strandB[75] \\
gateH[75] + strandB[75] &\xrightarrow{\mathrm{q_{max}}} X191 + gateL[75] \\
W190 + gateH[75] &\xrightarrow{\mathrm{q_{max}}} strandO[75] \\
strandO[75] + gateT[75] &\xrightarrow{\mathrm{q_{max}}} X191 + W190 + C0 \\
\\
X191 + gateL[76] &\xrightarrow{\mathrm{k}} gateH[76] + strandB[76] \\
gateH[76] + strandB[76] &\xrightarrow{\mathrm{q_{max}}} X191 + gateL[76] \\
W191 + gateH[76] &\xrightarrow{\mathrm{q_{max}}} strandO[76] \\
strandO[76] + gateT[76] &\xrightarrow{\mathrm{q_{max}}} X191 + W191 + C1 \\
\\
X200 + gateL[77] &\xrightarrow{\mathrm{k}} gateH[77] + strandB[77] \\
gateH[77] + strandB[77] &\xrightarrow{\mathrm{q_{max}}} X200 + gateL[77] \\
W200 + gateH[77] &\xrightarrow{\mathrm{q_{max}}} strandO[77] \\
strandO[77] + gateT[77] &\xrightarrow{\mathrm{q_{max}}} X200 + W200 + C1 \\
\\
X200 + gateL[78] &\xrightarrow{\mathrm{k}} gateH[78] + strandB[78] \\
gateH[78] + strandB[78] &\xrightarrow{\mathrm{q_{max}}} X200 + gateL[78] \\
W201 + gateH[78] &\xrightarrow{\mathrm{q_{max}}} strandO[78] \\
strandO[78] + gateT[78] &\xrightarrow{\mathrm{q_{max}}} X200 + W201 + C0 \\
\\
X201 + gateL[79] &\xrightarrow{\mathrm{k}} gateH[79] + strandB[79] \\
gateH[79] + strandB[79] &\xrightarrow{\mathrm{q_{max}}} X201 + gateL[79] \\
W200 + gateH[79] &\xrightarrow{\mathrm{q_{max}}} strandO[79] \\
strandO[79] + gateT[79] &\xrightarrow{\mathrm{q_{max}}} X201 + W200 + C0 \\
\\
X201 + gateL[80] &\xrightarrow{\mathrm{k}} gateH[80] + strandB[80] \\
gateH[80] + strandB[80] &\xrightarrow{\mathrm{q_{max}}} X201 + gateL[80] \\
W201 + gateH[80] &\xrightarrow{\mathrm{q_{max}}} strandO[80] \\
strandO[80] + gateT[80] &\xrightarrow{\mathrm{q_{max}}} X201 + W201 + C1
\end{aligned}
$$

$$
\begin{aligned}
X210 + gateL[81] &\xrightarrow{\text{k}} gateH[81] + strandB[81] \\
gateH[81] + strandB[81] &\xrightarrow{\text{q}_{\max}} X210 + gateL[81] \\
W210 + gateH[81] &\xrightarrow{\text{q}_{\max}} strandO[81] \\
strandO[81] + gateT[81] &\xrightarrow{\text{q}_{\max}} X210 + W210 + C1 \\
\\
X210 + gateL[82] &\xrightarrow{\text{k}} gateH[82] + strandB[82] \\
gateH[82] + strandB[82] &\xrightarrow{\text{q}_{\max}} X210 + gateL[82] \\
W211 + gateH[82] &\xrightarrow{\text{q}_{\max}} strandO[82] \\
strandO[82] + gateT[82] &\xrightarrow{\text{q}_{\max}} X210 + W211 + C0 \\
\\
X211 + gateL[83] &\xrightarrow{\text{k}} gateH[83] + strandB[83] \\
gateH[83] + strandB[83] &\xrightarrow{\text{q}_{\max}} X211 + gateL[83] \\
W210 + gateH[83] &\xrightarrow{\text{q}_{\max}} strandO[83] \\
strandO[83] + gateT[83] &\xrightarrow{\text{q}_{\max}} X211 + W210 + C0 \\
\\
X211 + gateL[84] &\xrightarrow{\text{k}} gateH[84] + strandB[84] \\
gateH[84] + strandB[84] &\xrightarrow{\text{q}_{\max}} X211 + gateL[84] \\
W211 + gateH[84] &\xrightarrow{\text{q}_{\max}} strandO[84] \\
strandO[84] + gateT[84] &\xrightarrow{\text{q}_{\max}} X211 + W211 + C1 \\
\\
X220 + gateL[85] &\xrightarrow{\text{k}} gateH[85] + strandB[85] \\
gateH[85] + strandB[85] &\xrightarrow{\text{q}_{\max}} X220 + gateL[85] \\
W220 + gateH[85] &\xrightarrow{\text{q}_{\max}} strandO[85] \\
strandO[85] + gateT[85] &\xrightarrow{\text{q}_{\max}} X220 + W220 + C1 \\
\\
X220 + gateL[86] &\xrightarrow{\text{k}} gateH[86] + strandB[86] \\
gateH[86] + strandB[86] &\xrightarrow{\text{q}_{\max}} X220 + gateL[86] \\
W221 + gateH[86] &\xrightarrow{\text{q}_{\max}} strandO[86] \\
strandO[86] + gateT[86] &\xrightarrow{\text{q}_{\max}} X220 + W221 + C0
\end{aligned}
$$

$$
\begin{aligned}
X221 + gateL[87] &\xrightarrow{\mathrm{k}} gateH[87] + strandB[87] \\
gateH[87] + strandB[87] &\xrightarrow{\mathrm{q_{max}}} X221 + gateL[87] \\
W220 + gateH[87] &\xrightarrow{\mathrm{q_{max}}} strandO[87] \\
strandO[87] + gateT[87] &\xrightarrow{\mathrm{q_{max}}} X221 + W220 + C0 \\
\\
X221 + gateL[88] &\xrightarrow{\mathrm{k}} gateH[88] + strandB[88] \\
gateH[88] + strandB[88] &\xrightarrow{\mathrm{q_{max}}} X221 + gateL[88] \\
W221 + gateH[88] &\xrightarrow{\mathrm{q_{max}}} strandO[88] \\
strandO[88] + gateT[88] &\xrightarrow{\mathrm{q_{max}}} X221 + W221 + C1 \\
\\
X230 + gateL[89] &\xrightarrow{\mathrm{k}} gateH[89] + strandB[89] \\
gateH[89] + strandB[89] &\xrightarrow{\mathrm{q_{max}}} X230 + gateL[89] \\
W230 + gateH[89] &\xrightarrow{\mathrm{q_{max}}} strandO[89] \\
strandO[89] + gateT[89] &\xrightarrow{\mathrm{q_{max}}} X230 + W230 + C1 \\
\\
X230 + gateL[90] &\xrightarrow{\mathrm{k}} gateH[90] + strandB[90] \\
gateH[90] + strandB[90] &\xrightarrow{\mathrm{q_{max}}} X230 + gateL[90] \\
W231 + gateH[90] &\xrightarrow{\mathrm{q_{max}}} strandO[90] \\
strandO[90] + gateT[90] &\xrightarrow{\mathrm{q_{max}}} X230 + W231 + C0 \\
\\
X231 + gateL[91] &\xrightarrow{\mathrm{k}} gateH[91] + strandB[91] \\
gateH[91] + strandB[91] &\xrightarrow{\mathrm{q_{max}}} X231 + gateL[91] \\
W230 + gateH[91] &\xrightarrow{\mathrm{q_{max}}} strandO[91] \\
strandO[91] + gateT[91] &\xrightarrow{\mathrm{q_{max}}} X231 + W230 + C0 \\
\\
X231 + gateL[92] &\xrightarrow{\mathrm{k}} gateH[92] + strandB[92] \\
gateH[92] + strandB[92] &\xrightarrow{\mathrm{q_{max}}} X231 + gateL[92] \\
W231 + gateH[92] &\xrightarrow{\mathrm{q_{max}}} strandO[92] \\
strandO[92] + gateT[92] &\xrightarrow{\mathrm{q_{max}}} X231 + W231 + C1
\end{aligned}
$$

$$
\begin{aligned}
X240 + gateL[93] &\xrightarrow{\text{k}} gateH[93] + strandB[93] \\
gateH[93] + strandB[93] &\xrightarrow{\text{q}_{\max}} X240 + gateL[93] \\
W240 + gateH[93] &\xrightarrow{\text{q}_{\max}} strandO[93] \\
strandO[93] + gateT[93] &\xrightarrow{\text{q}_{\max}} X240 + W240 + C1 \\
\\
X240 + gateL[94] &\xrightarrow{\text{k}} gateH[94] + strandB[94] \\
gateH[94] + strandB[94] &\xrightarrow{\text{q}_{\max}} X240 + gateL[94] \\
W241 + gateH[94] &\xrightarrow{\text{q}_{\max}} strandO[94] \\
strandO[94] + gateT[94] &\xrightarrow{\text{q}_{\max}} X240 + W241 + C0 \\
\\
X241 + gateL[95] &\xrightarrow{\text{k}} gateH[95] + strandB[95] \\
gateH[95] + strandB[95] &\xrightarrow{\text{q}_{\max}} X241 + gateL[95] \\
W240 + gateH[95] &\xrightarrow{\text{q}_{\max}} strandO[95] \\
strandO[95] + gateT[95] &\xrightarrow{\text{q}_{\max}} X241 + W240 + C0 \\
\\
X241 + gateL[96] &\xrightarrow{\text{k}} gateH[96] + strandB[96] \\
gateH[96] + strandB[96] &\xrightarrow{\text{q}_{\max}} X241 + gateL[96] \\
W241 + gateH[96] &\xrightarrow{\text{q}_{\max}} strandO[96] \\
strandO[96] + gateT[96] &\xrightarrow{\text{q}_{\max}} X241 + W241 + C1 \\
\\
X250 + gateL[97] &\xrightarrow{\text{k}} gateH[97] + strandB[97] \\
gateH[97] + strandB[97] &\xrightarrow{\text{q}_{\max}} X250 + gateL[97] \\
W250 + gateH[97] &\xrightarrow{\text{q}_{\max}} strandO[97] \\
strandO[97] + gateT[97] &\xrightarrow{\text{q}_{\max}} X250 + W250 + C1 \\
\\
X250 + gateL[98] &\xrightarrow{\text{k}} gateH[98] + strandB[98] \\
gateH[98] + strandB[98] &\xrightarrow{\text{q}_{\max}} X250 + gateL[98] \\
W251 + gateH[98] &\xrightarrow{\text{q}_{\max}} strandO[98] \\
strandO[98] + gateT[98] &\xrightarrow{\text{q}_{\max}} X250 + W251 + C0
\end{aligned}
$$

$$
\begin{aligned}
X251 + gateL[99] &\xrightarrow{\text{k}} gateH[99] + strandB[99] \\
gateH[99] + strandB[99] &\xrightarrow{\text{q}_{\max}} X251 + gateL[99] \\
W250 + gateH[99] &\xrightarrow{\text{q}_{\max}} strandO[99] \\
strandO[99] + gateT[99] &\xrightarrow{\text{q}_{\max}} X251 + W250 + C0 \\
\\
X251 + gateL[100] &\xrightarrow{\text{k}} gateH[100] + strandB[100] \\
gateH[100] + strandB[100] &\xrightarrow{\text{q}_{\max}} X251 + gateL[100] \\
W251 + gateH[100] &\xrightarrow{\text{q}_{\max}} strandO[100] \\
strandO[100] + gateT[100] &\xrightarrow{\text{q}_{\max}} X251 + W251 + C1 \\
\\
X260 + gateL[101] &\xrightarrow{\text{k}} gateH[101] + strandB[101] \\
gateH[101] + strandB[101] &\xrightarrow{\text{q}_{\max}} X260 + gateL[101] \\
W260 + gateH[101] &\xrightarrow{\text{q}_{\max}} strandO[101] \\
strandO[101] + gateT[101] &\xrightarrow{\text{q}_{\max}} X260 + W260 + C1 \\
\\
X260 + gateL[102] &\xrightarrow{\text{k}} gateH[102] + strandB[102] \\
gateH[102] + strandB[102] &\xrightarrow{\text{q}_{\max}} X260 + gateL[102] \\
W261 + gateH[102] &\xrightarrow{\text{q}_{\max}} strandO[102] \\
strandO[102] + gateT[102] &\xrightarrow{\text{q}_{\max}} X260 + W261 + C0 \\
\\
X261 + gateL[103] &\xrightarrow{\text{k}} gateH[103] + strandB[103] \\
gateH[103] + strandB[103] &\xrightarrow{\text{q}_{\max}} X261 + gateL[103] \\
W260 + gateH[103] &\xrightarrow{\text{q}_{\max}} strandO[103] \\
strandO[103] + gateT[103] &\xrightarrow{\text{q}_{\max}} X261 + W260 + C0 \\
\\
X261 + gateL[104] &\xrightarrow{\text{k}} gateH[104] + strandB[104] \\
gateH[104] + strandB[104] &\xrightarrow{\text{q}_{\max}} X261 + gateL[104] \\
W261 + gateH[104] &\xrightarrow{\text{q}_{\max}} strandO[104] \\
strandO[104] + gateT[104] &\xrightarrow{\text{q}_{\max}} X261 + W261 + C1
\end{aligned}
$$

$$
\begin{aligned}
X270 + gateL[105] &\xrightarrow{\text{k}} gateH[105] + strandB[105] \\
gateH[105] + strandB[105] &\xrightarrow{\text{q}_{\max}} X270 + gateL[105] \\
W270 + gateH[105] &\xrightarrow{\text{q}_{\max}} strandO[105] \\
strandO[105] + gateT[105] &\xrightarrow{\text{q}_{\max}} X270 + W270 + C1 \\
\\
X270 + gateL[106] &\xrightarrow{\text{k}} gateH[106] + strandB[106] \\
gateH[106] + strandB[106] &\xrightarrow{\text{q}_{\max}} X270 + gateL[106] \\
W271 + gateH[106] &\xrightarrow{\text{q}_{\max}} strandO[106] \\
strandO[106] + gateT[106] &\xrightarrow{\text{q}_{\max}} X270 + W271 + C0 \\
\\
X271 + gateL[107] &\xrightarrow{\text{k}} gateH[107] + strandB[107] \\
gateH[107] + strandB[107] &\xrightarrow{\text{q}_{\max}} X271 + gateL[107] \\
W270 + gateH[107] &\xrightarrow{\text{q}_{\max}} strandO[107] \\
strandO[107] + gateT[107] &\xrightarrow{\text{q}_{\max}} X271 + W270 + C0 \\
\\
X271 + gateL[108] &\xrightarrow{\text{k}} gateH[108] + strandB[108] \\
gateH[108] + strandB[108] &\xrightarrow{\text{q}_{\max}} X271 + gateL[108] \\
W271 + gateH[108] &\xrightarrow{\text{q}_{\max}} strandO[108] \\
strandO[108] + gateT[108] &\xrightarrow{\text{q}_{\max}} X271 + W271 + C1 \\
\\
X280 + gateL[109] &\xrightarrow{\text{k}} gateH[109] + strandB[109] \\
gateH[109] + strandB[109] &\xrightarrow{\text{q}_{\max}} X280 + gateL[109] \\
W280 + gateH[109] &\xrightarrow{\text{q}_{\max}} strandO[109] \\
strandO[109] + gateT[109] &\xrightarrow{\text{q}_{\max}} X280 + W280 + C1 \\
\\
X280 + gateL[110] &\xrightarrow{\text{k}} gateH[110] + strandB[110] \\
gateH[110] + strandB[110] &\xrightarrow{\text{q}_{\max}} X280 + gateL[110] \\
W281 + gateH[110] &\xrightarrow{\text{q}_{\max}} strandO[110] \\
strandO[110] + gateT[110] &\xrightarrow{\text{q}_{\max}} X280 + W281 + C0
\end{aligned}
$$

$$
\begin{aligned}
X281 + gateL[111] &\xrightarrow{\text{k}} gateH[111] + strandB[111] \\
gateH[111] + strandB[111] &\xrightarrow{\text{q}_{\max}} X281 + gateL[111] \\
W280 + gateH[111] &\xrightarrow{\text{q}_{\max}} strandO[111] \\
strandO[111] + gateT[111] &\xrightarrow{\text{q}_{\max}} X281 + W280 + C0 \\
\\
X281 + gateL[112] &\xrightarrow{\text{k}} gateH[112] + strandB[112] \\
gateH[112] + strandB[112] &\xrightarrow{\text{q}_{\max}} X281 + gateL[112] \\
W281 + gateH[112] &\xrightarrow{\text{q}_{\max}} strandO[112] \\
strandO[112] + gateT[112] &\xrightarrow{\text{q}_{\max}} X281 + W281 + C1 \\
\\
X290 + gateL[113] &\xrightarrow{\text{k}} gateH[113] + strandB[113] \\
gateH[113] + strandB[113] &\xrightarrow{\text{q}_{\max}} X290 + gateL[113] \\
W290 + gateH[113] &\xrightarrow{\text{q}_{\max}} strandO[113] \\
strandO[113] + gateT[113] &\xrightarrow{\text{q}_{\max}} X290 + W290 + C1 \\
\\
X290 + gateL[114] &\xrightarrow{\text{k}} gateH[114] + strandB[114] \\
gateH[114] + strandB[114] &\xrightarrow{\text{q}_{\max}} X290 + gateL[114] \\
W291 + gateH[114] &\xrightarrow{\text{q}_{\max}} strandO[114] \\
strandO[114] + gateT[114] &\xrightarrow{\text{q}_{\max}} X290 + W291 + C0 \\
\\
X291 + gateL[115] &\xrightarrow{\text{k}} gateH[115] + strandB[115] \\
gateH[115] + strandB[115] &\xrightarrow{\text{q}_{\max}} X291 + gateL[115] \\
W290 + gateH[115] &\xrightarrow{\text{q}_{\max}} strandO[115] \\
strandO[115] + gateT[115] &\xrightarrow{\text{q}_{\max}} X291 + W290 + C0 \\
\\
X291 + gateL[116] &\xrightarrow{\text{k}} gateH[116] + strandB[116] \\
gateH[116] + strandB[116] &\xrightarrow{\text{q}_{\max}} X291 + gateL[116] \\
W291 + gateH[116] &\xrightarrow{\text{q}_{\max}} strandO[116] \\
strandO[116] + gateT[116] &\xrightarrow{\text{q}_{\max}} X291 + W291 + C1
\end{aligned}
$$

$$
\begin{aligned}
X300 + gateL[117] &\xrightarrow{\text{k}} gateH[117] + strandB[117] \\
gateH[117] + strandB[117] &\xrightarrow{\text{q}_{\max}} X300 + gateL[117] \\
W300 + gateH[117] &\xrightarrow{\text{q}_{\max}} strandO[117] \\
strandO[117] + gateT[117] &\xrightarrow{\text{q}_{\max}} X300 + W300 + C1 \\
\\
X300 + gateL[118] &\xrightarrow{\text{k}} gateH[118] + strandB[118] \\
gateH[118] + strandB[118] &\xrightarrow{\text{q}_{\max}} X300 + gateL[118] \\
W301 + gateH[118] &\xrightarrow{\text{q}_{\max}} strandO[118] \\
strandO[118] + gateT[118] &\xrightarrow{\text{q}_{\max}} X300 + W301 + C0 \\
\\
X301 + gateL[119] &\xrightarrow{\text{k}} gateH[119] + strandB[119] \\
gateH[119] + strandB[119] &\xrightarrow{\text{q}_{\max}} X301 + gateL[119] \\
W300 + gateH[119] &\xrightarrow{\text{q}_{\max}} strandO[119] \\
strandO[119] + gateT[119] &\xrightarrow{\text{q}_{\max}} X301 + W300 + C0 \\
\\
X301 + gateL[120] &\xrightarrow{\text{k}} gateH[120] + strandB[120] \\
gateH[120] + strandB[120] &\xrightarrow{\text{q}_{\max}} X301 + gateL[120] \\
W301 + gateH[120] &\xrightarrow{\text{q}_{\max}} strandO[120] \\
strandO[120] + gateT[120] &\xrightarrow{\text{q}_{\max}} X301 + W301 + C1 \\
\\
X310 + gateL[121] &\xrightarrow{\text{k}} gateH[121] + strandB[121] \\
gateH[121] + strandB[121] &\xrightarrow{\text{q}_{\max}} X310 + gateL[121] \\
W310 + gateH[121] &\xrightarrow{\text{q}_{\max}} strandO[121] \\
strandO[121] + gateT[121] &\xrightarrow{\text{q}_{\max}} X310 + W310 + C1 \\
\\
X310 + gateL[122] &\xrightarrow{\text{k}} gateH[122] + strandB[122] \\
gateH[122] + strandB[122] &\xrightarrow{\text{q}_{\max}} X310 + gateL[122] \\
W311 + gateH[122] &\xrightarrow{\text{q}_{\max}} strandO[122] \\
strandO[122] + gateT[122] &\xrightarrow{\text{q}_{\max}} X310 + W311 + C0
\end{aligned}
$$

$$
\begin{aligned}
X311 + gateL[123] &\xrightarrow{\mathrm{k}} gateH[123] + strandB[123] \\
gateH[123] + strandB[123] &\xrightarrow{\mathrm{q_{max}}} X311 + gateL[123] \\
W310 + gateH[123] &\xrightarrow{\mathrm{q_{max}}} strandO[123] \\
strandO[123] + gateT[123] &\xrightarrow{\mathrm{q_{max}}} X311 + W310 + C0 \\
\\
X311 + gateL[124] &\xrightarrow{\mathrm{k}} gateH[124] + strandB[124] \\
gateH[124] + strandB[124] &\xrightarrow{\mathrm{q_{max}}} X311 + gateL[124] \\
W311 + gateH[124] &\xrightarrow{\mathrm{q_{max}}} strandO[124] \\
strandO[124] + gateT[124] &\xrightarrow{\mathrm{q_{max}}} X311 + W311 + C1 \\
\\
X320 + gateL[125] &\xrightarrow{\mathrm{k}} gateH[125] + strandB[125] \\
gateH[125] + strandB[125] &\xrightarrow{\mathrm{q_{max}}} X320 + gateL[125] \\
W320 + gateH[125] &\xrightarrow{\mathrm{q_{max}}} strandO[125] \\
strandO[125] + gateT[125] &\xrightarrow{\mathrm{q_{max}}} X320 + W320 + C1 \\
\\
X320 + gateL[126] &\xrightarrow{\mathrm{k}} gateH[126] + strandB[126] \\
gateH[126] + strandB[126] &\xrightarrow{\mathrm{q_{max}}} X320 + gateL[126] \\
W321 + gateH[126] &\xrightarrow{\mathrm{q_{max}}} strandO[126] \\
strandO[126] + gateT[126] &\xrightarrow{\mathrm{q_{max}}} X320 + W321 + C0 \\
\\
X321 + gateL[127] &\xrightarrow{\mathrm{k}} gateH[127] + strandB[127] \\
gateH[127] + strandB[127] &\xrightarrow{\mathrm{q_{max}}} X321 + gateL[127] \\
W320 + gateH[127] &\xrightarrow{\mathrm{q_{max}}} strandO[127] \\
strandO[127] + gateT[127] &\xrightarrow{\mathrm{q_{max}}} X321 + W320 + C0 \\
\\
X321 + gateL[128] &\xrightarrow{\mathrm{k}} gateH[128] + strandB[128] \\
gateH[128] + strandB[128] &\xrightarrow{\mathrm{q_{max}}} X321 + gateL[128] \\
W321 + gateH[128] &\xrightarrow{\mathrm{q_{max}}} strandO[128] \\
strandO[128] + gateT[128] &\xrightarrow{\mathrm{q_{max}}} X321 + W321 + C1
\end{aligned}
$$

$$
\begin{aligned}
X330 + gateL[129] &\xrightarrow{\mathrm{k}} gateH[129] + strandB[129] \\
gateH[129] + strandB[129] &\xrightarrow{\mathrm{q_{max}}} X330 + gateL[129] \\
W330 + gateH[129] &\xrightarrow{\mathrm{q_{max}}} strandO[129] \\
strandO[129] + gateT[129] &\xrightarrow{\mathrm{q_{max}}} X330 + W330 + C1
\end{aligned}
$$

$$
\begin{aligned}
X330 + gateL[130] &\xrightarrow{\mathrm{k}} gateH[130] + strandB[130] \\
gateH[130] + strandB[130] &\xrightarrow{\mathrm{q_{max}}} X330 + gateL[130] \\
W331 + gateH[130] &\xrightarrow{\mathrm{q_{max}}} strandO[130] \\
strandO[130] + gateT[130] &\xrightarrow{\mathrm{q_{max}}} X330 + W331 + C0
\end{aligned}
$$

$$
\begin{aligned}
X331 + gateL[131] &\xrightarrow{\mathrm{k}} gateH[131] + strandB[131] \\
gateH[131] + strandB[131] &\xrightarrow{\mathrm{q_{max}}} X331 + gateL[131] \\
W330 + gateH[131] &\xrightarrow{\mathrm{q_{max}}} strandO[131] \\
strandO[131] + gateT[131] &\xrightarrow{\mathrm{q_{max}}} X331 + W330 + C0
\end{aligned}
$$

$$
\begin{aligned}
X331 + gateL[132] &\xrightarrow{\mathrm{k}} gateH[132] + strandB[132] \\
gateH[132] + strandB[132] &\xrightarrow{\mathrm{q_{max}}} X331 + gateL[132] \\
W331 + gateH[132] &\xrightarrow{\mathrm{q_{max}}} strandO[132] \\
strandO[132] + gateT[132] &\xrightarrow{\mathrm{q_{max}}} X331 + W331 + C1
\end{aligned}
$$

$$
\begin{aligned}
C0 + gateG[133] &\xrightarrow{\mathrm{k_2}} strandO[133] \\
strandO[133] + gateT[133] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
C1 + gateG[134] &\xrightarrow{\mathrm{k_2}} strandO[134] \\
strandO[134] + gateT[134] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
C0 + gateL[135] &\xrightarrow{\mathrm{k}} gateH[135] + strandB[135] \\
gateH[135] + strandB[135] &\xrightarrow{\mathrm{q_{max}}} C0 + gateL[135] \\
C0 + gateH[135] &\xrightarrow{\mathrm{q_{max}}} strandO[135] \\
strandO[135] + gateT[135] &\xrightarrow{\mathrm{q_{max}}} C11 + C0 + C0
\end{aligned}
$$

$$
\begin{aligned}
C0 + gateL[136] &\xrightarrow{\mathrm{k}} gateH[136] + strandB[136] \\
gateH[136] + strandB[136] &\xrightarrow{\mathrm{q_{max}}} C0 + gateL[136] \\
C1 + gateH[136] &\xrightarrow{\mathrm{q_{max}}} strandO[136] \\
strandO[136] + gateT[136] &\xrightarrow{\mathrm{q_{max}}} C10 + C0 + C1
\end{aligned}
$$

$$
\begin{aligned}
C1 + gateL[137] &\xrightarrow{\mathrm{k}} gateH[137] + strandB[137] \\
gateH[137] + strandB[137] &\xrightarrow{\mathrm{q_{max}}} C1 + gateL[137] \\
C0 + gateH[137] &\xrightarrow{\mathrm{q_{max}}} strandO[137] \\
strandO[137] + gateT[137] &\xrightarrow{\mathrm{q_{max}}} C10 + C1 + C0
\end{aligned}
$$

$$
\begin{aligned}
C1 + gateL[138] &\xrightarrow{\mathrm{k}} gateH[138] + strandB[138] \\
gateH[138] + strandB[138] &\xrightarrow{\mathrm{q_{max}}} C1 + gateL[138] \\
C1 + gateH[138] &\xrightarrow{\mathrm{q_{max}}} strandO[138] \\
strandO[138] + gateT[138] &\xrightarrow{\mathrm{q_{max}}} C11 + C1 + C1
\end{aligned}
$$

$$
\begin{aligned}
C10 + gateG[139] &\xrightarrow{\mathrm{k_2}} strandO[139] \\
strandO[139] + gateT[139] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
C11 + gateG[140] &\xrightarrow{\mathrm{k_2}} strandO[140] \\
strandO[140] + gateT[140] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
A20 + gateL[141] &\xrightarrow{\mathrm{k}} gateH[141] + strandB[141] \\
gateH[141] + strandB[141] &\xrightarrow{\mathrm{q_{max}}} A20 + gateL[141] \\
C10 + gateH[141] &\xrightarrow{\mathrm{q_{max}}} strandO[141] \\
strandO[141] + gateT[141] &\xrightarrow{\mathrm{q_{max}}} C20 + A20 + C10
\end{aligned}
$$

$$
\begin{aligned}
A20 + gateL[142] &\xrightarrow{\mathrm{k}} gateH[142] + strandB[142] \\
gateH[142] + strandB[142] &\xrightarrow{\mathrm{q_{max}}} A20 + gateL[142] \\
C11 + gateH[142] &\xrightarrow{\mathrm{q_{max}}} strandO[142] \\
strandO[142] + gateT[142] &\xrightarrow{\mathrm{q_{max}}} C21 + A20 + C11
\end{aligned}
$$

$$
\begin{aligned}
A21 + gateL[143] &\xrightarrow{\mathrm{k}} gateH[143] + strandB[143] \\
gateH[143] + strandB[143] &\xrightarrow{\mathrm{q_{max}}} A21 + gateL[143] \\
C10 + gateH[143] &\xrightarrow{\mathrm{q_{max}}} strandO[143] \\
strandO[143] + gateT[143] &\xrightarrow{\mathrm{q_{max}}} C21 + A21 + C10
\end{aligned}
$$

$$
\begin{aligned}
A21 + gateL[144] &\xrightarrow{\mathrm{k}} gateH[144] + strandB[144] \\
gateH[144] + strandB[144] &\xrightarrow{\mathrm{q_{max}}} A21 + gateL[144] \\
C11 + gateH[144] &\xrightarrow{\mathrm{q_{max}}} strandO[144] \\
strandO[144] + gateT[144] &\xrightarrow{\mathrm{q_{max}}} C20 + A21 + C11
\end{aligned}
$$

$$
\begin{aligned}
C20 + gateG[145] &\xrightarrow{\mathrm{k_2}} strandO[145] \\
strandO[145] + gateT[145] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
C21 + gateG[146] &\xrightarrow{\mathrm{k_2}} strandO[146] \\
strandO[146] + gateT[146] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
A30 + gateL[147] &\xrightarrow{\mathrm{k}} gateH[147] + strandB[147] \\
gateH[147] + strandB[147] &\xrightarrow{\mathrm{q_{max}}} A30 + gateL[147] \\
C20 + gateH[147] &\xrightarrow{\mathrm{q_{max}}} strandO[147] \\
strandO[147] + gateT[147] &\xrightarrow{\mathrm{q_{max}}} C30 + A30 + C20
\end{aligned}
$$

$$
\begin{aligned}
A30 + gateL[148] &\xrightarrow{\mathrm{k}} gateH[148] + strandB[148] \\
gateH[148] + strandB[148] &\xrightarrow{\mathrm{q_{max}}} A30 + gateL[148] \\
C21 + gateH[148] &\xrightarrow{\mathrm{q_{max}}} strandO[148] \\
strandO[148] + gateT[148] &\xrightarrow{\mathrm{q_{max}}} C31 + A30 + C21
\end{aligned}
$$

$$
\begin{aligned}
A31 + gateL[149] &\xrightarrow{\mathrm{k}} gateH[149] + strandB[149] \\
gateH[149] + strandB[149] &\xrightarrow{\mathrm{q_{max}}} A31 + gateL[149] \\
B30 + gateH[149] &\xrightarrow{\mathrm{q_{max}}} strandO[149] \\
strandO[149] + gateT[149] &\xrightarrow{\mathrm{q_{max}}} C30 + A31 + B30
\end{aligned}
$$

$$
\begin{aligned}
A31 + gateL[150] &\xrightarrow{\mathrm{k}} gateH[150] + strandB[150] \\
gateH[150] + strandB[150] &\xrightarrow{\mathrm{q_{max}}} A31 + gateL[150] \\
B31 + gateH[150] &\xrightarrow{\mathrm{q_{max}}} strandO[150] \\
strandO[150] + gateT[150] &\xrightarrow{\mathrm{q_{max}}} C31 + A31 + B31 \\
\\
C30 + gateG[151] &\xrightarrow{\mathrm{k_2}} strandO[151] \\
strandO[151] + gateT[151] &\xrightarrow{\mathrm{q_{max}}} \varnothing \\
\\
C31 + gateG[152] &\xrightarrow{\mathrm{k_2}} strandO[152] \\
strandO[152] + gateT[152] &\xrightarrow{\mathrm{q_{max}}} \varnothing \\
\\
C10 + gateL[153] &\xrightarrow{\mathrm{k}} gateH[153] + strandB[153] \\
gateH[153] + strandB[153] &\xrightarrow{\mathrm{q_{max}}} C10 + gateL[153] \\
C30 + gateH[153] &\xrightarrow{\mathrm{q_{max}}} strandO[153] \\
strandO[153] + gateT[153] &\xrightarrow{\mathrm{q_{max}}} C41 + C10 + C30 \\
\\
C10 + gateL[154] &\xrightarrow{\mathrm{k}} gateH[154] + strandB[154] \\
gateH[154] + strandB[154] &\xrightarrow{\mathrm{q_{max}}} C10 + gateL[154] \\
C31 + gateH[154] &\xrightarrow{\mathrm{q_{max}}} strandO[154] \\
strandO[154] + gateT[154] &\xrightarrow{\mathrm{q_{max}}} C40 + C10 + C31 \\
\\
C11 + gateL[155] &\xrightarrow{\mathrm{k}} gateH[155] + strandB[155] \\
gateH[155] + strandB[155] &\xrightarrow{\mathrm{q_{max}}} C11 + gateL[155] \\
C30 + gateH[155] &\xrightarrow{\mathrm{q_{max}}} strandO[155] \\
strandO[155] + gateT[155] &\xrightarrow{\mathrm{q_{max}}} C40 + C11 + C30 \\
\\
C11 + gateL[156] &\xrightarrow{\mathrm{k}} gateH[156] + strandB[156] \\
gateH[156] + strandB[156] &\xrightarrow{\mathrm{q_{max}}} C11 + gateL[156] \\
C31 + gateH[156] &\xrightarrow{\mathrm{q_{max}}} strandO[156] \\
strandO[156] + gateT[156] &\xrightarrow{\mathrm{q_{max}}} C41 + C11 + C31 \\
\\
C40 + gateG[157] &\xrightarrow{\mathrm{k_2}} strandO[157] \\
strandO[157] + gateT[157] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
C41 + gateG[158] &\xrightarrow{k_2} strandO[158] \\
strandO[158] + gateT[158] &\xrightarrow{q_{max}} \varnothing \\
\\
A50 + gateL[159] &\xrightarrow{k} gateH[159] + strandB[159] \\
gateH[159] + strandB[159] &\xrightarrow{q_{max}} A50 + gateL[159] \\
C40 + gateH[159] &\xrightarrow{q_{max}} strandO[159] \\
strandO[159] + gateT[159] &\xrightarrow{q_{max}} C50 + A50 + C40 \\
\\
A50 + gateL[160] &\xrightarrow{k} gateH[160] + strandB[160] \\
gateH[160] + strandB[160] &\xrightarrow{q_{max}} A50 + gateL[160] \\
C41 + gateH[160] &\xrightarrow{q_{max}} strandO[160] \\
strandO[160] + gateT[160] &\xrightarrow{q_{max}} C51 + A50 + C41 \\
\\
A51 + gateL[161] &\xrightarrow{k} gateH[161] + strandB[161] \\
gateH[161] + strandB[161] &\xrightarrow{q_{max}} A51 + gateL[161] \\
B50 + gateH[161] &\xrightarrow{q_{max}} strandO[161] \\
strandO[161] + gateT[161] &\xrightarrow{q_{max}} C50 + A51 + B50 \\
\\
A51 + gateL[162] &\xrightarrow{k} gateH[162] + strandB[162] \\
gateH[162] + strandB[162] &\xrightarrow{q_{max}} A51 + gateL[162] \\
B51 + gateH[162] &\xrightarrow{q_{max}} strandO[162] \\
strandO[162] + gateT[162] &\xrightarrow{q_{max}} C51 + A51 + B51 \\
\\
C50 + gateG[163] &\xrightarrow{k_2} strandO[163] \\
strandO[163] + gateT[163] &\xrightarrow{q_{max}} \varnothing \\
\\
C51 + gateG[164] &\xrightarrow{k_2} strandO[164] \\
strandO[164] + gateT[164] &\xrightarrow{q_{max}} \varnothing \\
\\
C0 + gateL[165] &\xrightarrow{k} gateH[165] + strandB[165] \\
gateH[165] + strandB[165] &\xrightarrow{q_{max}} C0 + gateL[165] \\
C50 + gateH[165] &\xrightarrow{q_{max}} strandO[165] \\
strandO[165] + gateT[165] &\xrightarrow{q_{max}} C60 + C0 + C50
\end{aligned}
$$

$$
\begin{aligned}
C0 + gateL[166] &\xrightarrow{\mathrm{k}} gateH[166] + strandB[166] \\
gateH[166] + strandB[166] &\xrightarrow{\mathrm{q_{max}}} C0 + gateL[166] \\
C51 + gateH[166] &\xrightarrow{\mathrm{q_{max}}} strandO[166] \\
strandO[166] + gateT[166] &\xrightarrow{\mathrm{q_{max}}} C61 + C0 + C51
\end{aligned}
$$

$$
\begin{aligned}
C1 + gateL[167] &\xrightarrow{\mathrm{k}} gateH[167] + strandB[167] \\
gateH[167] + strandB[167] &\xrightarrow{\mathrm{q_{max}}} C1 + gateL[167] \\
C50 + gateH[167] &\xrightarrow{\mathrm{q_{max}}} strandO[167] \\
strandO[167] + gateT[167] &\xrightarrow{\mathrm{q_{max}}} C61 + C1 + C50
\end{aligned}
$$

$$
\begin{aligned}
C1 + gateL[168] &\xrightarrow{\mathrm{k}} gateH[168] + strandB[168] \\
gateH[168] + strandB[168] &\xrightarrow{\mathrm{q_{max}}} C1 + gateL[168] \\
C51 + gateH[168] &\xrightarrow{\mathrm{q_{max}}} strandO[168] \\
strandO[168] + gateT[168] &\xrightarrow{\mathrm{q_{max}}} C60 + C1 + C51
\end{aligned}
$$

$$
\begin{aligned}
C60 + gateG[169] &\xrightarrow{\mathrm{k_2}} strandO[169] \\
strandO[169] + gateT[169] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
C61 + gateG[170] &\xrightarrow{\mathrm{k_2}} strandO[170] \\
strandO[170] + gateT[170] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
C60 + gateG[171] &\xrightarrow{\mathrm{k_2}} strandO[171] \\
strandO[171] + gateT[171] &\xrightarrow{\mathrm{q_{max}}} C60 + cp
\end{aligned}
$$

$$
\begin{aligned}
C61 + gateG[172] &\xrightarrow{\mathrm{k_2}} strandO[172] \\
strandO[172] + gateT[172] &\xrightarrow{\mathrm{q_{max}}} C61 + cp
\end{aligned}
$$

$$
\begin{aligned}
cp + gateG[173] &\xrightarrow{\mathrm{k_2}} strandO[173] \\
strandO[173] + gateT[173] &\xrightarrow{\mathrm{q_{max}}} \varnothing
\end{aligned}
$$

$$
\begin{aligned}
C61 + gateG[174] &\xrightarrow{\mathrm{k_2}} strandO[174] \\
strandO[174] + gateT[174] &\xrightarrow{\mathrm{q_{max}}} C61 + c
\end{aligned}
$$

$$
\begin{aligned}
cp + gateL[175] &\xrightarrow{\mathrm{k}} gateH[175] + strandB[175] \\
gateH[175] + strandB[175] &\xrightarrow{\mathrm{q_{max}}} cp + gateL[175] \\
c + gateH[175] &\xrightarrow{\mathrm{q_{max}}} strandO[175] \\
strandO[175] + gateT[175] &\xrightarrow{\mathrm{q_{max}}} cp, \\
\\
A20 + gateLS[A20] &\xrightarrow{\mathrm{k_1}} gateHS[A20] + strandBS[A20] \\
gateHS[A20] + strandBS[A20] &\xrightarrow{\mathrm{q_{max}}} A20 + gateLS[A20] \\
\\
A21 + gateLS[A21] &\xrightarrow{\mathrm{k_1}} gateHS[A21] + strandBS[A21] \\
gateHS[A21] + strandBS[A21] &\xrightarrow{\mathrm{q_{max}}} A21 + gateLS[A21] \\
\\
A30 + gateLS[A30] &\xrightarrow{\mathrm{k_1}} gateHS[A30] + strandBS[A30] \\
gateHS[A30] + strandBS[A30] &\xrightarrow{\mathrm{q_{max}}} A30 + gateLS[A30] \\
\\
A31 + gateLS[A31] &\xrightarrow{\mathrm{k_1}} gateHS[A31] + strandBS[A31] \\
gateHS[A31] + strandBS[A31] &\xrightarrow{\mathrm{q_{max}}} A31 + gateLS[A31] \\
\\
A50 + gateLS[A50] &\xrightarrow{\mathrm{k_1}} gateHS[A50] + strandBS[A50] \\
gateHS[A50] + strandBS[A50] &\xrightarrow{\mathrm{q_{max}}} A50 + gateLS[A50] \\
\\
A51 + gateLS[A51] &\xrightarrow{\mathrm{k_1}} gateHS[A51] + strandBS[A51] \\
gateHS[A51] + strandBS[A51] &\xrightarrow{\mathrm{q_{max}}} A51 + gateLS[A51] \\
\\
B30 + gateLS[B30] &\xrightarrow{\mathrm{q_{max}}} gateHS[B30] + strandBS[B30] \\
gateHS[B30] + strandBS[B30] &\xrightarrow{\mathrm{q_{max}}} B30 + gateLS[B30] \\
\\
B31 + gateLS[B31] &\xrightarrow{\mathrm{q_{max}}} gateHS[B31] + strandBS[B31] \\
gateHS[B31] + strandBS[B31] &\xrightarrow{\mathrm{q_{max}}} B31 + gateLS[B31] \\
\\
B50 + gateLS[B50] &\xrightarrow{\mathrm{q_{max}}} gateHS[B50] + strandBS[B50] \\
gateHS[B50] + strandBS[B50] &\xrightarrow{\mathrm{q_{max}}} B50 + gateLS[B50] \\
\\
B51 + gateLS[B51] &\xrightarrow{\mathrm{q_{max}}} gateHS[B51] + strandBS[B51] \\
gateHS[B51] + strandBS[B51] &\xrightarrow{\mathrm{q_{max}}} B51 + gateLS[B51]
\end{aligned}
$$

$$
\begin{aligned}
c + gateLS[c] &\xrightarrow{q_{max}} gateHS[c] + strandBS[c] \\
gateHS[c] + strandBS[c] &\xrightarrow{q_{max}} c + gateLS[c] \\
\\
C10 + gateLS[C10] &\xrightarrow{k_1} gateHS[C10] + strandBS[C10] \\
gateHS[C10] + strandBS[C10] &\xrightarrow{q_{max}} C10 + gateLS[C10] \\
\\
C11 + gateLS[C11] &\xrightarrow{k_1} gateHS[C11] + strandBS[C11] \\
gateHS[C11] + strandBS[C11] &\xrightarrow{q_{max}} C11 + gateLS[C11] \\
\\
C20 + gateLS[C20] &\xrightarrow{q_{max}} gateHS[C20] + strandBS[C20] \\
gateHS[C20] + strandBS[C20] &\xrightarrow{q_{max}} C20 + gateLS[C20] \\
\\
C21 + gateLS[C21] &\xrightarrow{q_{max}} gateHS[C21] + strandBS[C21] \\
gateHS[C21] + strandBS[C21] &\xrightarrow{q_{max}} C21 + gateLS[C21] \\
\\
C30 + gateLS[C30] &\xrightarrow{q_{max}} gateHS[C30] + strandBS[C30] \\
gateHS[C30] + strandBS[C30] &\xrightarrow{q_{max}} C30 + gateLS[C30] \\
\\
C31 + gateLS[C31] &\xrightarrow{q_{max}} gateHS[C31] + strandBS[C31] \\
gateHS[C31] + strandBS[C31] &\xrightarrow{q_{max}} C31 + gateLS[C31] \\
\\
C40 + gateLS[C40] &\xrightarrow{q_{max}} gateHS[C40] + strandBS[C40] \\
gateHS[C40] + strandBS[C40] &\xrightarrow{q_{max}} C40 + gateLS[C40] \\
\\
C41 + gateLS[C41] &\xrightarrow{q_{max}} gateHS[C41] + strandBS[C41] \\
gateHS[C41] + strandBS[C41] &\xrightarrow{q_{max}} C41 + gateLS[C41] \\
\\
C50 + gateLS[C50] &\xrightarrow{q_{max}} gateHS[C50] + strandBS[C50] \\
gateHS[C50] + strandBS[C50] &\xrightarrow{q_{max}} C50 + gateLS[C50] \\
\\
C51 + gateLS[C51] &\xrightarrow{q_{max}} gateHS[C51] + strandBS[C51] \\
gateHS[C51] + strandBS[C51] &\xrightarrow{q_{max}} C51 + gateLS[C51] \\
\\
C60 + gateLS[C60] &\xrightarrow{q_{max}} gateHS[C60] + strandBS[C60] \\
gateHS[C60] + strandBS[C60] &\xrightarrow{q_{max}} C60 + gateLS[C60]
\end{aligned}
$$

$$
\begin{aligned}
C61 + gateLS[C61] &\xrightarrow{q_{max}} gateHS[C61] + strandBS[C61] \\
gateHS[C61] + strandBS[C61] &\xrightarrow{q_{max}} C61 + gateLS[C61] \\
\\
cp + gateLS[cp] &\xrightarrow{k_3} gateHS[cp] + strandBS[cp] \\
gateHS[cp] + strandBS[cp] &\xrightarrow{q_{max}} cp + gateLS[cp] \\
\\
\varnothing + gateLS[\varnothing] &\xrightarrow{q_{max}} gateHS[\varnothing] + strandBS[\varnothing] \\
gateHS[\varnothing] + strandBS[\varnothing] &\xrightarrow{q_{max}} \varnothing + gateLS[\varnothing] \\
\\
W10 + gateLS[W10] &\xrightarrow{q_{max}} gateHS[W10] + strandBS[W10] \\
gateHS[W10] + strandBS[W10] &\xrightarrow{q_{max}} W10 + gateLS[W10] \\
\\
W100 + gateLS[W100] &\xrightarrow{q_{max}} gateHS[W100] + strandBS[W100] \\
gateHS[W100] + strandBS[W100] &\xrightarrow{q_{max}} W100 + gateLS[W100] \\
\\
W101 + gateLS[W101] &\xrightarrow{q_{max}} gateHS[W101] + strandBS[W101] \\
gateHS[W101] + strandBS[W101] &\xrightarrow{q_{max}} W101 + gateLS[W101] \\
\\
W11 + gateLS[W11] &\xrightarrow{q_{max}} gateHS[W11] + strandBS[W11] \\
gateHS[W11] + strandBS[W11] &\xrightarrow{q_{max}} W11 + gateLS[W11] \\
\\
W110 + gateLS[W110] &\xrightarrow{q_{max}} gateHS[W110] + strandBS[W110] \\
gateHS[W110] + strandBS[W110] &\xrightarrow{q_{max}} W110 + gateLS[W110] \\
\\
W111 + gateLS[W111] &\xrightarrow{q_{max}} gateHS[W111] + strandBS[W111] \\
gateHS[W111] + strandBS[W111] &\xrightarrow{q_{max}} W111 + gateLS[W111] \\
\\
W120 + gateLS[W120] &\xrightarrow{q_{max}} gateHS[W120] + strandBS[W120] \\
gateHS[W120] + strandBS[W120] &\xrightarrow{q_{max}} W120 + gateLS[W120] \\
\\
W121 + gateLS[W121] &\xrightarrow{q_{max}} gateHS[W121] + strandBS[W121] \\
gateHS[W121] + strandBS[W121] &\xrightarrow{q_{max}} W121 + gateLS[W121] \\
\\
W130 + gateLS[W130] &\xrightarrow{q_{max}} gateHS[W130] + strandBS[W130] \\
gateHS[W130] + strandBS[W130] &\xrightarrow{q_{max}} W130 + gateLS[W130]
\end{aligned}
$$

$$W131 + gateLS[W131] \xrightarrow{q_{max}} gateHS[W131] + strandBS[W131]$$
$$gateHS[W131] + strandBS[W131] \xrightarrow{q_{max}} W131 + gateLS[W131]$$

$$W140 + gateLS[W140] \xrightarrow{q_{max}} gateHS[W140] + strandBS[W140]$$
$$gateHS[W140] + strandBS[W140] \xrightarrow{q_{max}} W140 + gateLS[W140]$$

$$W141 + gateLS[W141] \xrightarrow{q_{max}} gateHS[W141] + strandBS[W141]$$
$$gateHS[W141] + strandBS[W141] \xrightarrow{q_{max}} W141 + gateLS[W141]$$

$$W150 + gateLS[W150] \xrightarrow{q_{max}} gateHS[W150] + strandBS[W150]$$
$$gateHS[W150] + strandBS[W150] \xrightarrow{q_{max}} W150 + gateLS[W150]$$

$$W151 + gateLS[W151] \xrightarrow{q_{max}} gateHS[W151] + strandBS[W151]$$
$$gateHS[W151] + strandBS[W151] \xrightarrow{q_{max}} W151 + gateLS[W151]$$

$$W160 + gateLS[W160] \xrightarrow{q_{max}} gateHS[W160] + strandBS[W160]$$
$$gateHS[W160] + strandBS[W160] \xrightarrow{q_{max}} W160 + gateLS[W160]$$

$$W161 + gateLS[W161] \xrightarrow{q_{max}} gateHS[W161] + strandBS[W161]$$
$$gateHS[W161] + strandBS[W161] \xrightarrow{q_{max}} W161 + gateLS[W161]$$

$$W170 + gateLS[W170] \xrightarrow{q_{max}} gateHS[W170] + strandBS[W170]$$
$$gateHS[W170] + strandBS[W170] \xrightarrow{q_{max}} W170 + gateLS[W170]$$

$$W171 + gateLS[W171] \xrightarrow{q_{max}} gateHS[W171] + strandBS[W171]$$
$$gateHS[W171] + strandBS[W171] \xrightarrow{q_{max}} W171 + gateLS[W171]$$

$$W180 + gateLS[W180] \xrightarrow{q_{max}} gateHS[W180] + strandBS[W180]$$
$$gateHS[W180] + strandBS[W180] \xrightarrow{q_{max}} W180 + gateLS[W180]$$

$$W181 + gateLS[W181] \xrightarrow{q_{max}} gateHS[W181] + strandBS[W181]$$
$$gateHS[W181] + strandBS[W181] \xrightarrow{q_{max}} W181 + gateLS[W181]$$

$$W190 + gateLS[W190] \xrightarrow{q_{max}} gateHS[W190] + strandBS[W190]$$
$$gateHS[W190] + strandBS[W190] \xrightarrow{q_{max}} W190 + gateLS[W190]$$

$$W191 + gateLS[W191] \xrightarrow{q_{max}} gateHS[W191] + strandBS[W191]$$
$$gateHS[W191] + strandBS[W191] \xrightarrow{q_{max}} W191 + gateLS[W191]$$

$$W20 + gateLS[W20] \xrightarrow{q_{max}} gateHS[W20] + strandBS[W20]$$
$$gateHS[W20] + strandBS[W20] \xrightarrow{q_{max}} W20 + gateLS[W20]$$

$$W200 + gateLS[W200] \xrightarrow{q_{max}} gateHS[W200] + strandBS[W200]$$
$$gateHS[W200] + strandBS[W200] \xrightarrow{q_{max}} W200 + gateLS[W200]$$

$$W201 + gateLS[W201] \xrightarrow{q_{max}} gateHS[W201] + strandBS[W201]$$
$$gateHS[W201] + strandBS[W201] \xrightarrow{q_{max}} W201 + gateLS[W201]$$

$$W21 + gateLS[W21] \xrightarrow{q_{max}} gateHS[W21] + strandBS[W21]$$
$$gateHS[W21] + strandBS[W21] \xrightarrow{q_{max}} W21 + gateLS[W21]$$

$$W210 + gateLS[W210] \xrightarrow{q_{max}} gateHS[W210] + strandBS[W210]$$
$$gateHS[W210] + strandBS[W210] \xrightarrow{q_{max}} W210 + gateLS[W210]$$

$$W211 + gateLS[W211] \xrightarrow{q_{max}} gateHS[W211] + strandBS[W211]$$
$$gateHS[W211] + strandBS[W211] \xrightarrow{q_{max}} W211 + gateLS[W211]$$

$$W220 + gateLS[W220] \xrightarrow{q_{max}} gateHS[W220] + strandBS[W220]$$
$$gateHS[W220] + strandBS[W220] \xrightarrow{q_{max}} W220 + gateLS[W220]$$

$$W221 + gateLS[W221] \xrightarrow{q_{max}} gateHS[W221] + strandBS[W221]$$
$$gateHS[W221] + strandBS[W221] \xrightarrow{q_{max}} W221 + gateLS[W221]$$

$$W230 + gateLS[W230] \xrightarrow{q_{max}} gateHS[W230] + strandBS[W230]$$
$$gateHS[W230] + strandBS[W230] \xrightarrow{q_{max}} W230 + gateLS[W230]$$

$$W231 + gateLS[W231] \xrightarrow{q_{max}} gateHS[W231] + strandBS[W231]$$
$$gateHS[W231] + strandBS[W231] \xrightarrow{q_{max}} W231 + gateLS[W231]$$

$$W240 + gateLS[W240] \xrightarrow{q_{max}} gateHS[W240] + strandBS[W240]$$
$$gateHS[W240] + strandBS[W240] \xrightarrow{q_{max}} W240 + gateLS[W240]$$

$$W241 + gateLS[W241] \xrightarrow{q_{max}} gateHS[W241] + strandBS[W241]$$
$$gateHS[W241] + strandBS[W241] \xrightarrow{q_{max}} W241 + gateLS[W241]$$

$$W250 + gateLS[W250] \xrightarrow{q_{max}} gateHS[W250] + strandBS[W250]$$
$$gateHS[W250] + strandBS[W250] \xrightarrow{q_{max}} W250 + gateLS[W250]$$

$$W251 + gateLS[W251] \xrightarrow{q_{max}} gateHS[W251] + strandBS[W251]$$
$$gateHS[W251] + strandBS[W251] \xrightarrow{q_{max}} W251 + gateLS[W251]$$

$$W260 + gateLS[W260] \xrightarrow{q_{max}} gateHS[W260] + strandBS[W260]$$
$$gateHS[W260] + strandBS[W260] \xrightarrow{q_{max}} W260 + gateLS[W260]$$

$$W261 + gateLS[W261] \xrightarrow{q_{max}} gateHS[W261] + strandBS[W261]$$
$$gateHS[W261] + strandBS[W261] \xrightarrow{q_{max}} W261 + gateLS[W261]$$

$$W270 + gateLS[W270] \xrightarrow{q_{max}} gateHS[W270] + strandBS[W270]$$
$$gateHS[W270] + strandBS[W270] \xrightarrow{q_{max}} W270 + gateLS[W270]$$

$$W271 + gateLS[W271] \xrightarrow{q_{max}} gateHS[W271] + strandBS[W271]$$
$$gateHS[W271] + strandBS[W271] \xrightarrow{q_{max}} W271 + gateLS[W271]$$

$$W280 + gateLS[W280] \xrightarrow{q_{max}} gateHS[W280] + strandBS[W280]$$
$$gateHS[W280] + strandBS[W280] \xrightarrow{q_{max}} W280 + gateLS[W280]$$

$$W281 + gateLS[W281] \xrightarrow{q_{max}} gateHS[W281] + strandBS[W281]$$
$$gateHS[W281] + strandBS[W281] \xrightarrow{q_{max}} W281 + gateLS[W281]$$

$$W290 + gateLS[W290] \xrightarrow{q_{max}} gateHS[W290] + strandBS[W290]$$
$$gateHS[W290] + strandBS[W290] \xrightarrow{q_{max}} W290 + gateLS[W290]$$

$$W291 + gateLS[W291] \xrightarrow{q_{max}} gateHS[W291] + strandBS[W291]$$
$$gateHS[W291] + strandBS[W291] \xrightarrow{q_{max}} W291 + gateLS[W291]$$

$$W30 + gateLS[W30] \xrightarrow{q_{max}} gateHS[W30] + strandBS[W30]$$
$$gateHS[W30] + strandBS[W30] \xrightarrow{q_{max}} W30 + gateLS[W30]$$

$$W300 + gateLS[W300] \xrightarrow{q_{max}} gateHS[W300] + strandBS[W300]$$
$$gateHS[W300] + strandBS[W300] \xrightarrow{q_{max}} W300 + gateLS[W300]$$

$$W301 + gateLS[W301] \xrightarrow{q_{max}} gateHS[W301] + strandBS[W301]$$
$$gateHS[W301] + strandBS[W301] \xrightarrow{q_{max}} W301 + gateLS[W301]$$

$$W31 + gateLS[W31] \xrightarrow{q_{max}} gateHS[W31] + strandBS[W31]$$
$$gateHS[W31] + strandBS[W31] \xrightarrow{q_{max}} W31 + gateLS[W31]$$

$$W310 + gateLS[W310] \xrightarrow{q_{max}} gateHS[W310] + strandBS[W310]$$
$$gateHS[W310] + strandBS[W310] \xrightarrow{q_{max}} W310 + gateLS[W310]$$

$$W311 + gateLS[W311] \xrightarrow{q_{max}} gateHS[W311] + strandBS[W311]$$
$$gateHS[W311] + strandBS[W311] \xrightarrow{q_{max}} W311 + gateLS[W311]$$

$$W320 + gateLS[W320] \xrightarrow{q_{max}} gateHS[W320] + strandBS[W320]$$
$$gateHS[W320] + strandBS[W320] \xrightarrow{q_{max}} W320 + gateLS[W320]$$

$$W321 + gateLS[W321] \xrightarrow{q_{max}} gateHS[W321] + strandBS[W321]$$
$$gateHS[W321] + strandBS[W321] \xrightarrow{q_{max}} W321 + gateLS[W321]$$

$$W330 + gateLS[W330] \xrightarrow{q_{max}} gateHS[W330] + strandBS[W330]$$
$$gateHS[W330] + strandBS[W330] \xrightarrow{q_{max}} W330 + gateLS[W330]$$

$$W331 + gateLS[W331] \xrightarrow{q_{max}} gateHS[W331] + strandBS[W331]$$
$$gateHS[W331] + strandBS[W331] \xrightarrow{q_{max}} W331 + gateLS[W331]$$

$$W40 + gateLS[W40] \xrightarrow{q_{max}} gateHS[W40] + strandBS[W40]$$
$$gateHS[W40] + strandBS[W40] \xrightarrow{q_{max}} W40 + gateLS[W40]$$

$$W41 + gateLS[W41] \xrightarrow{q_{max}} gateHS[W41] + strandBS[W41]$$
$$gateHS[W41] + strandBS[W41] \xrightarrow{q_{max}} W41 + gateLS[W41]$$

$$W50 + gateLS[W50] \xrightarrow{q_{max}} gateHS[W50] + strandBS[W50]$$
$$gateHS[W50] + strandBS[W50] \xrightarrow{q_{max}} W50 + gateLS[W50]$$

$$W51 + gateLS[W51] \xrightarrow{q_{max}} gateHS[W51] + strandBS[W51]$$
$$gateHS[W51] + strandBS[W51] \xrightarrow{q_{max}} W51 + gateLS[W51]$$

$$W60 + gateLS[W60] \xrightarrow{q_{max}} gateHS[W60] + strandBS[W60]$$
$$gateHS[W60] + strandBS[W60] \xrightarrow{q_{max}} W60 + gateLS[W60]$$

$$W61 + gateLS[W61] \xrightarrow{q_{max}} gateHS[W61] + strandBS[W61]$$
$$gateHS[W61] + strandBS[W61] \xrightarrow{q_{max}} W61 + gateLS[W61]$$

$$W70 + gateLS[W70] \xrightarrow{q_{max}} gateHS[W70] + strandBS[W70]$$
$$gateHS[W70] + strandBS[W70] \xrightarrow{q_{max}} W70 + gateLS[W70]$$

$$W71 + gateLS[W71] \xrightarrow{q_{max}} gateHS[W71] + strandBS[W71]$$
$$gateHS[W71] + strandBS[W71] \xrightarrow{q_{max}} W71 + gateLS[W71]$$

$$W80 + gateLS[W80] \xrightarrow{q_{max}} gateHS[W80] + strandBS[W80]$$
$$gateHS[W80] + strandBS[W80] \xrightarrow{q_{max}} W80 + gateLS[W80]$$

$$W81 + gateLS[W81] \xrightarrow{q_{max}} gateHS[W81] + strandBS[W81]$$
$$gateHS[W81] + strandBS[W81] \xrightarrow{q_{max}} W81 + gateLS[W81]$$

$$W90 + gateLS[W90] \xrightarrow{q_{max}} gateHS[W90] + strandBS[W90]$$
$$gateHS[W90] + strandBS[W90] \xrightarrow{q_{max}} W90 + gateLS[W90]$$

$$W91 + gateLS[W91] \xrightarrow{q_{max}} gateHS[W91] + strandBS[W91]$$
$$gateHS[W91] + strandBS[W91] \xrightarrow{q_{max}} W91 + gateLS[W91]$$

$$X10 + gateLS[X10] \xrightarrow{k_1} gateHS[X10] + strandBS[X10]$$
$$gateHS[X10] + strandBS[X10] \xrightarrow{q_{max}} X10 + gateLS[X10]$$

$$X100 + gateLS[X100] \xrightarrow{k_1} gateHS[X100] + strandBS[X100]$$
$$gateHS[X100] + strandBS[X100] \xrightarrow{q_{max}} X100 + gateLS[X100]$$

$$X101 + gateLS[X101] \xrightarrow{k_1} gateHS[X101] + strandBS[X101]$$
$$gateHS[X101] + strandBS[X101] \xrightarrow{q_{max}} X101 + gateLS[X101]$$

$$X11 + gateLS[X11] \xrightarrow{k_1} gateHS[X11] + strandBS[X11]$$
$$gateHS[X11] + strandBS[X11] \xrightarrow{q_{max}} X11 + gateLS[X11]$$

$$X110 + gateLS[X110] \xrightarrow{k_1} gateHS[X110] + strandBS[X110]$$
$$gateHS[X110] + strandBS[X110] \xrightarrow{q_{max}} X110 + gateLS[X110]$$

$$X111 + gateLS[X111] \xrightarrow{k_1} gateHS[X111] + strandBS[X111]$$
$$gateHS[X111] + strandBS[X111] \xrightarrow{q_{max}} X111 + gateLS[X111]$$

$$X120 + gateLS[X120] \xrightarrow{k_1} gateHS[X120] + strandBS[X120]$$
$$gateHS[X120] + strandBS[X120] \xrightarrow{q_{max}} X120 + gateLS[X120]$$

$$X121 + gateLS[X121] \xrightarrow{k_1} gateHS[X121] + strandBS[X121]$$
$$gateHS[X121] + strandBS[X121] \xrightarrow{q_{max}} X121 + gateLS[X121]$$

$$X130 + gateLS[X130] \xrightarrow{k_1} gateHS[X130] + strandBS[X130]$$
$$gateHS[X130] + strandBS[X130] \xrightarrow{q_{max}} X130 + gateLS[X130]$$

$$X131 + gateLS[X131] \xrightarrow{k_1} gateHS[X131] + strandBS[X131]$$
$$gateHS[X131] + strandBS[X131] \xrightarrow{q_{max}} X131 + gateLS[X131]$$

$$X140 + gateLS[X140] \xrightarrow{k_1} gateHS[X140] + strandBS[X140]$$
$$gateHS[X140] + strandBS[X140] \xrightarrow{q_{max}} X140 + gateLS[X140]$$

$$X141 + gateLS[X141] \xrightarrow{k_1} gateHS[X141] + strandBS[X141]$$
$$gateHS[X141] + strandBS[X141] \xrightarrow{q_{max}} X141 + gateLS[X141]$$

$$X150 + gateLS[X150] \xrightarrow{k_1} gateHS[X150] + strandBS[X150]$$
$$gateHS[X150] + strandBS[X150] \xrightarrow{q_{max}} X150 + gateLS[X150]$$

$$X151 + gateLS[X151] \xrightarrow{k_1} gateHS[X151] + strandBS[X151]$$
$$gateHS[X151] + strandBS[X151] \xrightarrow{q_{max}} X151 + gateLS[X151]$$

$$X160 + gateLS[X160] \xrightarrow{k_1} gateHS[X160] + strandBS[X160]$$
$$gateHS[X160] + strandBS[X160] \xrightarrow{q_{max}} X160 + gateLS[X160]$$

$$X161 + gateLS[X161] \xrightarrow{k_1} gateHS[X161] + strandBS[X161]$$
$$gateHS[X161] + strandBS[X161] \xrightarrow{q_{max}} X161 + gateLS[X161]$$

$$X170 + gateLS[X170] \xrightarrow{k_1} gateHS[X170] + strandBS[X170]$$
$$gateHS[X170] + strandBS[X170] \xrightarrow{q_{max}} X170 + gateLS[X170]$$

$$X171 + gateLS[X171] \xrightarrow{k_1} gateHS[X171] + strandBS[X171]$$
$$gateHS[X171] + strandBS[X171] \xrightarrow{q_{max}} X171 + gateLS[X171]$$

$$X180 + gateLS[X180] \xrightarrow{k_1} gateHS[X180] + strandBS[X180]$$
$$gateHS[X180] + strandBS[X180] \xrightarrow{q_{max}} X180 + gateLS[X180]$$

$$X181 + gateLS[X181] \xrightarrow{k_1} gateHS[X181] + strandBS[X181]$$
$$gateHS[X181] + strandBS[X181] \xrightarrow{q_{max}} X181 + gateLS[X181]$$

$$X190 + gateLS[X190] \xrightarrow{k_1} gateHS[X190] + strandBS[X190]$$
$$gateHS[X190] + strandBS[X190] \xrightarrow{q_{max}} X190 + gateLS[X190]$$

$$X191 + gateLS[X191] \xrightarrow{k_1} gateHS[X191] + strandBS[X191]$$
$$gateHS[X191] + strandBS[X191] \xrightarrow{q_{max}} X191 + gateLS[X191]$$

$$X20 + gateLS[X20] \xrightarrow{k_1} gateHS[X20] + strandBS[X20]$$
$$gateHS[X20] + strandBS[X20] \xrightarrow{q_{max}} X20 + gateLS[X20]$$

$$X200 + gateLS[X200] \xrightarrow{k_1} gateHS[X200] + strandBS[X200]$$
$$gateHS[X200] + strandBS[X200] \xrightarrow{q_{max}} X200 + gateLS[X200]$$

$$X201 + gateLS[X201] \xrightarrow{k_1} gateHS[X201] + strandBS[X201]$$
$$gateHS[X201] + strandBS[X201] \xrightarrow{q_{max}} X201 + gateLS[X201]$$

$$X21 + gateLS[X21] \xrightarrow{k_1} gateHS[X21] + strandBS[X21]$$
$$gateHS[X21] + strandBS[X21] \xrightarrow{q_{max}} X21 + gateLS[X21]$$

$$X210 + gateLS[X210] \xrightarrow{k_1} gateHS[X210] + strandBS[X210]$$
$$gateHS[X210] + strandBS[X210] \xrightarrow{q_{max}} X210 + gateLS[X210]$$

$$X211 + gateLS[X211] \xrightarrow{k_1} gateHS[X211] + strandBS[X211]$$
$$gateHS[X211] + strandBS[X211] \xrightarrow{q_{max}} X211 + gateLS[X211]$$

$$X220 + gateLS[X220] \xrightarrow{k_1} gateHS[X220] + strandBS[X220]$$
$$gateHS[X220] + strandBS[X220] \xrightarrow{q_{max}} X220 + gateLS[X220]$$

$$X221 + gateLS[X221] \xrightarrow{k_1} gateHS[X221] + strandBS[X221]$$
$$gateHS[X221] + strandBS[X221] \xrightarrow{q_{max}} X221 + gateLS[X221]$$

$$X230 + gateLS[X230] \xrightarrow{k_1} gateHS[X230] + strandBS[X230]$$
$$gateHS[X230] + strandBS[X230] \xrightarrow{q_{max}} X230 + gateLS[X230]$$

$$X231 + gateLS[X231] \xrightarrow{k_1} gateHS[X231] + strandBS[X231]$$
$$gateHS[X231] + strandBS[X231] \xrightarrow{q_{max}} X231 + gateLS[X231]$$

$$X240 + gateLS[X240] \xrightarrow{k_1} gateHS[X240] + strandBS[X240]$$
$$gateHS[X240] + strandBS[X240] \xrightarrow{q_{max}} X240 + gateLS[X240]$$

$$X241 + gateLS[X241] \xrightarrow{k_1} gateHS[X241] + strandBS[X241]$$
$$gateHS[X241] + strandBS[X241] \xrightarrow{q_{max}} X241 + gateLS[X241]$$

$$X250 + gateLS[X250] \xrightarrow{k_1} gateHS[X250] + strandBS[X250]$$
$$gateHS[X250] + strandBS[X250] \xrightarrow{q_{max}} X250 + gateLS[X250]$$

$$X251 + gateLS[X251] \xrightarrow{k_1} gateHS[X251] + strandBS[X251]$$
$$gateHS[X251] + strandBS[X251] \xrightarrow{q_{max}} X251 + gateLS[X251]$$

$$X260 + gateLS[X260] \xrightarrow{k_1} gateHS[X260] + strandBS[X260]$$
$$gateHS[X260] + strandBS[X260] \xrightarrow{q_{max}} X260 + gateLS[X260]$$

$$X261 + gateLS[X261] \xrightarrow{k_1} gateHS[X261] + strandBS[X261]$$
$$gateHS[X261] + strandBS[X261] \xrightarrow{q_{max}} X261 + gateLS[X261]$$

$$X270 + gateLS[X270] \xrightarrow{k_1} gateHS[X270] + strandBS[X270]$$
$$gateHS[X270] + strandBS[X270] \xrightarrow{q_{max}} X270 + gateLS[X270]$$

$$X271 + gateLS[X271] \xrightarrow{k_1} gateHS[X271] + strandBS[X271]$$
$$gateHS[X271] + strandBS[X271] \xrightarrow{q_{max}} X271 + gateLS[X271]$$

$$X280 + gateLS[X280] \xrightarrow{k_1} gateHS[X280] + strandBS[X280]$$
$$gateHS[X280] + strandBS[X280] \xrightarrow{q_{max}} X280 + gateLS[X280]$$

$$X281 + gateLS[X281] \xrightarrow{k_1} gateHS[X281] + strandBS[X281]$$
$$gateHS[X281] + strandBS[X281] \xrightarrow{q_{max}} X281 + gateLS[X281]$$

$$X290 + gateLS[X290] \xrightarrow{k_1} gateHS[X290] + strandBS[X290]$$
$$gateHS[X290] + strandBS[X290] \xrightarrow{q_{max}} X290 + gateLS[X290]$$

$$X291 + gateLS[X291] \xrightarrow{k_1} gateHS[X291] + strandBS[X291]$$
$$gateHS[X291] + strandBS[X291] \xrightarrow{q_{max}} X291 + gateLS[X291]$$

$$X30 + gateLS[X30] \xrightarrow{k_1} gateHS[X30] + strandBS[X30]$$
$$gateHS[X30] + strandBS[X30] \xrightarrow{q_{max}} X30 + gateLS[X30]$$

$$X300 + gateLS[X300] \xrightarrow{k_1} gateHS[X300] + strandBS[X300]$$
$$gateHS[X300] + strandBS[X300] \xrightarrow{q_{max}} X300 + gateLS[X300]$$

$$X301 + gateLS[X301] \xrightarrow{k_1} gateHS[X301] + strandBS[X301]$$
$$gateHS[X301] + strandBS[X301] \xrightarrow{q_{max}} X301 + gateLS[X301]$$

$$X31 + gateLS[X31] \xrightarrow{k_1} gateHS[X31] + strandBS[X31]$$
$$gateHS[X31] + strandBS[X31] \xrightarrow{q_{max}} X31 + gateLS[X31]$$

$$X310 + gateLS[X310] \xrightarrow{k_1} gateHS[X310] + strandBS[X310]$$
$$gateHS[X310] + strandBS[X310] \xrightarrow{q_{max}} X310 + gateLS[X310]$$

$$X311 + gateLS[X311] \xrightarrow{k_1} gateHS[X311] + strandBS[X311]$$
$$gateHS[X311] + strandBS[X311] \xrightarrow{q_{max}} X311 + gateLS[X311]$$

$$X320 + gateLS[X320] \xrightarrow{k_1} gateHS[X320] + strandBS[X320]$$
$$gateHS[X320] + strandBS[X320] \xrightarrow{q_{max}} X320 + gateLS[X320]$$

$$X321 + gateLS[X321] \xrightarrow{k_1} gateHS[X321] + strandBS[X321]$$
$$gateHS[X321] + strandBS[X321] \xrightarrow{q_{max}} X321 + gateLS[X321]$$

$$X330 + gateLS[X330] \xrightarrow{k_1} gateHS[X330] + strandBS[X330]$$
$$gateHS[X330] + strandBS[X330] \xrightarrow{q_{max}} X330 + gateLS[X330]$$

$$X331 + gateLS[X331] \xrightarrow{k_1} gateHS[X331] + strandBS[X331]$$
$$gateHS[X331] + strandBS[X331] \xrightarrow{q_{max}} X331 + gateLS[X331]$$

$$X40 + gateLS[X40] \xrightarrow{k_1} gateHS[X40] + strandBS[X40]$$
$$gateHS[X40] + strandBS[X40] \xrightarrow{q_{max}} X40 + gateLS[X40]$$

$$X41 + gateLS[X41] \xrightarrow{k_1} gateHS[X41] + strandBS[X41]$$
$$gateHS[X41] + strandBS[X41] \xrightarrow{q_{max}} X41 + gateLS[X41]$$

$$X50 + gateLS[X50] \xrightarrow{k_1} gateHS[X50] + strandBS[X50]$$
$$gateHS[X50] + strandBS[X50] \xrightarrow{q_{max}} X50 + gateLS[X50]$$

$$X51 + gateLS[X51] \xrightarrow{k_1} gateHS[X51] + strandBS[X51]$$
$$gateHS[X51] + strandBS[X51] \xrightarrow{q_{max}} X51 + gateLS[X51]$$

$$X60 + gateLS[X60] \xrightarrow{k_1} gateHS[X60] + strandBS[X60]$$
$$gateHS[X60] + strandBS[X60] \xrightarrow{q_{max}} X60 + gateLS[X60]$$

$$X61 + gateLS[X61] \xrightarrow{k_1} gateHS[X61] + strandBS[X61]$$
$$gateHS[X61] + strandBS[X61] \xrightarrow{q_{max}} X61 + gateLS[X61]$$

$$X70 + gateLS[X70] \xrightarrow{k_1} gateHS[X70] + strandBS[X70]$$
$$gateHS[X70] + strandBS[X70] \xrightarrow{q_{max}} X70 + gateLS[X70]$$

$$X71 + gateLS[X71] \xrightarrow{k_1} gateHS[X71] + strandBS[X71]$$
$$gateHS[X71] + strandBS[X71] \xrightarrow{q_{max}} X71 + gateLS[X71]$$

$$X80 + gateLS[X80] \xrightarrow{k_1} gateHS[X80] + strandBS[X80]$$
$$gateHS[X80] + strandBS[X80] \xrightarrow{q_{max}} X80 + gateLS[X80]$$

$$
\begin{aligned}
X81 + gateLS[X81] &\xrightarrow{k_1} gateHS[X81] + strandBS[X81] \\
gateHS[X81] + strandBS[X81] &\xrightarrow{q_{max}} X81 + gateLS[X81] \\
\\
X90 + gateLS[X90] &\xrightarrow{k_1} gateHS[X90] + strandBS[X90] \\
gateHS[X90] + strandBS[X90] &\xrightarrow{q_{max}} X90 + gateLS[X90] \\
\\
X91 + gateLS[X91] &\xrightarrow{k_1} gateHS[X91] + strandBS[X91] \\
gateHS[X91] + strandBS[X91] &\xrightarrow{q_{max}} X91 + gateLS[X91]
\end{aligned}
$$

A.2.2 ADC-3bit DNA

$$
\begin{aligned}
i1 + gateL[1] &\xrightarrow{k} gateH[1] + strandB[1] \\
gateH[1] + strandB[1] &\xrightarrow{k} i1 + gateL[1] \\
T1 + gateH[1] &\xrightarrow{k} strandO[1] \\
strandO[1] + gateT[1] &\xrightarrow{k} W1 \\
i1 + gateL[2] &\xrightarrow{k} gateH[2] + strandB[2] \\
gateH[2] + strandB[2] &\xrightarrow{k} i1 + gateL[2] \\
x2n + gateH[2] &\xrightarrow{k} strandO[2] \\
strandO[2] + gateT[2] &\xrightarrow{k} i1 + x2p \\
\\
T1 + gateL[3] &\xrightarrow{k} gateH[3] + strandB[3] \\
gateH[3] + strandB[3] &\xrightarrow{k} T1 + gateL[3] \\
x2p + gateH[3] &\xrightarrow{k} strandO[3] \\
strandO[3] + gateT[3] &\xrightarrow{k} T1 + x2n \\
\\
x2p + i1 + T2 &\xrightarrow{k} W2 + x2p \\
x2n + gateL[5] &\xrightarrow{k} gateH[5] + strandB[5] \\
gateH[5] + strandB[5] &\xrightarrow{k} x2n + gateL[5] \\
W2 + gateH[5] &\xrightarrow{k} strandO[5] \\
strandO[5] + gateT[5] &\xrightarrow{k} T2 + x2n + i1
\end{aligned}
$$

$$
\begin{aligned}
x2p + i1 + x1n &\xrightarrow{k} x1p + i1 + x2p \\
x2p + x1p + T2 &\xrightarrow{k} T2 + x1n + x2p \\
x2n + W1 + Tp2 &\xrightarrow{k} Wp2 + x2n \\
x2p + gateL[9] &\xrightarrow{k} gateH[9] + strandB[9] \\
gateH[9] + strandB[9] &\xrightarrow{k} x2p + gateL[9] \\
Wp2 + gateH[9] &\xrightarrow{k} strandO[9] \\
strandO[9] + gateT[9] &\xrightarrow{k} Tp2 + x2p + W1
\end{aligned}
$$

$$
\begin{aligned}
x2n + W1 + x1n &\xrightarrow{k} x1p + W1 + x2n \\
x2n + x1p + Tp2 &\xrightarrow{k} Tp2 + x1n + x2n \\
x2p + gateL[12] &\xrightarrow{k} gateH[12] + strandB[12] \\
gateH[12] + strandB[12] &\xrightarrow{k} x2p + gateL[12] \\
x1p + gateH[12] &\xrightarrow{k} strandO[12] \\
strandO[12] + gateT[12] &\xrightarrow{k} q1 + x2p + x1p
\end{aligned}
$$

$$
\begin{aligned}
q1 + gateL[13] &\xrightarrow{k} gateH[13] + strandB[13] \\
gateH[13] + strandB[13] &\xrightarrow{k} q1 + gateL[13] \\
q1 + gateH[13] &\xrightarrow{k} strandO[13] \\
strandO[13] + gateT[13] &\xrightarrow{k} nth \\
i1 + gateL[14] &\xrightarrow{k} gateH[14] + strandB[14] \\
gateH[14] + strandB[14] &\xrightarrow{k} i1 + gateL[14] \\
T3 + gateH[14] &\xrightarrow{k} strandO[14] \\
strandO[14] + gateT[14] &\xrightarrow{k} q2 \\
q1 + gateL[15] &\xrightarrow{k} gateH[15] + strandB[15] \\
gateH[15] + strandB[15] &\xrightarrow{k} q1 + gateL[15] \\
q2 + gateH[15] &\xrightarrow{k} strandO[15] \\
strandO[15] + gateT[15] &\xrightarrow{k} W3 \\
W3 + gateL[16] &\xrightarrow{k} gateH[16] + strandB[16] \\
gateH[16] + strandB[16] &\xrightarrow{k} W3 + gateL[16] \\
x1n + gateH[16] &\xrightarrow{k} strandO[16] \\
strandO[16] + gateT[16] &\xrightarrow{k} i1 + T3 + x1n \\
W3 + gateL[17] &\xrightarrow{k} gateH[17] + strandB[17] \\
gateH[17] + strandB[17] &\xrightarrow{k} W3 + gateL[17] \\
x2n + gateH[17] &\xrightarrow{k} strandO[17] \\
strandO[17] + gateT[17] &\xrightarrow{k} i1 + T3 + x2n \\
\\
q2 + gateL[18] &\xrightarrow{k} gateH[18] + strandB[18] \\
gateH[18] + strandB[18] &\xrightarrow{k} q2 + gateL[18] \\
x1n + gateH[18] &\xrightarrow{k} strandO[18] \\
strandO[18] + gateT[18] &\xrightarrow{k} i1 + T3 + x1n
\end{aligned}
$$

$$
\begin{aligned}
q2 + gateL[19] &\xrightarrow{k} gateH[19] + strandB[19] \\
gateH[19] + strandB[19] &\xrightarrow{k} q2 + gateL[19] \\
x2n + gateH[19] &\xrightarrow{k} strandO[19] \\
strandO[19] + gateT[19] &\xrightarrow{k} i1 + T3 + x2n \\
\\
x1n + gateL[20] &\xrightarrow{k} gateH[20] + strandB[20] \\
gateH[20] + strandB[20] &\xrightarrow{k} x1n + gateL[20] \\
W3 + gateH[20] &\xrightarrow{k} strandO[20] \\
strandO[20] + gateT[20] &\xrightarrow{k} T3 + i1 + x1n \\
\\
x2n + gateL[21] &\xrightarrow{k} gateH[21] + strandB[21] \\
gateH[21] + strandB[21] &\xrightarrow{k} x2n + gateL[21] \\
W3 + gateH[21] &\xrightarrow{k} strandO[21] \\
strandO[21] + gateT[21] &\xrightarrow{k} T3 + i1 + x2n \\
\\
i1 + gateL[22] &\xrightarrow{k} gateH[22] + strandB[22] \\
gateH[22] + strandB[22] &\xrightarrow{k} i1 + gateL[22] \\
x0n + gateH[22] &\xrightarrow{k} strandO[22] \\
strandO[22] + gateT[22] &\xrightarrow{k} q3 + i1 \\
\\
T3 + gateL[23] &\xrightarrow{k} gateH[23] + strandB[23] \\
gateH[23] + strandB[23] &\xrightarrow{k} T3 + gateL[23] \\
q3 + gateH[23] &\xrightarrow{k} strandO[23] \\
strandO[23] + gateT[23] &\xrightarrow{k} x0n + T3
\end{aligned}
$$

$$
\begin{aligned}
q1 + gateL[24] &\xrightarrow{k} gateH[24] + strandB[24] \\
gateH[24] + strandB[24] &\xrightarrow{k} q1 + gateL[24] \\
q3 + gateH[24] &\xrightarrow{k} strandO[24] \\
strandO[24] + gateT[24] &\xrightarrow{k} x0p \\
x2p + x1p + T3 + x0p &\xrightarrow{k} x0n + x2p + x1p + T3 \\
x2p + x1n + W2 + Tp3 &\xrightarrow{k} Wp3 + x2p + x1n \\
x2n + gateL[27] &\xrightarrow{k} gateH[27] + strandB[27] \\
gateH[27] + strandB[27] &\xrightarrow{k} x2n + gateL[27] \\
Wp3 + gateH[27] &\xrightarrow{k} strandO[27] \\
strandO[27] + gateT[27] &\xrightarrow{k} Tp3 + W2 + x2n \\
\\
x1p + gateL[28] &\xrightarrow{k} gateH[28] + strandB[28] \\
gateH[28] + strandB[28] &\xrightarrow{k} x1p + gateL[28] \\
Wp3 + gateH[28] &\xrightarrow{k} strandO[28] \\
strandO[28] + gateT[28] &\xrightarrow{k} Tp3 + W2 + x1p \\
\\
x2p + x1n + W2 + x0n &\xrightarrow{k} x0p + x2p + x1n + W2 \\
x2p + x1n + Tp3 + x0p &\xrightarrow{k} x0n + x2p + x1n + Tp3 \\
x2n + x1p + W1 + Tpp3 &\xrightarrow{k} Wpp3 + x2n + x1p \\
x2p + gateL[32] &\xrightarrow{k} gateH[32] + strandB[32] \\
gateH[32] + strandB[32] &\xrightarrow{k} x2p + gateL[32] \\
Wpp3 + gateH[32] &\xrightarrow{k} strandO[32] \\
strandO[32] + gateT[32] &\xrightarrow{k} Tpp3 + W1 + x2p \\
\\
x1n + gateL[33] &\xrightarrow{k} gateH[33] + strandB[33] \\
gateH[33] + strandB[33] &\xrightarrow{k} x1n + gateL[33] \\
Wpp3 + gateH[33] &\xrightarrow{k} strandO[33] \\
strandO[33] + gateT[33] &\xrightarrow{k} Tpp3 + W1 + x1n
\end{aligned}
$$

$$
\begin{aligned}
x2n + x1p + W1 + x0n &\xrightarrow{\mathrm{k}} x0p + x2n + x1p + W1 \\
x2n + x1p + Tpp3 + x0p &\xrightarrow{\mathrm{k}} x0n + x2n + x1p + Tpp3 \\
x2n + x1n + Wp2 + Tppp3 &\xrightarrow{\mathrm{k}} Wppp3 + x2n + x1n \\
x2p + gateL[37] &\xrightarrow{\mathrm{k}} gateH[37] + strandB[37] \\
gateH[37] + strandB[37] &\xrightarrow{\mathrm{k}} x2p + gateL[37] \\
Wppp3 + gateH[37] &\xrightarrow{\mathrm{k}} strandO[37] \\
strandO[37] + gateT[37] &\xrightarrow{\mathrm{k}} Tppp3 + Wp2 + x2p
\end{aligned}
$$

$$
\begin{aligned}
x1p + gateL[38] &\xrightarrow{\mathrm{k}} gateH[38] + strandB[38] \\
gateH[38] + strandB[38] &\xrightarrow{\mathrm{k}} x1p + gateL[38] \\
Wppp3 + gateH[38] &\xrightarrow{\mathrm{k}} strandO[38] \\
strandO[38] + gateT[38] &\xrightarrow{\mathrm{k}} Tppp3 + Wp2 + x1p
\end{aligned}
$$

$$
\begin{aligned}
x2n + x1n + Wp2 + x0n &\xrightarrow{\mathrm{k}} x0p + x2n + x1n + Wp2 \\
x2n + x1n + Tppp3 + x0p &\xrightarrow{\mathrm{k}} x0n + x2n + x1n + Tppp3
\end{aligned}
$$

A.2.3 DAC-3bit DNA

$$
\begin{aligned}
b3 + gateL[1] &\xrightarrow{\mathrm{k_{f1}}} gateH[1] + strandB[1] \\
gateH[1] + strandB[1] &\xrightarrow{\mathrm{q_{fmax1}}} b3 + gateL[1] \\
o3 + gateH[1] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[1] \\
strandO[1] + gateT[1] &\xrightarrow{\mathrm{q_{fmax1}}} out + b3 + m3
\end{aligned}
$$

$$
\begin{aligned}
u3 + m3 + out &\xrightarrow{\mathrm{k}} o3 + u3 \\
b2 + gateL[3] &\xrightarrow{\mathrm{k_{f1}}} gateH[3] + strandB[3] \\
gateH[3] + strandB[3] &\xrightarrow{\mathrm{q_{fmax1}}} b2 + gateL[3] \\
o2 + gateH[3] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[3] \\
strandO[3] + gateT[3] &\xrightarrow{\mathrm{q_{fmax1}}} out + b2 + m2
\end{aligned}
$$

$$
\begin{aligned}
u2 + m2 + out &\xrightarrow{\mathrm{k}} o2 + u2 \\
b1 + gateL[5] &\xrightarrow{\mathrm{k_{f1}}} gateH[5] + strandB[5] \\
gateH[5] + strandB[5] &\xrightarrow{\mathrm{q_{fmax1}}} b1 + gateL[5] \\
o1 + gateH[5] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[5] \\
strandO[5] + gateT[5] &\xrightarrow{\mathrm{q_{fmax1}}} out + b1 + m1
\end{aligned}
$$

$$
\begin{aligned}
u1 + m1 + out &\xrightarrow{\mathrm{k}} o1 + u1 \\
i1 + gateL[7] &\xrightarrow{\mathrm{k_{f1}}} gateH[7] + strandB[7] \\
gateH[7] + strandB[7] &\xrightarrow{\mathrm{q_{fmax1}}} i1 + gateL[7] \\
T1 + gateH[7] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[7] \\
strandO[7] + gateT[7] &\xrightarrow{\mathrm{q_{fmax1}}} W1
\end{aligned}
$$

$$
\begin{aligned}
i1 + gateL[8] &\xrightarrow{\mathrm{k_{f1}}} gateH[8] + strandB[8] \\
gateH[8] + strandB[8] &\xrightarrow{\mathrm{q_{fmax1}}} i1 + gateL[8] \\
u1 + gateH[8] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[8] \\
strandO[8] + gateT[8] &\xrightarrow{\mathrm{q_{fmax1}}} i1 + b1
\end{aligned}
$$

$$
\begin{aligned}
T1 + gateL[9] &\xrightarrow{\mathrm{k_{f1}}} gateH[9] + strandB[9] \\
gateH[9] + strandB[9] &\xrightarrow{\mathrm{q_{fmax1}}} T1 + gateL[9] \\
b1 + gateH[9] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[9] \\
strandO[9] + gateT[9] &\xrightarrow{\mathrm{q_{fmax1}}} T1 + u1
\end{aligned}
$$

$$
\begin{aligned}
b1 + i1 + T2 &\xrightarrow{\mathrm{k}} W2 + b1 \\
u1 + gateL[11] &\xrightarrow{\mathrm{k_{f1}}} gateH[11] + strandB[11] \\
gateH[11] + strandB[11] &\xrightarrow{\mathrm{q_{fmax1}}} u1 + gateL[11] \\
W2 + gateH[11] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[11] \\
strandO[11] + gateT[11] &\xrightarrow{\mathrm{q_{fmax1}}} T2 + u1 + i1
\end{aligned}
$$

$$
\begin{aligned}
b1 + i1 + u2 &\xrightarrow{\mathrm{k}} b2 + i1 + b1 \\
b1 + b2 + T2 &\xrightarrow{\mathrm{k}} T2 + u2 + b1 \\
u1 + W1 + Tp2 &\xrightarrow{\mathrm{k}} Wp2 + u1 \\
b1 + gateL[15] &\xrightarrow{\mathrm{k_{f1}}} gateH[15] + strandB[15] \\
gateH[15] + strandB[15] &\xrightarrow{\mathrm{q_{fmax1}}} b1 + gateL[15] \\
Wp2 + gateH[15] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[15] \\
strandO[15] + gateT[15] &\xrightarrow{\mathrm{q_{fmax1}}} Tp2 + b1 + W1 \\
\\
u1 + W1 + u2 &\xrightarrow{\mathrm{k}} b2 + W1 + u1 \\
u1 + b2 + Tp2 &\xrightarrow{\mathrm{k}} Tp2 + u2 + u1 \\
b1 + b2 + i1 + T3 &\xrightarrow{\mathrm{k}} W3 + b1 + b2 \\
u2 + gateL[19] &\xrightarrow{\mathrm{k_{f1}}} gateH[19] + strandB[19] \\
gateH[19] + strandB[19] &\xrightarrow{\mathrm{q_{fmax1}}} u2 + gateL[19] \\
W3 + gateH[19] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[19] \\
strandO[19] + gateT[19] &\xrightarrow{\mathrm{q_{fmax1}}} T3 + i1 + u2 \\
\\
u1 + gateL[20] &\xrightarrow{\mathrm{k_{f1}}} gateH[20] + strandB[20] \\
gateH[20] + strandB[20] &\xrightarrow{\mathrm{q_{fmax1}}} u1 + gateL[20] \\
W3 + gateH[20] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[20] \\
strandO[20] + gateT[20] &\xrightarrow{\mathrm{q_{fmax1}}} T3 + i1 + u1 \\
\\
b1 + b2 + i1 + u3 &\xrightarrow{\mathrm{k}} b3 + b1 + b2 + i1 \\
b1 + b2 + T3 + b3 &\xrightarrow{\mathrm{k}} u3 + b1 + b2 + T3 \\
b1 + u2 + W2 + Tp3 &\xrightarrow{\mathrm{k}} Wp3 + b1 + u2 \\
u1 + gateL[24] &\xrightarrow{\mathrm{k_{f1}}} gateH[24] + strandB[24] \\
gateH[24] + strandB[24] &\xrightarrow{\mathrm{q_{fmax1}}} u1 + gateL[24] \\
Wp3 + gateH[24] &\xrightarrow{\mathrm{q_{fmax1}}} strandO[24] \\
strandO[24] + gateT[24] &\xrightarrow{\mathrm{q_{fmax1}}} Tp3 + W2 + u1
\end{aligned}
$$

$$
\begin{aligned}
b2 + gateL[25] &\xrightarrow{k_{f1}} gateH[25] + strandB[25] \\
gateH[25] + strandB[25] &\xrightarrow{q_{fmax1}} b2 + gateL[25] \\
Wp3 + gateH[25] &\xrightarrow{q_{fmax1}} strandO[25] \\
strandO[25] + gateT[25] &\xrightarrow{q_{fmax1}} Tp3 + W2 + b2 \\
\\
b1 + u2 + W2 + u3 &\xrightarrow{k} b3 + b1 + u2 + W2 \\
b1 + u2 + Tp3 + b3 &\xrightarrow{k} u3 + b1 + u2 + Tp3 \\
u1 + b2 + W1 + Tpp3 &\xrightarrow{k} Wpp3 + u1 + b2 \\
b1 + gateL[29] &\xrightarrow{k_{f1}} gateH[29] + strandB[29] \\
gateH[29] + strandB[29] &\xrightarrow{q_{fmax1}} b1 + gateL[29] \\
Wpp3 + gateH[29] &\xrightarrow{q_{fmax1}} strandO[29] \\
strandO[29] + gateT[29] &\xrightarrow{q_{fmax1}} Tpp3 + W1 + b1 \\
\\
u2 + gateL[30] &\xrightarrow{k_{f1}} gateH[30] + strandB[30] \\
gateH[30] + strandB[30] &\xrightarrow{q_{fmax1}} u2 + gateL[30] \\
Wpp3 + gateH[30] &\xrightarrow{q_{fmax1}} strandO[30] \\
strandO[30] + gateT[30] &\xrightarrow{q_{fmax1}} Tpp3 + W1 + u2 \\
\\
u1 + b2 + W1 + u3 &\xrightarrow{k} b3 + u1 + b2 + W1 \\
u1 + b2 + Tpp3 + b3 &\xrightarrow{k} u3 + u1 + b2 + Tpp3 \\
u1 + u2 + Wp2 + Tppp3 &\xrightarrow{k} Wppp3 + u1 + u2 \\
b1 + gateL[34] &\xrightarrow{k_{f1}} gateH[34] + strandB[34] \\
gateH[34] + strandB[34] &\xrightarrow{q_{fmax1}} b1 + gateL[34] \\
Wppp3 + gateH[34] &\xrightarrow{q_{fmax1}} strandO[34] \\
strandO[34] + gateT[34] &\xrightarrow{q_{fmax1}} Tppp3 + Wp2 + b1 \\
\\
b2 + gateL[35] &\xrightarrow{k_{f1}} gateH[35] + strandB[35] \\
gateH[35] + strandB[35] &\xrightarrow{q_{fmax1}} b2 + gateL[35] \\
Wppp3 + gateH[35] &\xrightarrow{q_{fmax1}} strandO[35] \\
strandO[35] + gateT[35] &\xrightarrow{q_{fmax1}} Tppp3 + Wp2 + b2
\end{aligned}
$$

$$
\begin{aligned}
u1 + u2 + Wp2 + u3 &\xrightarrow{k} b3 + u1 + u2 + Wp2 \\
u1 + u2 + Tppp3 + b3 &\xrightarrow{k} u3 + u1 + u2 + Tppp3
\end{aligned}
$$

A.2.4 Markov Chain DNA

$$
\begin{aligned}
AV + gateL[1] &\xrightarrow{k} gateH[1] + strandB[1] \\
gateH[1] + strandB[1] &\xrightarrow{k} AV + gateL[1] \\
A1 + gateH[1] &\xrightarrow{k} strandO[1] \\
strandO[1] + gateT[1] &\xrightarrow{k} BV + A1 \\
\\
AV + gateL[2] &\xrightarrow{k} gateH[2] + strandB[2] \\
gateH[2] + strandB[2] &\xrightarrow{k} AV + gateL[2] \\
A2 + gateH[2] &\xrightarrow{k} strandO[2] \\
strandO[2] + gateT[2] &\xrightarrow{k} SV + A2 \\
\\
BV + gateL[3] &\xrightarrow{k} gateH[3] + strandB[3] \\
gateH[3] + strandB[3] &\xrightarrow{k} BV + gateL[3] \\
B1 + gateH[3] &\xrightarrow{k} strandO[3] \\
strandO[3] + gateT[3] &\xrightarrow{k} CV + B1 \\
\\
BV + gateL[4] &\xrightarrow{k} gateH[4] + strandB[4] \\
gateH[4] + strandB[4] &\xrightarrow{k} BV + gateL[4] \\
B2 + gateH[4] &\xrightarrow{k} strandO[4] \\
strandO[4] + gateT[4] &\xrightarrow{k} AV + B2 \\
\\
CV + gateL[5] &\xrightarrow{k} gateH[5] + strandB[5] \\
gateH[5] + strandB[5] &\xrightarrow{k} CV + gateL[5] \\
C1 + gateH[5] &\xrightarrow{k} strandO[5] \\
strandO[5] + gateT[5] &\xrightarrow{k} DV + C1
\end{aligned}
$$

$$
\begin{aligned}
CV + gateL[6] &\xrightarrow{k} gateH[6] + strandB[6] \\
gateH[6] + strandB[6] &\xrightarrow{k} CV + gateL[6] \\
C2 + gateH[6] &\xrightarrow{k} strandO[6] \\
strandO[6] + gateT[6] &\xrightarrow{k} BV + C2 \\
\\
DV + gateL[7] &\xrightarrow{k} gateH[7] + strandB[7] \\
gateH[7] + strandB[7] &\xrightarrow{k} DV + gateL[7] \\
D1 + gateH[7] &\xrightarrow{k} strandO[7] \\
strandO[7] + gateT[7] &\xrightarrow{k} EV + D1 \\
\\
DV + gateL[8] &\xrightarrow{k} gateH[8] + strandB[8] \\
gateH[8] + strandB[8] &\xrightarrow{k} DV + gateL[8] \\
D2 + gateH[8] &\xrightarrow{k} strandO[8] \\
strandO[8] + gateT[8] &\xrightarrow{k} CV + D2 \\
\\
EV + gateL[9] &\xrightarrow{k} gateH[9] + strandB[9] \\
gateH[9] + strandB[9] &\xrightarrow{k} EV + gateL[9] \\
E1 + gateH[9] &\xrightarrow{k} strandO[9] \\
strandO[9] + gateT[9] &\xrightarrow{k} FV + E1 \\
\\
EV + gateL[10] &\xrightarrow{k} gateH[10] + strandB[10] \\
gateH[10] + strandB[10] &\xrightarrow{k} EV + gateL[10] \\
E2 + gateH[10] &\xrightarrow{k} strandO[10] \\
strandO[10] + gateT[10] &\xrightarrow{k} DV + E2 \\
\\
FV + gateL[11] &\xrightarrow{k} gateH[11] + strandB[11] \\
gateH[11] + strandB[11] &\xrightarrow{k} FV + gateL[11] \\
F1 + gateH[11] &\xrightarrow{k} strandO[11] \\
strandO[11] + gateT[11] &\xrightarrow{k} GV + F1
\end{aligned}
$$

$$
\begin{aligned}
FV + gateL[12] &\xrightarrow{\text{k}} gateH[12] + strandB[12] \\
gateH[12] + strandB[12] &\xrightarrow{\text{k}} FV + gateL[12] \\
F2 + gateH[12] &\xrightarrow{\text{k}} strandO[12] \\
strandO[12] + gateT[12] &\xrightarrow{\text{k}} EV + F2 \\
\\
GV + gateL[13] &\xrightarrow{\text{k}} gateH[13] + strandB[13] \\
gateH[13] + strandB[13] &\xrightarrow{\text{k}} GV + gateL[13] \\
G1 + gateH[13] &\xrightarrow{\text{k}} strandO[13] \\
strandO[13] + gateT[13] &\xrightarrow{\text{k}} HV + G1 \\
\\
GV + gateL[14] &\xrightarrow{\text{k}} gateH[14] + strandB[14] \\
gateH[14] + strandB[14] &\xrightarrow{\text{k}} GV + gateL[14] \\
G2 + gateH[14] &\xrightarrow{\text{k}} strandO[14] \\
strandO[14] + gateT[14] &\xrightarrow{\text{k}} FV + G2 \\
\\
HV + gateL[15] &\xrightarrow{\text{k}} gateH[15] + strandB[15] \\
gateH[15] + strandB[15] &\xrightarrow{\text{k}} HV + gateL[15] \\
H1 + gateH[15] &\xrightarrow{\text{k}} strandO[15] \\
strandO[15] + gateT[15] &\xrightarrow{\text{k}} ENDV + H1 \\
\\
HV + gateL[16] &\xrightarrow{\text{k}} gateH[16] + strandB[16] \\
gateH[16] + strandB[16] &\xrightarrow{\text{k}} HV + gateL[16] \\
H2 + gateH[16] &\xrightarrow{\text{k}} strandO[16] \\
strandO[16] + gateT[16] &\xrightarrow{\text{k}} GV + H2)
\end{aligned}
$$

A.2.5 $y(x) = \frac{3}{4}x^2 - x + \frac{3}{4}$ DNA

$$
\begin{aligned}
X10 + gateL[1] &\xrightarrow{\text{q}_{\text{max}1}} gateH[1] + strandB[1] \\
gateH[1] + strandB[1] &\xrightarrow{\text{q}_{\text{max}}} X10 + gateL[1] \\
X00 + gateH[1] &\xrightarrow{\text{q}_{\text{max}}} strandO[1] \\
strandO[1] + gateT[1] &\xrightarrow{\text{q}_{\text{max}}} S0
\end{aligned}
$$

$$
\begin{aligned}
X10 + gateL[2] &\xrightarrow{q_{max1}} gateH[2] + strandB[2] \\
gateH[2] + strandB[2] &\xrightarrow{q_{max}} X10 + gateL[2] \\
X01 + gateH[2] &\xrightarrow{q_{max}} strandO[2] \\
strandO[2] + gateT[2] &\xrightarrow{q_{max}} S1 \\
\\
X11 + gateL[3] &\xrightarrow{q_{max1}} gateH[3] + strandB[3] \\
gateH[3] + strandB[3] &\xrightarrow{q_{max}} X11 + gateL[3] \\
X00 + gateH[3] &\xrightarrow{q_{max}} strandO[3] \\
strandO[3] + gateT[3] &\xrightarrow{q_{max}} S1 \\
\\
X11 + gateL[4] &\xrightarrow{q_{max1}} gateH[4] + strandB[4] \\
gateH[4] + strandB[4] &\xrightarrow{q_{max}} X11 + gateL[4] \\
X01 + gateH[4] &\xrightarrow{q_{max}} strandO[4] \\
strandO[4] + gateT[4] &\xrightarrow{q_{max}} S2 \\
\\
C00 + gateL[5] &\xrightarrow{q_{max1}} gateH[5] + strandB[5] \\
gateH[5] + strandB[5] &\xrightarrow{q_{max}} C00 + gateL[5] \\
S0 + gateH[5] &\xrightarrow{q_{max}} strandO[5] \\
strandO[5] + gateT[5] &\xrightarrow{q_{max}} Y0 \\
\\
C01 + gateL[6] &\xrightarrow{q_{max1}} gateH[6] + strandB[6] \\
gateH[6] + strandB[6] &\xrightarrow{q_{max}} C01 + gateL[6] \\
S0 + gateH[6] &\xrightarrow{q_{max}} strandO[6] \\
strandO[6] + gateT[6] &\xrightarrow{q_{max}} Y1 \\
\\
C10 + gateL[7] &\xrightarrow{q_{max1}} gateH[7] + strandB[7] \\
gateH[7] + strandB[7] &\xrightarrow{q_{max}} C10 + gateL[7] \\
S1 + gateH[7] &\xrightarrow{q_{max}} strandO[7] \\
strandO[7] + gateT[7] &\xrightarrow{q_{max}} Y0
\end{aligned}
$$

$$
\begin{aligned}
C11 + gateL[8] &\xrightarrow{q_{max1}} gateH[8] + strandB[8] \\
gateH[8] + strandB[8] &\xrightarrow{q_{max}} C11 + gateL[8] \\
S1 + gateH[8] &\xrightarrow{q_{max}} strandO[8] \\
strandO[8] + gateT[8] &\xrightarrow{q_{max}} Y1 \\
\\
C20 + gateL[9] &\xrightarrow{q_{max1}} gateH[9] + strandB[9] \\
gateH[9] + strandB[9] &\xrightarrow{q_{max}} C20 + gateL[9] \\
S2 + gateH[9] &\xrightarrow{q_{max}} strandO[9] \\
strandO[9] + gateT[9] &\xrightarrow{q_{max}} Y0 \\
\\
C21 + gateL[10] &\xrightarrow{q_{max1}} gateH[10] + strandB[10] \\
gateH[10] + strandB[10] &\xrightarrow{q_{max}} C21 + gateL[10] \\
S2 + gateH[10] &\xrightarrow{q_{max}} strandO[10] \\
strandO[10] + gateT[10] &\xrightarrow{q_{max}} Y1
\end{aligned}
$$

A.2.6 Function e^{-x} DNA

$$
\begin{aligned}
A10 + gateL[1] &\xrightarrow{k} gateH[1] + strandB[1] \\
gateH[1] + strandB[1] &\xrightarrow{q_{max}} A10 + gateL[1] \\
Ap10 + gateH[1] &\xrightarrow{q_{max}} strandO[1] \\
strandO[1] + gateT[1] &\xrightarrow{q_{max}} C11 + A10 + Ap10 \\
\\
A10 + gateL[2] &\xrightarrow{k} gateH[2] + strandB[2] \\
gateH[2] + strandB[2] &\xrightarrow{q_{max}} A10 + gateL[2] \\
Ap11 + gateH[2] &\xrightarrow{q_{max}} strandO[2] \\
strandO[2] + gateT[2] &\xrightarrow{q_{max}} C11 + A10 + Ap11 \\
\\
A11 + gateL[3] &\xrightarrow{k} gateH[3] + strandB[3] \\
gateH[3] + strandB[3] &\xrightarrow{q_{max}} A11 + gateL[3] \\
Ap10 + gateH[3] &\xrightarrow{q_{max}} strandO[3] \\
strandO[3] + gateT[3] &\xrightarrow{q_{max}} C11 + A11 + Ap10
\end{aligned}
$$

$$
\begin{aligned}
A11 + gateL[4] &\xrightarrow{\mathrm{k}} gateH[4] + strandB[4] \\
gateH[4] + strandB[4] &\xrightarrow{\mathrm{q_{max}}} A11 + gateL[4] \\
Ap11 + gateH[4] &\xrightarrow{\mathrm{q_{max}}} strandO[4] \\
strandO[4] + gateT[4] &\xrightarrow{\mathrm{q_{max}}} C10 + A11 + Ap11 \\
\\
C10 + gateG[5] &\xrightarrow{\mathrm{q_m}} strandO[5] \\
strandO[5] + gateT[5] &\xrightarrow{\mathrm{q_{max}}} nth \\
\\
C11 + gateG[6] &\xrightarrow{\mathrm{q_m}} strandO[6] \\
strandO[6] + gateT[6] &\xrightarrow{\mathrm{q_{max}}} nth \\
\\
A20 + gateL[7] &\xrightarrow{\mathrm{k}} gateH[7] + strandB[7] \\
gateH[7] + strandB[7] &\xrightarrow{\mathrm{q_{max}}} A20 + gateL[7] \\
C10 + gateH[7] &\xrightarrow{\mathrm{q_{max}}} strandO[7] \\
strandO[7] + gateT[7] &\xrightarrow{\mathrm{q_{max}}} C20 + A20 + C10 \\
\\
A20 + gateL[8] &\xrightarrow{\mathrm{k}} gateH[8] + strandB[8] \\
gateH[8] + strandB[8] &\xrightarrow{\mathrm{q_{max}}} A20 + gateL[8] \\
C11 + gateH[8] &\xrightarrow{\mathrm{q_{max}}} strandO[8] \\
strandO[8] + gateT[8] &\xrightarrow{\mathrm{q_{max}}} C20 + A20 + C11 \\
\\
A21 + gateL[9] &\xrightarrow{\mathrm{k}} gateH[9] + strandB[9] \\
gateH[9] + strandB[9] &\xrightarrow{\mathrm{q_{max}}} A21 + gateL[9] \\
C10 + gateH[9] &\xrightarrow{\mathrm{q_{max}}} strandO[9] \\
strandO[9] + gateT[9] &\xrightarrow{\mathrm{q_{max}}} C20 + A21 + C10 \\
\\
A21 + gateL[10] &\xrightarrow{\mathrm{k}} gateH[10] + strandB[10] \\
gateH[10] + strandB[10] &\xrightarrow{\mathrm{q_{max}}} A21 + gateL[10] \\
C11 + gateH[10] &\xrightarrow{\mathrm{q_{max}}} strandO[10] \\
strandO[10] + gateT[10] &\xrightarrow{\mathrm{q_{max}}} C21 + A21 + C11 \\
\\
C20 + gateG[11] &\xrightarrow{\mathrm{q_m}} strandO[11] \\
strandO[11] + gateT[11] &\xrightarrow{\mathrm{q_{max}}} nth
\end{aligned}
$$

$$
\begin{aligned}
C21 + gateG[12] &\xrightarrow{q_m} strandO[12] \\
strandO[12] + gateT[12] &\xrightarrow{q_{max}} nth \\
\\
A10 + gateL[13] &\xrightarrow{k} gateH[13] + strandB[13] \\
gateH[13] + strandB[13] &\xrightarrow{q_{max}} A10 + gateL[13] \\
C20 + gateH[13] &\xrightarrow{q_{max}} strandO[13] \\
strandO[13] + gateT[13] &\xrightarrow{q_{max}} C31 + A10 + C20 \\
\\
A10 + gateL[14] &\xrightarrow{k} gateH[14] + strandB[14] \\
gateH[14] + strandB[14] &\xrightarrow{q_{max}} A10 + gateL[14] \\
C21 + gateH[14] &\xrightarrow{q_{max}} strandO[14] \\
strandO[14] + gateT[14] &\xrightarrow{q_{max}} C31 + A10 + C21 \\
\\
A11 + gateL[15] &\xrightarrow{k} gateH[15] + strandB[15] \\
gateH[15] + strandB[15] &\xrightarrow{q_{max}} A11 + gateL[15] \\
C20 + gateH[15] &\xrightarrow{q_{max}} strandO[15] \\
strandO[15] + gateT[15] &\xrightarrow{q_{max}} C31 + A11 + C20 \\
\\
A11 + gateL[16] &\xrightarrow{k} gateH[16] + strandB[16] \\
gateH[16] + strandB[16] &\xrightarrow{q_{max}} A11 + gateL[16] \\
C21 + gateH[16] &\xrightarrow{q_{max}} strandO[16] \\
strandO[16] + gateT[16] &\xrightarrow{q_{max}} C30 + A11 + C21 \\
\\
C30 + gateG[17] &\xrightarrow{q_m} strandO[17] \\
strandO[17] + gateT[17] &\xrightarrow{q_{max}} nth \\
\\
C31 + gateG[18] &\xrightarrow{q_m} strandO[18] \\
strandO[18] + gateT[18] &\xrightarrow{q_{max}} nth \\
\\
A40 + gateL[19] &\xrightarrow{k} gateH[19] + strandB[19] \\
gateH[19] + strandB[19] &\xrightarrow{q_{max}} A40 + gateL[19] \\
C30 + gateH[19] &\xrightarrow{q_{max}} strandO[19] \\
strandO[19] + gateT[19] &\xrightarrow{q_{max}} C40 + A40 + C30
\end{aligned}
$$

$$
\begin{aligned}
A40 + gateL[20] &\xrightarrow{\mathrm{k}} gateH[20] + strandB[20] \\
gateH[20] + strandB[20] &\xrightarrow{\mathrm{q_{max}}} A40 + gateL[20] \\
C31 + gateH[20] &\xrightarrow{\mathrm{q_{max}}} strandO[20] \\
strandO[20] + gateT[20] &\xrightarrow{\mathrm{q_{max}}} C40 + A40 + C31
\end{aligned}
$$

$$
\begin{aligned}
A41 + gateL[21] &\xrightarrow{\mathrm{k}} gateH[21] + strandB[21] \\
gateH[21] + strandB[21] &\xrightarrow{\mathrm{q_{max}}} A41 + gateL[21] \\
C30 + gateH[21] &\xrightarrow{\mathrm{q_{max}}} strandO[21] \\
strandO[21] + gateT[21] &\xrightarrow{\mathrm{q_{max}}} C40 + A41 + C30
\end{aligned}
$$

$$
\begin{aligned}
A41 + gateL[22] &\xrightarrow{\mathrm{k}} gateH[22] + strandB[22] \\
gateH[22] + strandB[22] &\xrightarrow{\mathrm{q_{max}}} A41 + gateL[22] \\
C31 + gateH[22] &\xrightarrow{\mathrm{q_{max}}} strandO[22] \\
strandO[22] + gateT[22] &\xrightarrow{\mathrm{q_{max}}} C41 + A41 + C31
\end{aligned}
$$

$$
\begin{aligned}
C40 + gateG[23] &\xrightarrow{\mathrm{q_m}} strandO[23] \\
strandO[23] + gateT[23] &\xrightarrow{\mathrm{q_{max}}} nth
\end{aligned}
$$

$$
\begin{aligned}
C41 + gateG[24] &\xrightarrow{\mathrm{q_m}} strandO[24] \\
strandO[24] + gateT[24] &\xrightarrow{\mathrm{q_{max}}} nth
\end{aligned}
$$

$$
\begin{aligned}
A10 + gateL[25] &\xrightarrow{\mathrm{k}} gateH[25] + strandB[25] \\
gateH[25] + strandB[25] &\xrightarrow{\mathrm{q_{max}}} A10 + gateL[25] \\
C40 + gateH[25] &\xrightarrow{\mathrm{q_{max}}} strandO[25] \\
strandO[25] + gateT[25] &\xrightarrow{\mathrm{q_{max}}} C51 + A10 + C40
\end{aligned}
$$

$$
\begin{aligned}
A10 + gateL[26] &\xrightarrow{\mathrm{k}} gateH[26] + strandB[26] \\
gateH[26] + strandB[26] &\xrightarrow{\mathrm{q_{max}}} A10 + gateL[26] \\
C41 + gateH[26] &\xrightarrow{\mathrm{q_{max}}} strandO[26] \\
strandO[26] + gateT[26] &\xrightarrow{\mathrm{q_{max}}} C51 + A10 + C41
\end{aligned}
$$

$$
\begin{aligned}
A11 + gateL[27] &\xrightarrow{\mathrm{k}} gateH[27] + strandB[27] \\
gateH[27] + strandB[27] &\xrightarrow{\mathrm{q_{max}}} A11 + gateL[27] \\
C40 + gateH[27] &\xrightarrow{\mathrm{q_{max}}} strandO[27] \\
strandO[27] + gateT[27] &\xrightarrow{\mathrm{q_{max}}} C51 + A11 + C40 \\
\\
A11 + gateL[28] &\xrightarrow{\mathrm{k}} gateH[28] + strandB[28] \\
gateH[28] + strandB[28] &\xrightarrow{\mathrm{q_{max}}} A11 + gateL[28] \\
C41 + gateH[28] &\xrightarrow{\mathrm{q_{max}}} strandO[28] \\
strandO[28] + gateT[28] &\xrightarrow{\mathrm{q_{max}}} C50 + A11 + C41 \\
\\
C50 + gateG[29] &\xrightarrow{\mathrm{q_{m}}} strandO[29] \\
strandO[29] + gateT[29] &\xrightarrow{\mathrm{q_{max}}} nth \\
\\
C51 + gateG[30] &\xrightarrow{\mathrm{q_{m}}} strandO[30] \\
strandO[30] + gateT[30] &\xrightarrow{\mathrm{q_{max}}} nth \\
\\
A50 + gateL[31] &\xrightarrow{\mathrm{k}} gateH[31] + strandB[31] \\
gateH[31] + strandB[31] &\xrightarrow{\mathrm{q_{max}}} A50 + gateL[31] \\
C50 + gateH[31] &\xrightarrow{\mathrm{q_{max}}} strandO[31] \\
strandO[31] + gateT[31] &\xrightarrow{\mathrm{q_{max}}} C60 + A50 + C50 \\
\\
A50 + gateL[32] &\xrightarrow{\mathrm{k}} gateH[32] + strandB[32] \\
gateH[32] + strandB[32] &\xrightarrow{\mathrm{q_{max}}} A50 + gateL[32] \\
C51 + gateH[32] &\xrightarrow{\mathrm{q_{max}}} strandO[32] \\
strandO[32] + gateT[32] &\xrightarrow{\mathrm{q_{max}}} C60 + A50 + C51 \\
\\
A51 + gateL[33] &\xrightarrow{\mathrm{k}} gateH[33] + strandB[33] \\
gateH[33] + strandB[33] &\xrightarrow{\mathrm{q_{max}}} A51 + gateL[33] \\
C50 + gateH[33] &\xrightarrow{\mathrm{q_{max}}} strandO[33] \\
strandO[33] + gateT[33] &\xrightarrow{\mathrm{q_{max}}} C60 + A51 + C50
\end{aligned}
$$

$$
\begin{aligned}
A51 + gateL[34] &\xrightarrow{\mathrm{k}} gateH[34] + strandB[34] \\
gateH[34] + strandB[34] &\xrightarrow{\mathrm{q_{max}}} A51 + gateL[34] \\
C51 + gateH[34] &\xrightarrow{\mathrm{q_{max}}} strandO[34] \\
strandO[34] + gateT[34] &\xrightarrow{\mathrm{q_{max}}} C61 + A51 + C51 \\
\\
C60 + gateG[35] &\xrightarrow{\mathrm{q_m}} strandO[35] \\
strandO[35] + gateT[35] &\xrightarrow{\mathrm{q_{max}}} nth \\
\\
C61 + gateG[36] &\xrightarrow{\mathrm{q_m}} strandO[36] \\
strandO[36] + gateT[36] &\xrightarrow{\mathrm{q_{max}}} nth \\
\\
A10 + gateL[37] &\xrightarrow{\mathrm{k}} gateH[37] + strandB[37] \\
gateH[37] + strandB[37] &\xrightarrow{\mathrm{q_{max}}} A10 + gateL[37] \\
C60 + gateH[37] &\xrightarrow{\mathrm{q_{max}}} strandO[37] \\
strandO[37] + gateT[37] &\xrightarrow{\mathrm{q_{max}}} C71 + A10 + C60 \\
\\
A10 + gateL[38] &\xrightarrow{\mathrm{k}} gateH[38] + strandB[38] \\
gateH[38] + strandB[38] &\xrightarrow{\mathrm{q_{max}}} A10 + gateL[38] \\
C61 + gateH[38] &\xrightarrow{\mathrm{q_{max}}} strandO[38] \\
strandO[38] + gateT[38] &\xrightarrow{\mathrm{q_{max}}} C71 + A10 + C61 \\
\\
A11 + gateL[39] &\xrightarrow{\mathrm{k}} gateH[39] + strandB[39] \\
gateH[39] + strandB[39] &\xrightarrow{\mathrm{q_{max}}} A11 + gateL[39] \\
C60 + gateH[39] &\xrightarrow{\mathrm{q_{max}}} strandO[39] \\
strandO[39] + gateT[39] &\xrightarrow{\mathrm{q_{max}}} C71 + A11 + C60 \\
\\
A11 + gateL[40] &\xrightarrow{\mathrm{k}} gateH[40] + strandB[40] \\
gateH[40] + strandB[40] &\xrightarrow{\mathrm{q_{max}}} A11 + gateL[40] \\
C61 + gateH[40] &\xrightarrow{\mathrm{q_{max}}} strandO[40] \\
strandO[40] + gateT[40] &\xrightarrow{\mathrm{q_{max}}} C70 + A11 + C61 \\
\\
C70 + gateG[41] &\xrightarrow{\mathrm{q_m}} strandO[41] \\
strandO[41] + gateT[41] &\xrightarrow{\mathrm{q_{max}}} nth
\end{aligned}
$$

$$
\begin{aligned}
C71 + gateG[42] &\xrightarrow{q_m} strandO[42] \\
strandO[42] + gateT[42] &\xrightarrow{q_{max}} nth
\end{aligned}
$$

$$
\begin{aligned}
A10 + gateL[43] &\xrightarrow{k} gateH[43] + strandB[43] \\
gateH[43] + strandB[43] &\xrightarrow{q_{max}} A10 + gateL[43] \\
C70 + gateH[43] &\xrightarrow{q_{max}} strandO[43] \\
strandO[43] + gateT[43] &\xrightarrow{q_{max}} C81 + A10 + C70
\end{aligned}
$$

$$
\begin{aligned}
A10 + gateL[44] &\xrightarrow{k} gateH[44] + strandB[44] \\
gateH[44] + strandB[44] &\xrightarrow{q_{max}} A10 + gateL[44] \\
C71 + gateH[44] &\xrightarrow{q_{max}} strandO[44] \\
strandO[44] + gateT[44] &\xrightarrow{q_{max}} C81 + A10 + C71
\end{aligned}
$$

$$
\begin{aligned}
A11 + gateL[45] &\xrightarrow{k} gateH[45] + strandB[45] \\
gateH[45] + strandB[45] &\xrightarrow{q_{max}} A11 + gateL[45] \\
C70 + gateH[45] &\xrightarrow{q_{max}} strandO[45] \\
strandO[45] + gateT[45] &\xrightarrow{q_{max}} C81 + A11 + C70
\end{aligned}
$$

$$
\begin{aligned}
A11 + gateL[46] &\xrightarrow{k} gateH[46] + strandB[46] \\
gateH[46] + strandB[46] &\xrightarrow{q_{max}} A11 + gateL[46] \\
C71 + gateH[46] &\xrightarrow{q_{max}} strandO[46] \\
strandO[46] + gateT[46] &\xrightarrow{q_{max}} C80 + A11 + C71
\end{aligned}
$$