Computational Simulation of Gene Regulatory Networks Implementing an Extendable Synchronous Single-Input Delay Flip-Flop and State Machine

Imad Hoteit

A Thesis
In the Department
of
Electrical and Computer Engineering

Presented in Partial Fulfilment of the Requirements
For the Degree of Master of Science (Electrical and Computer Engineering) at
Concordia University
Montreal, Quebec, Canada

September 2011

© Imad Hoteit, 2011

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Imad Hoteit

Entitled: Computational Simulation of Gene Regulatory Networks Implementing an Extendable Synchronous Single-Input Delay Flip-Flop and State Machine.

and submitted in partial fulfilment of the requirements for the degree of

Masters of Applied Science

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________________ Chair
Dr. L. Lopes

______________________________________ Examiner
Dr. S. Gleason

______________________________________ Examiner
Dr. P. Grogono

______________________________________ Supervisor
Dr. N. Kharma

Approved by ______________________________________
Chair of Department or Graduate Program Director

____
Dr. Robin A. L. Drew, Dean
Faculty of Engineering and Computer Science

Date ______________________________________

ABSTRACT

Computational Simulation of a Gene Regulatory Network Implementing an Extendable Synchronous Single-Input Delay Flip-Flop

Imad Hoteit

We present a detailed and extendable design of the first *synchronous single-input* delay flip-flop implemented as a gene regulatory network in *Escherichia coli* (*E. coli*). The device, which we call the *BioD*, has one data input (trans-acting RNA), one clock input (far-red light) and an output that reports the state of the device using green fluorescent protein (GFP). The proposed design builds on Gardner's toggle switch, to provide a more sophisticated device that can be synchronized with other devices within or without the same cell, and which requires only one data input. We provide a mathematical model of the system and simulation results. The results show that the device behaves in line with desired functionality. Further, we discuss the constraints of the design, which pertain to ranges of parameter values. The *BioD* is extended via the addition of an update function and input and output interfaces. The result is the *BioFSM,* which constitutes a synchronous and modular finite state machine, which uses an update function to change its state, stored in the *BioD*. The *BioFSM* uses its input and output interfaces for inter-cellular communications. This opens the door to the design of a circular cellular automata (the *BioCell*), which is envisioned as a number of communicating *E. coli* colonies, each made of clones of one *BioFSM*.

ACKNOWLEDGEMENTS

I would like to start by acknowledging my parents Hassan and Chadia to whom this work is first and foremost dedicated. They have redefined the meaning of patience and endurance for me. They were there every step of the way, helping me, encouraging me, pushing me to achieve what they said was “the least of my capabilities”. Thank you for loving me the way you do and for your endless pride, even when it was undeserved. Huge thanks to my sisters Ola and Rasha who cared for me and comforted me in times of hardship and distress. Thank you for your endless understanding, I love you.

To Michelle: My home away from home. For so many years, you’ve been the rock that I stood upon, the only one I relied on, my real guide and confidant, my best friend, my sister, my teacher. I thank you more than I can say.

To Dr. Ahmad: I have never needed help and not found myself better off for talking to you. Your objectivity and poised nature, your ambition and strive, and the example you so well present with this life so well lived have been something to behold from the day that I met you. Since then you’ve become much more than I could ever bear to lose. Thank you for all your guidance and opinions without which this thesis would have greatly suffered.

To Jalal, Ashraf and Makram: You've sheltered me. Each in his own way. Took care of me when I was lost. Accepted my many flaws lightly. You are the best definition of a friend I could come up with, ever. I've learned much from you. Much more than you are aware. I rely on you, and I love you for it. Thank you.

To Ben: You are more than a friend to me. Though I didn't notice it at first, you became a mentor to me very quickly when we met. I find in you a better version of myself and I love you for it. Your support for this thesis was never direct, but ever-present. I am better for knowing you. A bright guiding light you are. Mein älterer Bruder.

To Ramzi: You are the younger brother I never knew I had. You are such a wonderful person that I relied on you and on your help much more than I should sometimes. Thank you cousin for being in my life. For all you've done for me, I say I love and thank you from all my heart; and for all that I see you capable of accomplishing, I say you truly have a wonderful life ahead of you that I can't wait to share in.

To Jad, Anthony and Nadim: Throughout this thesis, you've bore the brunt of my busy schedule, and yet remained great friends. I always could count on your support and I hope the next journey will make me a much better friend.

To Hassan: Well my friend, the day has come. Since our teenage years we’ve shared our lives. Now I am married and you have a son. All that was missing was a thesis… Thank you for your constant guidance and support over the years. I love you and miss you.

To Mo, Ale and Mosha: my labmates and friends. We’ve shared much together. Thank you for your support, the unending cosmic conversations, and the opinions you’ve provided over the years. You are all amazing people and I am better for knowing you.

I would also like to thank Dr. Eusebius Doedel for the invaluable help understanding non-linear systems and the manner in which to analyse them. Many thanks go to Dr. Luc Varin. You have been my main source of biological knowledge. This thesis rests upon the knowledge I acquired from you in and outside the classroom.

I would also like to thank my committee, Dr. Scott Gleason and Dr. Peter Grogono for their insightful comments and corrections.

To my wonderful partner, whom I JUST married, Nathalie: If acknowledging you are the love of my life and my light in the dark is generally seen as enough, it isn’t the case here. You helped me write this thesis. From the introduction, to the literature review, to an endless number of images and more, you’ve been the endless support I never deserved. I tried to finish before you came in my life, but my life took a better turn only after meeting you. It’s as if this thesis was waiting for you to be finalized. But that is not all. You've sacrificed so much for me, some many hours, so many opportunities, so much energy. I have put so much weight on you and I promise you that it ends now. I love you and I promise you the life of a queen. "You to me are everything, the sweetest song that I could sing…", and you are my true rock, my goal, my confidant, my love and my soul. I finished because of you and because I love you.

Last but not least, to my supervisor, Dr. Nawwaf Kharma. Words cannot express how much I owe you. More than anyone, I owe you the rest of my life since I would have thrown it away without your help, and your love. You've been a true friend but also a father figure, and I can only explain the patience you've shown me over the years with this. You are a wonderful WONDERFUL human being, and I am lucky to have been your student. I am sorry for the stress I caused and I thank you from the bottom of my heart.

Table of Contents

List of Figures

List of Tables

List of Acronyms

GRN:	Gene Regulatory Network
FR:	Far-Red
RNA:	RiboNucleic Acid
DNA:	DeoxyriboNucleic Acid
RNAP:	RNA Polymerase

CHAPTER 1.

INTRODUCTION

Known as the "building blocks of life", the cells of an organism are identical in the genetic information they hold but can be quite different in their shape and in the functionality they provide. They are in fact the smallest unit of life, and can either separately be an independent organism or, they can be a small part of an intricate whole composed of different types of cells which is called a multi-cellular organism.

1.1 Cell and Genome

Cells are the structural and functional units of all living organisms. Each cell can take in nutrients, convert these nutrients into energy, carry out specialized functions, and reproduce as necessary. Furthermore, each cell stores its own set of instructions for carrying out each of these activities.

There are two general categories of cells: *prokaryotes* and *eukaryotes*. Bacteria are the best known and most studied form of prokaryotic organisms. *Prokaryotes* are unicellular organisms and are distinguished from eukaryotes on the basis of nuclear organization, specifically their lack of a nuclear membrane. Prokaryotes also lack any of the intracellular organelles and structures that are characteristic of eukaryotic cells.

Eukaryotes include fungi, animals, and plants as well as some unicellular organisms. Eukaryotic cells contain a nucleus, a membrane-delineated compartment that houses the eukaryotic cell's *DNA*[1]. Eukaryotic organisms also have other specialized structures, called organelles, which are small structures within cells that perform dedicated functions. For a descriptive listing of eukaryotic organelles, the reader is referred to (Mullock and Luzio, 2005).

It is worth noting that eukaryotes use the same genetic code and metabolic processes as prokaryotes. Whether they come from the same organism or belong to different ones, of similar type or unrelated, prokaryotes or eukaryotes, all cells hold within them the genome of the organism. The *genome*[2] guides and drives the behaviour and functionality of the cell. It is the entire set of hereditary

[1] A nucleic acid that carries the genetic information in the cell and is capable of self-replication and synthesis of RNA. The abbreviation stands for *deoxyribonucleic acid*.

[2] The full complement of genetic material within an organism.

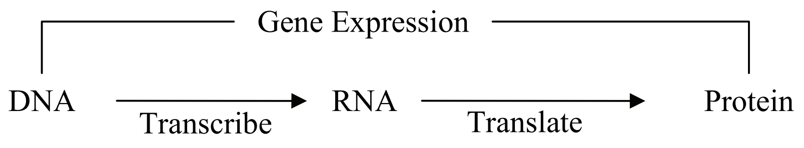
instructions for building, running, and maintaining an organism, and passing life on to the next generation.

The information in the genome is organized in logical sub-units. In a few words, the genome is divided into chromosomes, chromosomes contain genes, and genes are made of DNA. A chromosome is made of DNA and protein. It is a package containing some of an organism's genes. Chromosomes help a cell to keep a large amount of genetic information neat, organized, and compact as well as guide the separation and flow of genetic material during cell division (Hartwell et al., 2010). Genes are found on chromosomes and are made of DNA. Different genes determine the different characteristics, or traits, of an organism. One gene might determine the colour of a bird's feathers, while another gene would determine the shape of its beak. Most genes however, code for much more specialized functionality; a protein to catalyze a reaction, the production of a required substance or even a protein needed for the regulation of another protein. Regulatory proteins are discussed in more detail a couple of sections ahead.

1.2 Genome and Regulation

A gene regulatory network or GRN is a set of genes which interact with each other and with other substances in the cell, thereby governing the rates at which genes in the network are transcribed into mRNAs or/and the rate at which mRNAs is translated into proteins.

The genome of a cell (or its DNA) holds most of the information needed for a cell to function. That genetic material contains blocks of information called genes which encode specific proteins that determine function and behaviour. They do so by producing specific proteins and by releasing them in the cellular cytoplasm. Found on the DNA, a gene needs to be *transcribed* into an mRNA strand and, before it can be *translated* into a protein. Each mRNA has, in addition to the open reading frame or ORF (the region encoding the amino acid sequence making up the protein), a region called Ribosome Binding Site (RBS) to which the Ribosome binds before starting the process of translation of the ORF into the corresponding protein.



Gene regulation adds levels of control to this process. The DNA strands are not always transcribed without regulation (i.e. constitutively) but are almost always controlled by other molecules such as activators and repressors. These work by binding to specific sequences on the DNA called operators, and hence affecting the process of transcription of the mRNA; other molecules bind to specific sites on the mRNA and hence influence the process of translation

1.2.1 Operon Structure

A typical operon has three distinct parts or regions. The *promoter* region is recognized by RNA Polymerase (RNAP), thus allowing the initiation of transcription. The *operator* region serves as a stage for repression or activation of transcription. The *structural genes* region contains the genes that are co-regulated by the same promoter.

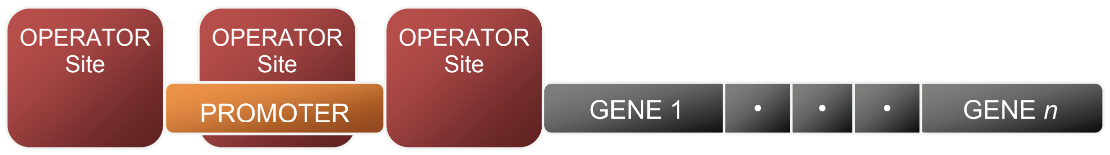


Figure 1-1. Operon structure. Operator sites can be located before, inside or after the promoter. More than one operator site can be occupied simultaneously.

The expression of the genes of the operon is regulated by the repressors and activators acting at and around the promoter region. The following section describes the four types of regulation encountered.

1.2.2 Types of Regulation

There are many levels of regulation of gene expression. We highlight four main categories:

Transcriptional regulation is the change in gene expression levels by altering transcription rates *i.e.* controlling the production of mRNA mainly using transcription factors (explained below).

Post-transcriptional regulation is the control of gene expression at the RNA level, therefore between the transcription and the translation of the gene. The main tool in this category is RNA interference (RNAi).

Translational regulation refers to the control of the levels of protein synthesized from its mRNA. The corresponding mechanisms are primarily targeted on the control of ribosome recruitment on the initiation codon, but can also involve modulation of the elongation or termination of protein synthesis. In most cases, translational regulation involves specific RNA secondary structures on the mRNA.

Post-translational regulation refers to the control of the levels of active protein. There are several forms. It is performed either by means of reversible events (Post-translational modifications, such as Phosphorylation or sequestration) or by means of irreversible events (proteolysis).

1.2.3 Transcriptional Regulation

Since the overwhelming type of regulation used in this thesis is transcriptional, that regulation is discussed in more detail, and we classify it as follows. Transcriptional regulation of genes and operons is categorized into four different modes: negative inducible, negative repressible, positive inducible and positive repressible.

Negative Regulation

Negative regulation occurs in operons whose operator sites bind a repressor protein. A repressor protein typically denies RNAP from binding and initiating transcription of the genes on the operon.

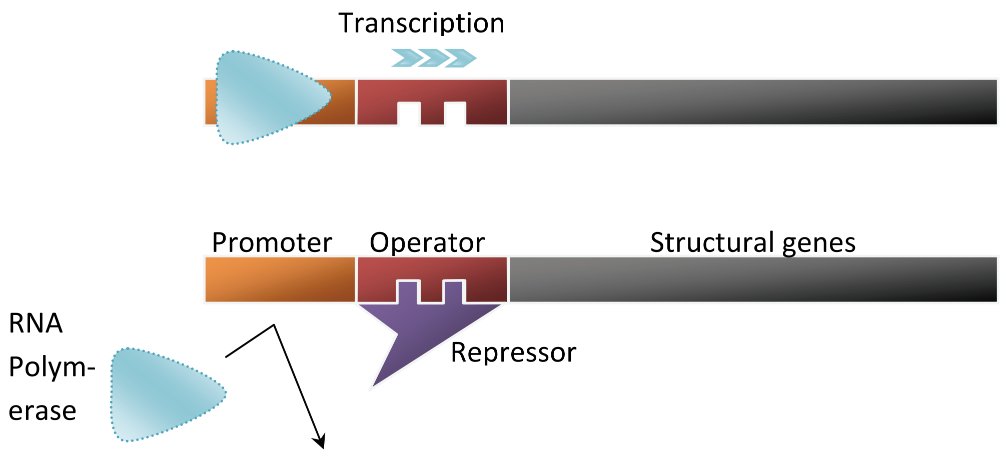


Figure 1-2. Simple negative regulation.
No transcription occurs in the presence of the repressor.

a. *Negative Inducible Operons*

In these operons, the repressor protein is normally bound to the operator site and prevents transcription. However, if an inducer molecule is introduced, it binds to the repressor protein. This binding changes the latter's configuration, so it can no longer bind to the operator, thus *inducing* transcription.

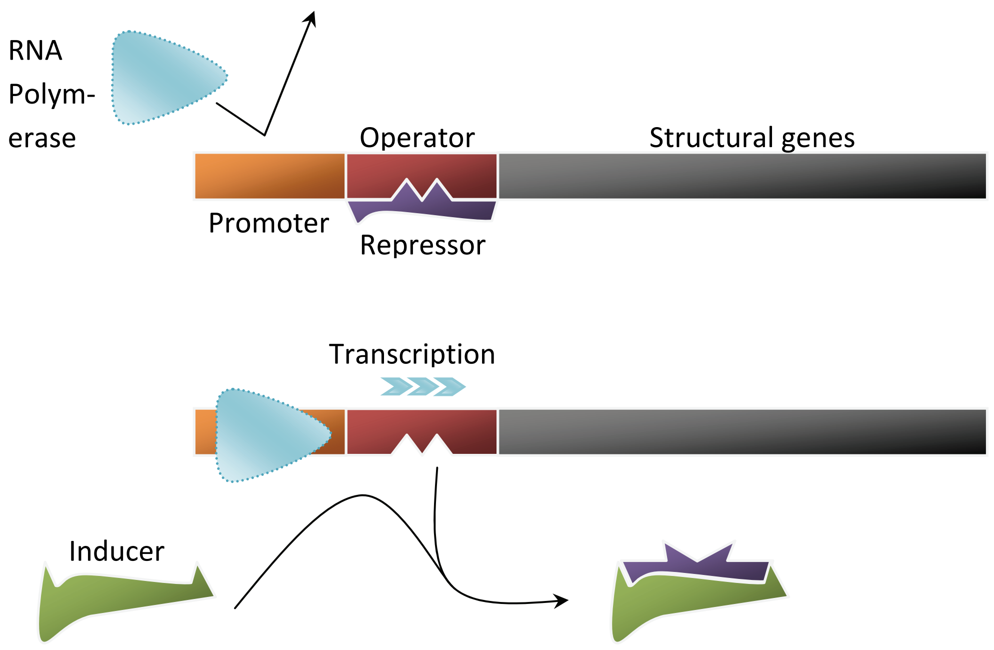


Figure 1-3. Negative inducible regulation.
Transcription is OFF in the presence of the repressor. The inducer causes a conformational change in the repressor protein, preventing repression, thus inducing transcription.

b. Negative Repressible Operons

In these operons, transcription normally takes place. The repressor protein cannot bind to the operator site in its normal configuration. However, with the introduction of a certain molecule called a co-repressor, which binds to the repressor protein, the configuration can be changed such that it can bind to the operator site and *repress* transcription.

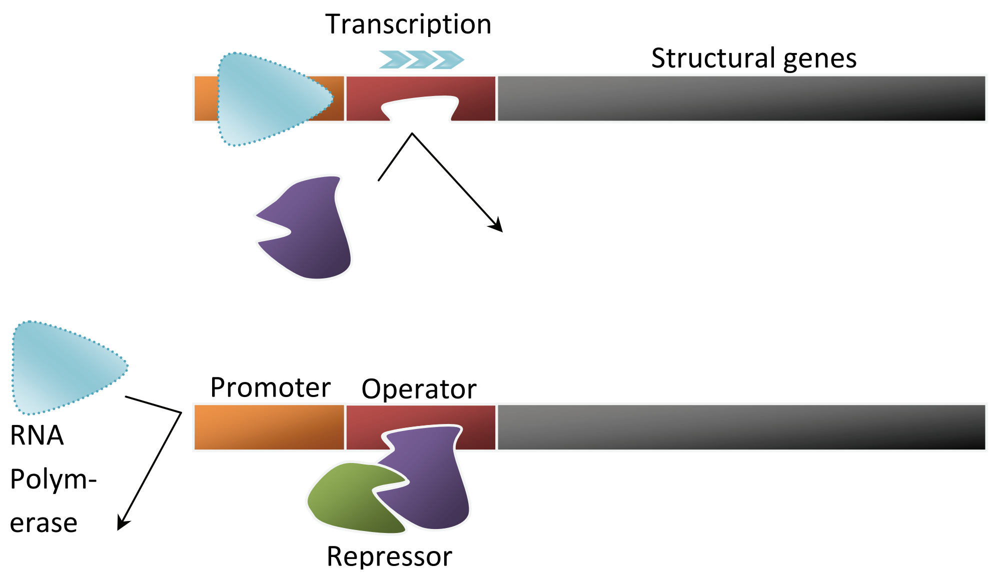


Figure 1-4. Negative Repressible regulation.
Transcription is ON in the presence of the repressor. The introduction of another repressor, which binds to the first and causes a conformational change, allows for repression to occur and transcription is stopped.

Positive Regulation

Positive regulation occurs in operons whose DNA binds an activator protein[3]. Activator proteins either induce or stimulate transcription. An operon that binds an activator protein can vastly increase production (more than a thousand fold).

[3] Activator proteins usually bind at a site other than the operator. For simplicity however, the figures in Table 1 do not display that difference.

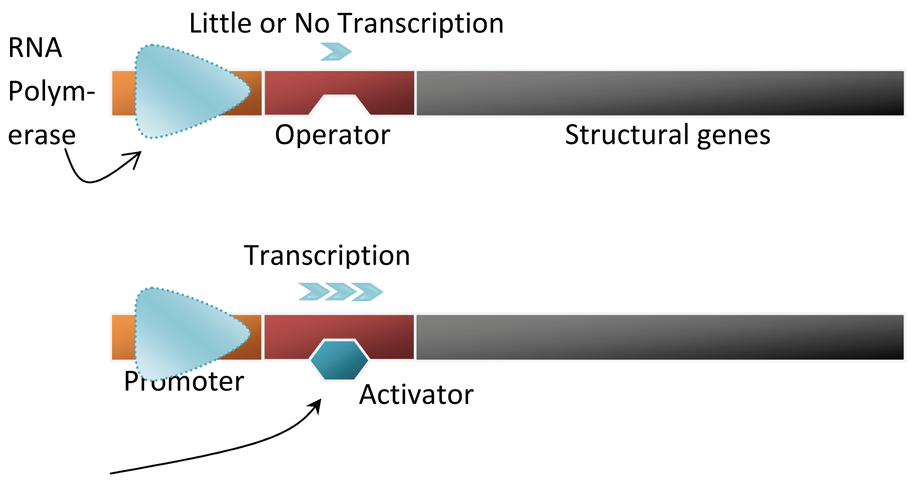


Figure 1-5. Simple positive regulation.
Transcription is significantly boosted in the presence of the activator.

a. Positive Inducible Operons

In these operons, the activator protein cannot bind to the DNA in its normal configuration. However, with the introduction of an inducer molecule, which binds to the activator protein, the configuration can be changed such that it can bind to the operator site and activates or stimulates transcription.

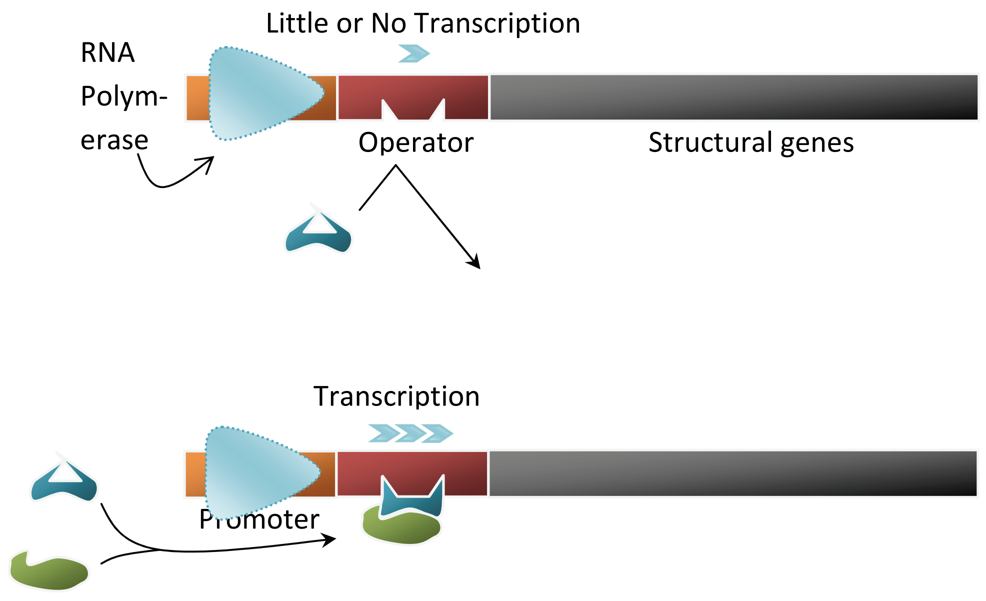


Figure 1-6. Positive inducible regulation.
Transcription is OFF in the presence of the activator. The introduction of the inducer, which causes a conformational change in the activator, allows for activation to occur and transcription is induced.

b. Positive Repressible Operons

In these operons, the activator protein is normally bound to the operator site. However, if a co-repressor molecule is introduced, it binds to the activator protein. This binding changes the latter’s configuration, so it can no longer bind to the DNA, thus stopping transcription.

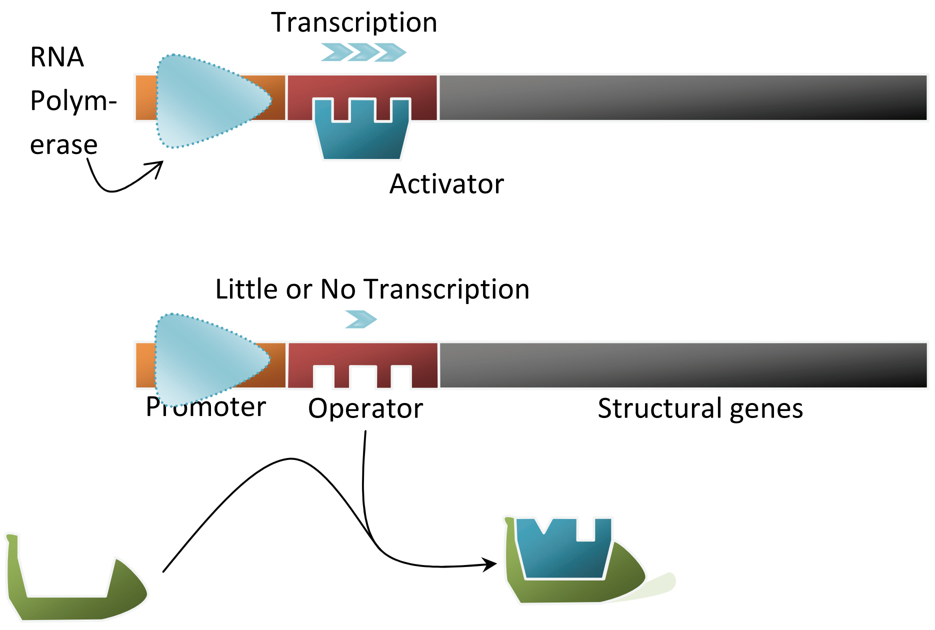


Figure 1-7. Positive repressible regulation.
Transcription is ON in the presence of the activator. The repressor causes a conformational change in the activator protein, preventing binding, thus repressing transcription.

1.3 Need for Simulation

When designing new systems, or deciding which route to take in a project, even when testing for failure points, simulation of any kind has always been a desirable tool to making sure we build what we intended to build. There are various reasons for this not the least of which are cost effectiveness, speed, and the inability to test in real time.

Simulation is also an appropriate proof of concept. An "analytic" model is appropriate when mathematics can be used to find the exact (deterministic) or probable (stochastic) values of the measures of performance.

In the case of GRNs, many simulation methods exist that are deterministic or stochastic, discrete or continuous, static or dynamic, and qualitative or quantitative.

The following section discusses the simulations methods we applied in the thesis. We used ordinary differential equations (ODEs) to generate two types of simulations that were deterministic and stochastic, while being discrete dynamic and quantitative.

1.4 Hill Equation

Please note that from this moment on, the promoter is taken to mean both the promoter and operator regions of an operon. The reason is because the operator sites can be found either before the promoter site, or after it (or much after it) and sometimes in it (between the -10 and -35 sites). Thus both sites are in effect the same region and we understand that any "regulated" promoter has to have the required operator sites for said regulation.

The most direct way of modelling the changes in concentrations of substances in a network is using the Hill equation. Consider the case in Figure 1-8 of an unregulated (or constitutively expressed) gene.

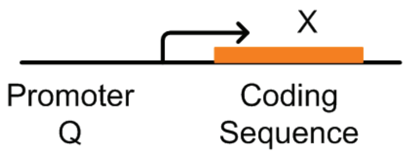


Figure 1-8. Unregulated operon.
DNA strand that notably contains a promoter and the coding sequence of a protein, thus forming a constitutively expressed gene.

The promoter region of this gene is not sensitive to any stimulus, negative or positive, and is therefore continually working. RNAP attaches itself unobstructed to its binding sites in the promoter and proceeds to transcribe this gene. This is followed by another RNAP molecule and so on and so forth, only limited by (i) the strength of the promoter and by (ii) the availability of building blocks.

The strength of a promoter is defined as the strength of the binding that occurs with the RNAP. The stronger the binding, the more likely it is to occur, and the less likely it is to dissociate once occurred. Each promoter (Q) has a specific strength which we model as the transcription rate ρ_Q. Thus a constitutively expressed gene X is transcribed according to

$$\frac{d[mX]}{dt} = \underbrace{\rho_Q}_{\substack{\text{max.}\\ \text{transcription}}} \tag{1.1}$$

where $[mX]$ is the concentration of the mRNA transcripts of gene X.

Since mRNA transcripts have a limited half life, their total concentration is better approximated in equation (1.2) where a degradation term is added that is proportional to the concentration of existing transcripts.

$$\frac{d[mX]}{dt} = \underbrace{\rho_Q}_{\substack{\text{max.}\\ \text{transcription}}} - \underbrace{\omega_{mX}[mX]}_{\text{degradation}} \tag{1.2}$$

This equation is read as: the change in the concentration gene X mRNA transcripts per unit of time is equal to the transcription rate of the promoter Q, minus the degradation rate of the existing transcripts.

The most common regulation comes in the form of a repressor which inhibits transcription by binding to its operator site and prevents RNAP from binding to the promoter and begin transcription (see Figure 1-9).

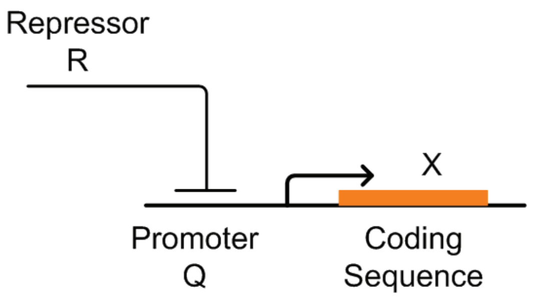


Figure 1-9. Negative regulation.
The repressor R binds to its operator site and prevents RNAP from binding to the promoter, effectively inhibiting transcription.

The transcription rate ρ_Q is now influenced by $[R]$, the concentration of the repressor R in the system. This dynamic is modelled as follows

$$\frac{d[mX]}{dt} = \underbrace{\rho_Q}_{\substack{\text{max.}\\ \text{transcription}}} \cdot \underbrace{\left(\frac{1}{1+\left(\frac{[R]}{K_R}\right)^{n_R}}\right)}_{\text{inhibition from R}} - \underbrace{\omega_{mX}[mX]}_{\text{degradation}} \tag{1.3}$$

where K_R and n_R are the dissociation constant of R and the Hill cooperativity coefficient of R respectively.

The dissociation constant generally termed K_d, measures the strength of the binding of the repressor to the operator. This is described in further detail in section 4.1.5 below, and its effect on the shape of the curve is displayed in Figure 1-10.

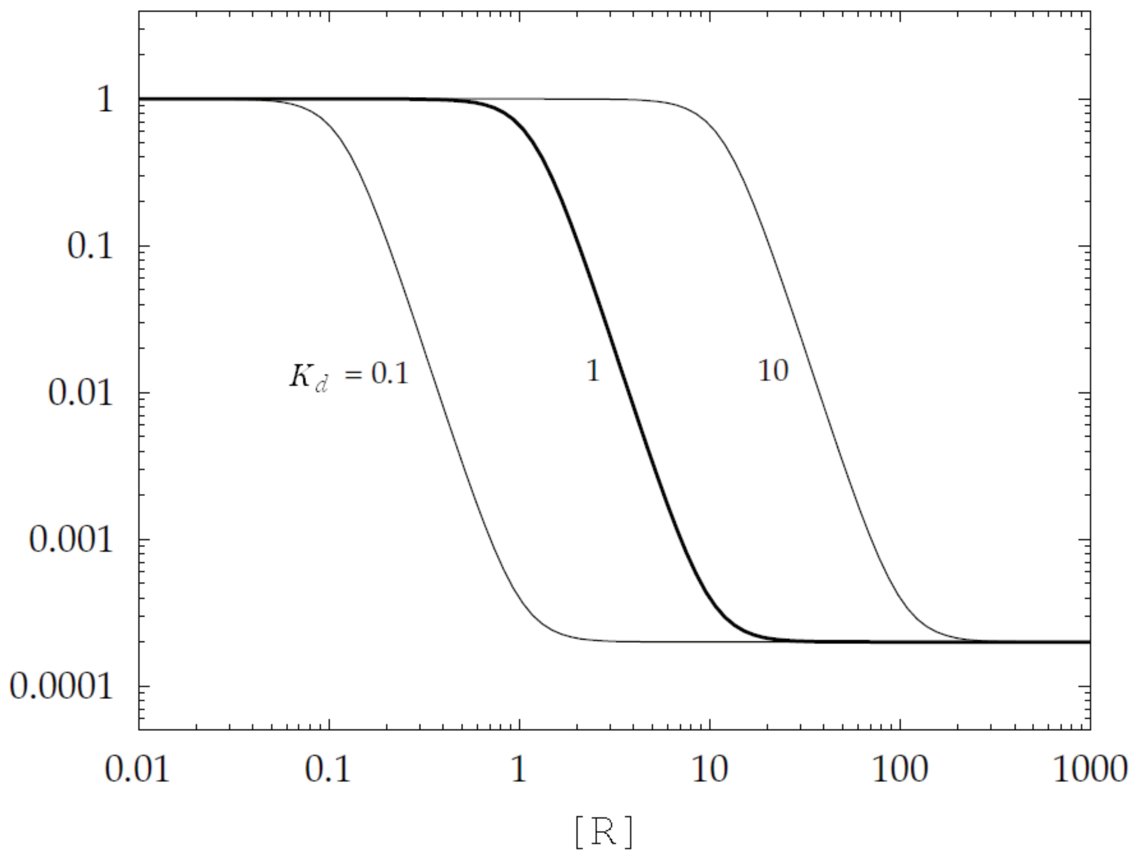


Figure 1-10. Effect of the dissociation constant
on the speed of the inhibition.

The Hill cooperativity coefficient here reflects the manner (positive, negative or non-cooperative) in which the repressor is binding to the operator site on the DNA strand. In biochemistry, complex molecules and multimers often are assembled using their binding sites one block at a time. That assembly, or distributed process of binding can be enhanced or inhibited after the first (or later) binding(s). This is known as cooperative binding and the Hill coefficient provides a way to quantify that effect. When the binding is enhanced and we have a positively cooperative reaction, $n_d > 1$; when the binding is inhibited and we have a negatively cooperative reaction, $n_d < 1$; when the binding is unaffected and we have a non-cooperative reaction, $n_d = 1$. Its effect on the sigmoid is illustrated in Figure 1-11.

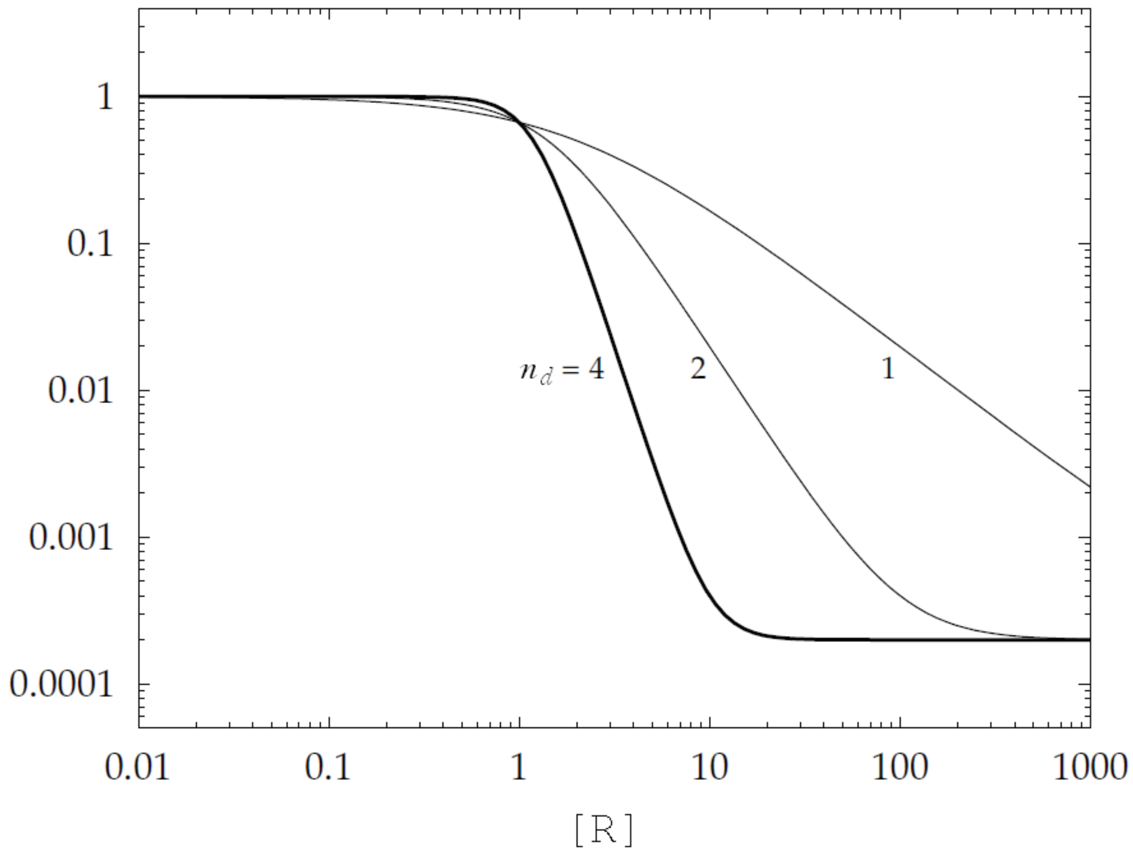


Figure 1-11. Effect of the Hill coefficient
on the slope of the sigmoidal curve created by the Hill equation.

Biological systems are inherently imprecise, and gene regulation is no different. Indeed, for realistic simulations, the designs must incorporate the notion of *leakage* in the equation. That is, when a repressor R is present in a high enough concentration to completely inhibit the transcription of gene X, that transcription is almost never shutdown completely. Rather it is brought down to a very low but still present basal level due to the leakage of the repression. We modify equation (1.3) to incorporate a leakage term as follows

$$\frac{d[mX]}{dt} = \underbrace{\rho_Q}_{\substack{\text{max.}\\ \text{transcription}}} \cdot \left(\underbrace{a + (1-a)}_{\text{leakage}} \cdot \underbrace{\left(\frac{1}{1 + \left(\frac{[R]}{K_R}\right)^{n_R}} \right)}_{\text{inhibition from R}} \right) - \underbrace{\omega_{mX}[mX]}_{\text{degradation}} \tag{1.4}$$

where $0 \le a \le 1$ represents the leakage percentage, typically (but not necessarily) 1%.

It is possible to have more than one repressor regulate the same gene. The promoter region can be designed to have more than one operator binding sites for different repressors. Figure 1-12 displays the common case of dual repression. Two different repressors $R1$ and $R2$, bind to their respective operator sites in promoter Q and inhibit transcription of gene X.

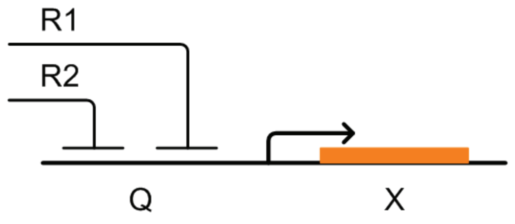


Figure 1-12. Dual repression.

The above configuration yields the strongest transcription in the absence of *R1* and *R2* and the strongest repression in their dual presence. The presence of just one of the repressors yields a weaker repression. This dynamic is modelled as follows

$$\frac{d[mX]}{dt} = \underbrace{\rho_Q}_{\substack{\text{max.}\\ \text{transcription}}} \cdot \left(\underbrace{a + (1-a)}_{\text{leakage}} \cdot \underbrace{\left(\frac{1}{1 + \left(\frac{[R1]}{K_{R1}}\right)^{n_{R1}}} \right)}_{\text{inhibition from R1}} \cdot \underbrace{\left(\frac{1}{1 + \left(\frac{[R2]}{K_{R2}}\right)^{n_{R2}}} \right)}_{\text{inhibition from R2}} \right) - \underbrace{\omega_{mX}[mX]}_{\text{degradation}} \quad (1.5)$$

The last case we will see used in this thesis is the activation and repression by two different substances of the same transcription process.

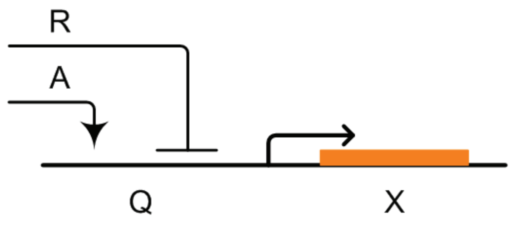


Figure 1-13. Activation and repression.

The activation term in the equation below is the inverse form of the Hill equation. At low concentrations of the activator *A*, the transcription is reduced to basal level, while an increase in the concentration of *A* increases the transcription rate.

$$\frac{d[mX]}{dt} = \underbrace{\rho_Q}_{\substack{\text{max.}\\ \text{transcription}}} \cdot \left(\underbrace{a + (1-a)}_{\text{leakage}} \cdot \underbrace{\left(\frac{1}{1 + \left(\frac{[R]}{K_R}\right)^{n_R}} \right)}_{\text{inhibition from R}} \cdot \underbrace{\left(\frac{\left(\frac{[A]}{K_A}\right)^{n_A}}{1 + \left(\frac{[A]}{K_A}\right)^{n_A}} \right)}_{\text{activation from A}} \right) - \underbrace{\omega_{mX}[mX]}_{\text{degradation}} \tag{1.6}$$

1.5 Law of Mass Action

The law of mass action is a mathematical model that explains and predicts behaviours of solutions in dynamic equilibrium. It can be described with two aspects: 1) the equilibrium aspect, concerning the composition of a reaction mixture at equilibrium and 2) the kinetic aspect concerning the rate equations for elementary reactions.

The law states that the rate of an elementary reaction is proportional to the product of the concentrations of the participating molecules.

$$\text{Receptor} + \text{Ligand} \underset{K_{OFF}}{\overset{K_{ON}}{\rightleftarrows}} \text{Receptor} \bullet \text{Ligand} \tag{1.7}$$

Equilibrium is reached when the rate at which new ligand•receptor complexes are formed equals the rate at which the ligand•receptor complexes dissociate. At equilibrium:

$$[\text{Ligand}].[\text{Receptor}].K_{ON} = [\text{Ligand} \bullet \text{Receptor}].K_{OFF} \qquad (1.8)$$

That derivation is used to generate equation (3.15) below.

1.6 Michaelis-Menten Kinetics

Enzymes are molecules that act as catalysts to a reaction. Enzymatic reactions abound in a cell and are very different from the transcript generating process described above; hence the equation follows a different mathematical model.

An enzyme E helps turn a substrate S into a product P but is not consumed by the process. Rather, the enzyme binds (in a reversible process) to the substrate forming a complex ES which in turn is converted into a product P and the enzyme.

$$E + S \underset{k_r}{\overset{k_f}{\rightleftarrows}} ES \xrightarrow{k_{cat}} E + P \qquad (1.9)$$

where k_f, k_r and k_{cat} denote the forward, reverse and catalysed reaction rate constants respectively.

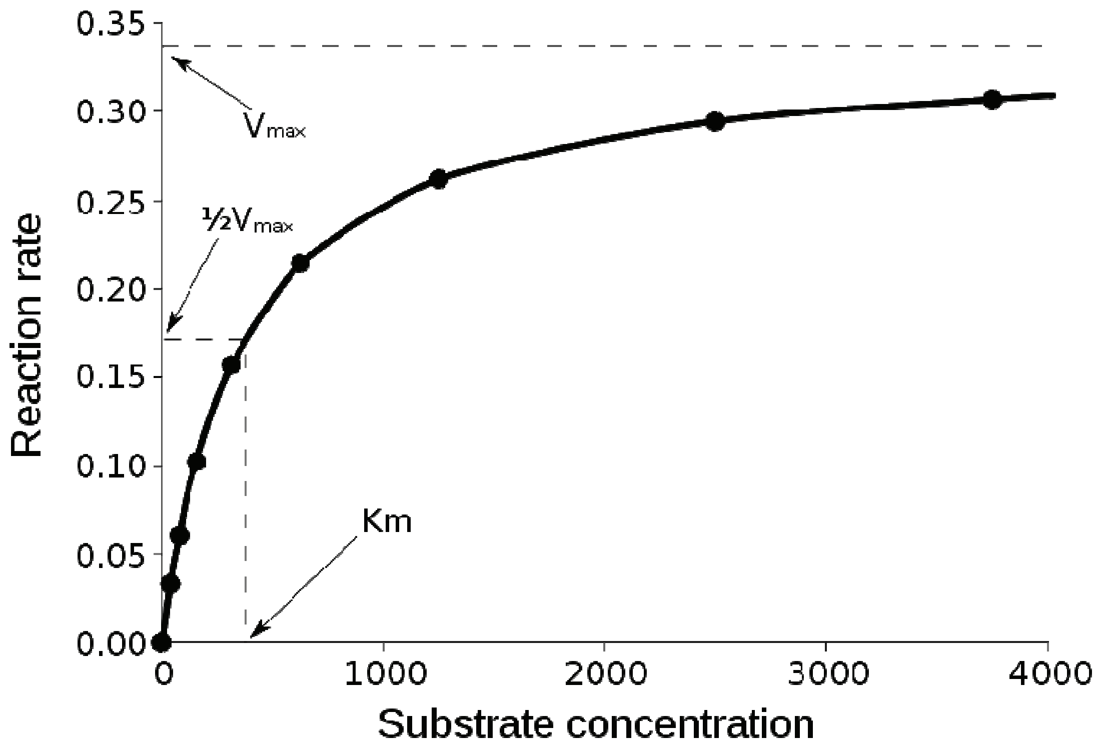


Figure 1-14. Michaelis-Menten Kinetics[4]

Saturation curve for an enzyme (substrate concentration vs reaction rate).

The reaction rate is given with respect to the concentration v_0 of a substrate S. The formula is given by

$$v_0 = \frac{V_{\max}[S]}{K_M + [S]} \tag{1.10}$$

where $V_{\max}$ is the maximum rate achieved by the system, i.e. during saturated substrate concentrations. K_M is the Michaelis constant, and is defined as the substrate concentration at which the reaction rate is half of $V_{\max}$.

[4] Image taken from Wikipedia: "http://en.wikipedia.org/wiki/Enzyme_kinetics"

CHAPTER 2.

LITERATURE REVIEW

Most of the complex processes that take place in a cell are governed by gene expression, which is regulated at several levels along the pathway leading from DNA to protein. Gene expression may be regulated during transcription and post-transcriptionally, including during protein translation and via post-translational modification of proteins. Notably, much of the control of gene expression is done either by regulatory proteins or by RNAs, which are themselves the products of genes. Hence, the interactions between DNA, RNAs, proteins, and other molecules, form natural gene regulatory networks (or GRNs) of varied complexity.

While studying these networks and their components provides invaluable information, it is essential to: (a) thoroughly investigate these components in different environments, while performing different functions, and (b) integrate this knowledge to build new synthetic gene regulatory networks and other devices. The discipline of Synthetic Biology aims at systematically designing, building, combining and testing new biological functions and systems that do not occur in nature. Indeed, individual parts such as promoters and protein coding sequences can be assembled into GRNs that perform desired functionalities, such as computing machines.

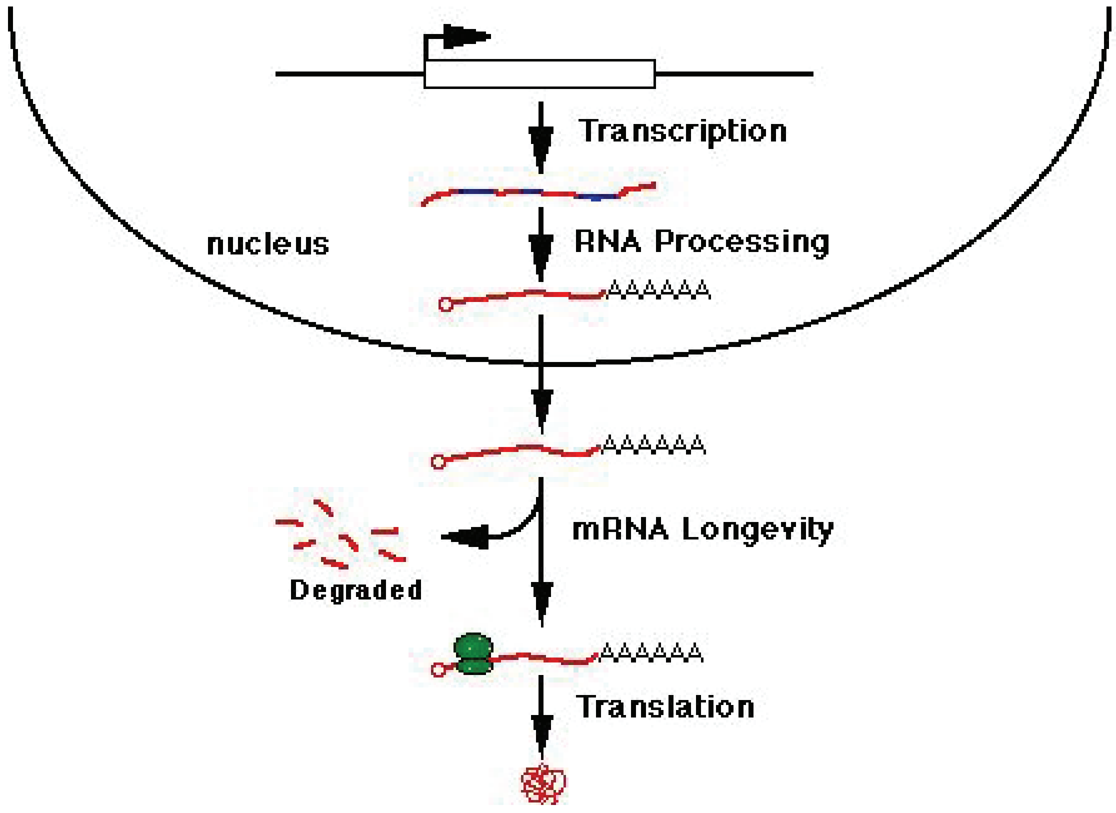


Figure 2-1. Gene expression in a eukaryote cell.[5]
The various steps in the gene expression can all be regulated. Inducers, repressors, activators, RNA interference and various other substances can be used to either inhibit or enhance this path.

2.1 Computing machines

The synthesis of computing machines *via* the manipulation of DNA within or without living organisms, started in 1994 when Adleman

[5] Image taken from the genetics website of Professor Robert S. Winning at Eastern Michigan University: "http://www.emunix.emich.edu/~rwinning/genetics/eureg.htm"

executed an experimental procedure that used DNA, *in vitro*, to solve an instance of the directed Hamiltonian path problem (Adleman, 1994). In contrast, *in vivo* cell-based or cellular computing started in 1998 with the modification of the genome of the prokaryote *E. coli*, to realize 1- and 2-input *combinatorial* Boolean logic gates (e.g. NOT, AND and IMPLIES) (Knight, Jr. and Sussman, 1998; Weiss et al., 1998); a similar feat was achieved with eukaryotic cells by Kramer *et al.* (Kramer et al., 2004). Along another dimension, time-dependant or sequential Boolean logic devices have also been implemented in living cells, starting with a hysteretic 2-input toggle switch by Gardner *et al.* (Gardner et al., 2000), a synthetic oscillator by Elowitz and Leibler (Elowitz and Leibler, 2000), and followed by Becskei *et al.*'s (Becskei et al., 2001) yeast-based memory device using positive feedback.

In one decade this field has grown to generate many elementary devices (Drubin et al., 2007; Boyle and Silver, 2009; Tigges et al., 2009; Haynes and Silver, 2009), including band-pass filters (Basu et al., 2005) and counters (Friedland et al., 2009). More complicated devices such as engineered multi-cellular pulse and pattern generators (Basu et al., 2004; Basu et al., 2005), single cell biosensors (Levskaya et al., 2005; Tecon et al., 2006), tumour-targeting bacteria (Anderson et al., 2006), and cell-based computers (Cox et al., 2007; Balagadde et al., 2008) have also been synthesized or proposed.

2.2 Simulation and Modelling

In parallel to advances in GRN design, mathematical modelling and simulation tools have been developed to help make approximate predictions of the behaviour of GRNs before significant resources are allotted to their synthesis. These include, but are not limited to, deterministic (Hindmarsh et al., 2005) and stochastic simulation algorithms (Gillespie, 1977), metabolic control analysis (MCA) (Olivier et al., 2005), structural analysis (Olivier et al., 2005) and flux-balance analysis (FBA) (Orth et al., 2010). Deterministic simulation models include differential equations, Boolean networks, logical networks and rule-based formalisms (de Jong, 2002). Stochastic models include Bayesian networks and master equations (de Jong, 2002). MCA quantifies how variables, such as fluxes and species concentrations, depend on network parameters. Structural analysis is mostly used for genome-scale models to determine reduced stoichiometric matrices. FBA is used for optimizing the growth rate of a modelled organism, while falling within the constraints of its internal metabolites.

2.3 Switch and Oscillator Designs

In the particular case of switching devices, there has been a fair number of switches built or theorized, which involve (a) DNA modification (e.g. using invertases), (b) regulation of the process of transcription, (c) post-transcriptional regulation (involving various RNA molecules), as well as (d) post-translational regulation (by changing the state of expressed proteins).

2.3.1 DNA Level Using Flipase Protein

The first example of the use of invertases is Ham *et al.* (Ham et al., 2006), which places the promoter of a gene between two specific elements targeted by the `FimE` flipase. The flipase inverts the inversion region between these two elements (including them). This completely disables transcription from that promoter, rendering the associated gene silent. This is a unidirectional operation and it does not require qualification by a clock. In 2008, Ham *et al.* (Ham et al., 2008) expanded their initial concept by using both the `hin` and `fimE` inversion mechanisms. This allowed them to use the relative positions of the elements marking the inversion regions to propose three- and five-state machines, which rely completely on the two flipases to change state. It is worth noting that this method of defining state is heritable as changes to the DNA are permanent and hence, inherited by the offspring.

2.3.2 Transcription Level

The most prominent example of a toggle switch that is transcriptionally controlled is that of Gardner *et al.* (Gardner et al., 2000). However, this toggle switch requires two inputs and operates asynchronously (is not controlled by a clock input).

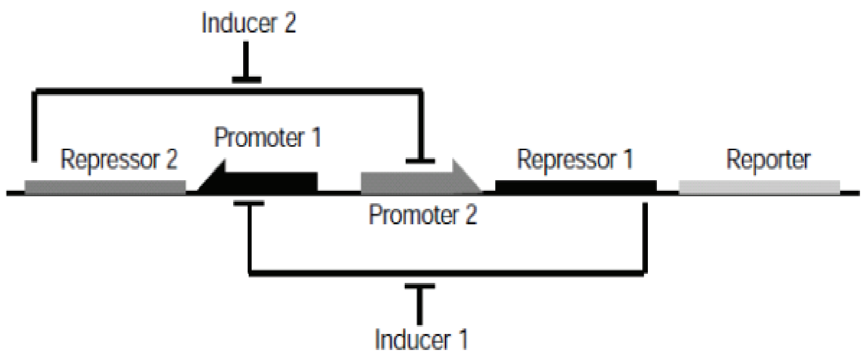


Figure 2-2. Toggle switch design.[6]
Repressor 1 inhibits transcription from Promoter 1 and is induced by Inducer 1. Repressor 2 inhibits transcription from Promoter 2 and is induced by Inducer 2.

Elowitz and Leibler (Elowitz and Leibler, 2000) synthesized a three gene oscillator (plus an additional gene for reporting), dubbed *repressilator*. The product of each of the three genes represses the next gene in a loop, with the last gene repressing the first one. The repressilator is not a bi-stable switch but rather a self-maintaining oscillator that proceeds from one state to the next, autonomously and without the need for any clock input. Becskei et al. (Becskei et al., 2001) presented a bi-stable positive feedback loop expressed in yeast in which a tetracycline-dependent activator turns on its own expression. They discussed how positive auto-regulation in GRNs can

[6] Figure taken from (Gardner et al., 2000)

turn an analogue input such as the concentration of a signalling molecule into a stepped (or digital) response with multiple steady states, allowing the pathway to be used as a memory element. Kobayashi *et al.* (Kobayashi et al., 2004) utilized slightly modified versions of Gardner's toggle switch as memory modules of larger networks that sensed specific events (e.g. DNA damage) and generated particular responses (e.g. biofilm formation). In this case, the toggle switch is, *by default*, in one specific state, which flips in response to the sensed event. It does not have two inputs, but it does not have two stable states either. And, as is the case with Gardner's switch, it operates asynchronously. Stricker *et al.* (Stricker et al., 2008) synthesized a two gene oscillating network, where one gene is responsible for the activation of both genes, and the other gene is responsible for repressing both genes. This network improves on the repressilator in terms of speed, durability of the oscillation and the ability to externally tune its oscillations. Nevertheless, this network is not a switch that can be used as a memory module, such as Gardner's toggle. Lou *et al.* (Lou et al., 2010) propose a single-input toggle switch, made of a Gardner-like two-gene memory module and a single-gene NOR gate module. The memory module is, by default, in a particular stable state. Upon the introduction of a UV input, several proteins degrade, which causes the memory module, with help from the NOR module to switch to a new state and maintain it. This is, in fact, a single-input switch, but it lacks a clock input.

2.3.3 Post-Transcription Level

One very significant work of RNA-based switching behavior is that of Bayer and Smolke (Bayer and Smolke, 2005). They present devices that are regulated post-transcriptionally using RNA *riboswitches*. A riboswitch is an RNA molecule containing two domains: (i) a ligand-binding aptamer domain and (ii) an antisense regulator domain. The latter is used to block the ribosome binding site (RBS) and prevent translation, while the former binds a ligand that triggers a conformational change in the riboswitch, resulting in either the covering or uncovering of the anti-sense regulator domain. Riboswitches have the advantage that they can be designed and/or evolved to respond to many ligands including proteins and RNA molecules. Riboswitches have been synthesized to respond to one or more inputs (ligands). Although current riboswitches change state uni-directionally, it is possible to imagine riboswitches that respond to inducible small protein ligands. So far, riboswitches act asynchronously. Another type of oscillations was demonstrated by Swinburne *et al.* (Swinburne et al., 2008) who proposed a self-repressed device containing an intron. An intron is any nucleotide sequence within a gene that is removed post-transcriptionally by RNA splicing to generate the final mature RNA product of a gene. The device demonstrated pulses of expression in mammalian cells. The frequency of the pulses was dependent on intron length.

2.3.4 Protein Level

Finally, a good example of how switches can be regulated at the protein level is the work of Dueber *et al.* (Dueber et al., 2003), which modified the natural N-WASP allosteric switch to synthesize 1- and 2-input synthetic protein switches. In the 2-input switch, the hybrid protein was engineered to have two A-terminal auto-inhibitory domains that correspond to the output domain and a C-terminal domain on the protein. The way in which the protein responded to the two input ligands (PDZ and Cdc42) relied on the relative positioning of the four domains. They used this to synthesize various switches, whose state (active or not) depended on combinatorial functions of the two inputs. All of their devices are asynchronous and unidirectional.

2.4 Making the Case

Despite the many works on genetic switches (also called flip-flops), all published synthesized and proposed designs work *asynchronously*, usually utilizing *more* than one external logical input. Lack of synchronization-ability entails that the operation of a flip-flop cannot be synchronized with the operation of other parts of a larger system, using a single global clock. Also, a true delay flip-flop has but one logical input. Though the use of a single input complicates design, it does simplify use and allow for easier expansion of function.

Table 2-1. Summary of Properties of Different Proposed Switching and Oscillating Circuits

Names	Year	2-Way Switching	1-Logical Input	Bi-stability	Synchronous	Realized
Gardner *et al.*	2000	✓		✓		✓
Elowitz *et al.*	2000	✓				✓
Becskei *et al.*	2001		✓			✓
Dueber *et al.*	2003		✓			✓
Kobayashi *et al.*	2004		✓	✓		✓
Bayer and Smolke	2005		✓			✓
Ham *et al.*	2006		✓	✓		✓
Swinburne *et al.*	2008	✓				✓
Stricker *et al.*	2008	✓				✓
Lou *et al.*	2010	✓	✓	✓		✓
Hoteit *et al.*	2011	✓	✓	✓	✓	

Table 2-1 lists a selection of proposed switches and highlights their publication year and five of their properties:

- *2-way Switching*: The circuit can switch more than once, from state *A* to *B* or from state *B* to *A*.
- *1-Logical Input*: Switching occurs using the same single input from

state *A* to *B* or from state *B* to *A*.

– *Bi-stability*: The switch is stable in state *A*, or in state *B*, before or after switching.

– *Synchronous*: The switch works on a clock.

– *Realized*: The design was realized in a lab, *in vitro* or *in vivo*.

We call the proposed GRN embodying a synchronous single-input delay flip-flop the *BioD*. It is, in summary, a novel GRN that changes states in response to a single logical input, and only on the rising edge of a clock signal. Its specification and detailed design, modelling and simulation results follow.

CHAPTER 3.

NETWORK DESIGN AND MODELLING

In abstract terms, the *BioD* is a gene regulatory network acting as a delay flip-flop. By delay flip-flop, we mean a logical device that has an input (*D*), a clock (*CLK*), and an output (*Q*) equal to its state (*S*); see the logical block diagram in Figure 3-1 ($\overline{Q}$ is the second output and is equal to the logical complement of *Q*). The state of the delay switch is held constant unless and until its input differs from its state, on the rising edge of the clock. In that case, the next state of the delay switch will copy the value of the input (i.e., $Q = D$). Hence, a cell that acts as a delay switch is effectively a 1-bit memory device, controlled by an input and a clock. The *BioD* also exhibits its state by expressing (or not) a fluorescent protein. This was the specification of the *BioD*; following is its internal *design*.

3.1 BioD

The *BioD* has two (logical and control) inputs: *trans*-activating RNA or *ta*RNA as input *D*, and the presence or absence of far-red (FR) light as the clock (*CLK*). It has two complementary outputs (*Q* and $\overline{Q}$) defining the state of the flip-flop: the ON state is indicated by the presence, in high concentrations, of green fluorescent protein (*GFP*), while the OFF state is indicated by its absence. As with its electronic equivalent, the *BioD*'s output follows the input on the rising edge of the clock. As shown in Figure 3-2, the gene regulatory network implementing the *BioD* is comprised of three major parts: the INPUT genes, SELECTION genes and STATE genes.

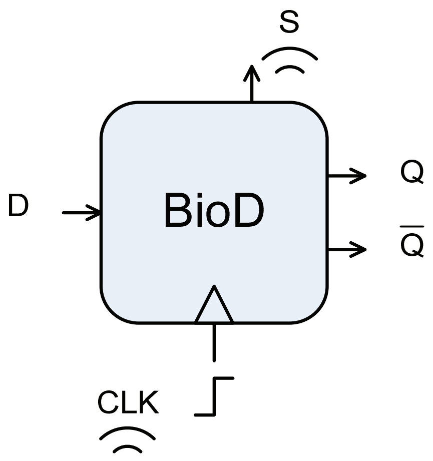


Figure 3-1. The Logical Block Diagram for *BioD*.

Please note that the design involves several operons that include more than one protein coding sequence. To simplify our language without loss of accuracy, we refer to both genes and operons as genes (there are seven of them, numbered 1 to 7). Kindly note that we use italicized courier new for gene names (e.g. *TetR*) and courier new for proteins (e.g. TetR) as well as protein complexes. We also use italicized courier new for RNAs other than transcripts (e.g. *taR12*), while distinguishing transcripts by attaching an "m" prefix to their names (e.g. *mTetR*).

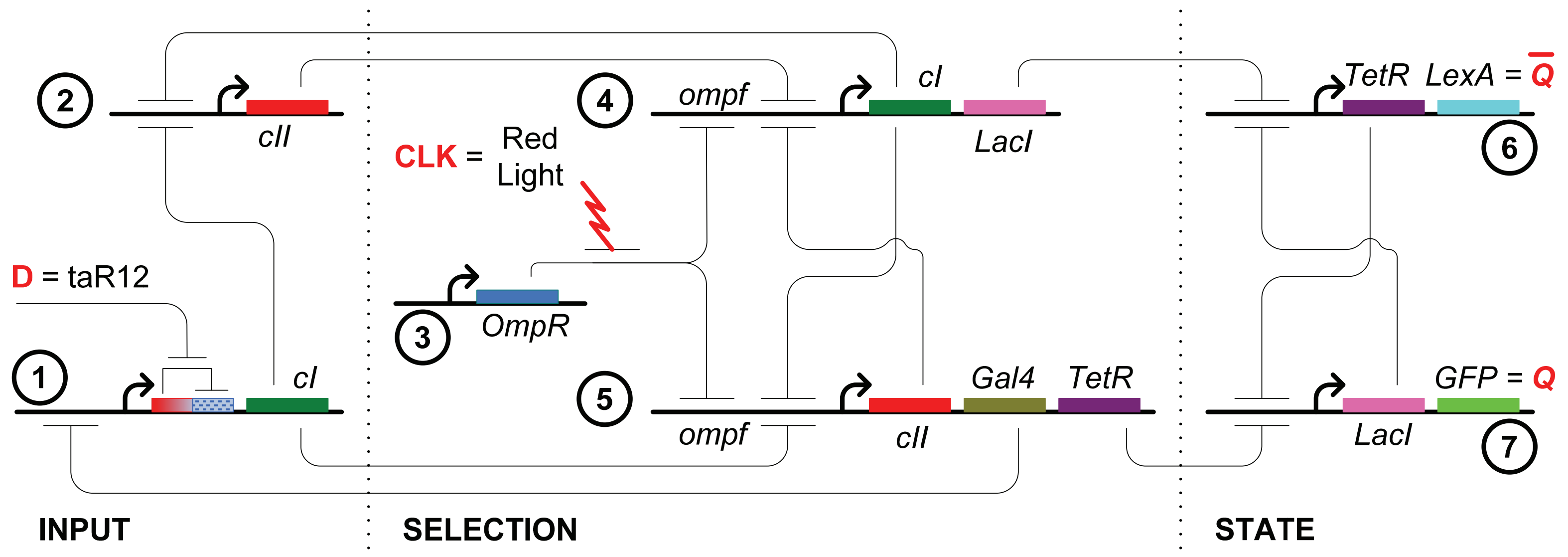


Figure 3-2. The gene regulatory network of the *BioD*.

The real genes selected to realize this network are just one possible implementation of the logical device (see Figure 3-3). The network consists of three segments. The STATE genes reflect the state of the network. The SELECTION genes determine the next state of the network by regulating the STATE genes, but only when the clock has just turned ON. The external logical input to the whole network goes through the INPUT genes, which in turn affect the SELECTION genes.

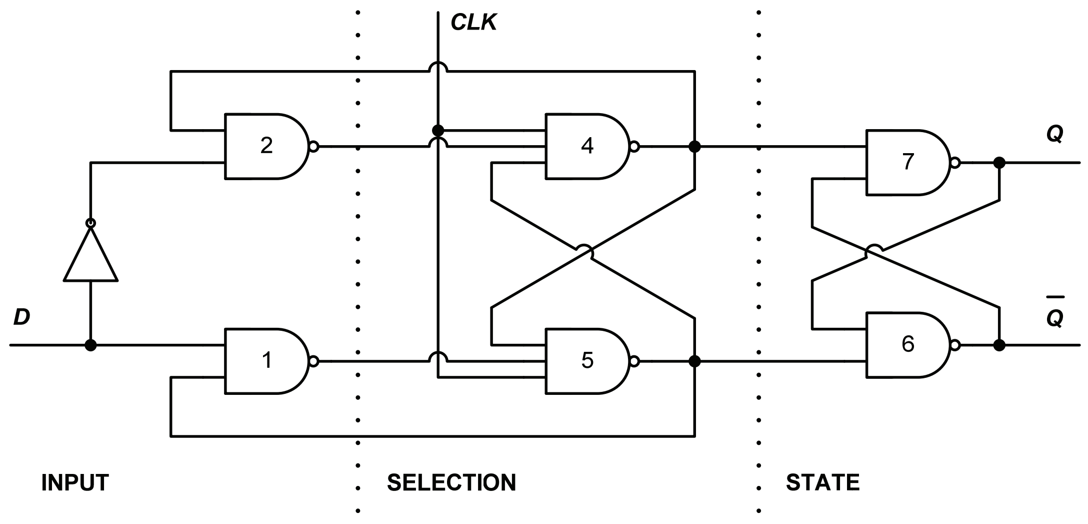


Figure 3-3. A logical circuit representation of the *BioD*.
Logically, this circuit behaves like the GRN of Figure 3-2. It is not an exact representation, but it is useful in following the steps the *BioD* network takes when changing state. The gene numbers in Figure 3-2 match gate numbers here. A low *CLK* signal disables SELECTION gates 4 and 5, and sends a high signal (identity for NAND gates) to the STATE gates 6 and 7, maintaining their state. Since the outputs of INPUT gates 1 and 2 are complements, when the *CLK* signal is turned ON, only one of gates 4 and 5 becomes active and thus (i) affects one of the STATE gates (6 or 7) and (ii) disables its enabling INPUT gate (1 or 2). The INPUT gates are re-enabled after the *CLK* goes low, leaving them free to respond to new input values (at *D*).

3.1.1 INPUT Genes

The INPUT genes convey to the SELECTION genes whether an input signal is present or not. They do so by tipping the dynamic balance between the two mutually-repressed genes, 4 and 5; this process is detailed in section 3 below.

In order to sense input *D*, gene 1 is designed to be self-repressed, and this self-repression can only be lifted through the introduction of input *D*. To achieve this, a form of ribo-regulation is used called *cis*. This *cis*-regulation or in our case, *cis*-repression prevents the translation of the transcript of gene 1, as part of the transcript bends over to hybridize with the ribosome binding site (RBS), effectively locking it. The key comes in the form of *trans-activating* RNA (*ta*RNA), which hybridizes with a particular location on the transcript in a manner that frees the RBS site from its *cis*-repression. This allows the ribosome to bind at the RBS and start the process of translation (Isaacs et al., 2004). The *ta*RNA chosen for input *D* is *taR12* which is specifically designed to unlock the *cis*-repression of (the transcript of) gene 1, called *crR12*.

When input *D* is present, the transcript of gene 1 gets translated into the *cI* repressor (originally, from the λ phage). *cI* in turn represses gene 2. In the absence of input *D,* however, the *cis*-repressed transcript of gene 1 does *not* get translated into the corresponding repressor protein. This leads to the lifting of repression of gene 2, and hence the expression of its own repressor protein, *cII* (originally, from the P22 phage).

In summary, the presence of input *D* results in the production of the *cI* protein, while its absence leads to the production of the *cII* protein.

3.1.2 STATE Genes

The STATE genes have an analogous configuration to that of Gardner's toggle. They consist of two co-repressed genes, and as such define the state of the *BioD* device. The products of genes 7 and 6 represent complementary outputs Q and $\overline{Q}$, respectively. The presence of a green fluorescent protein (GFP) signals the presence of logical output Q, while its absence signals the presence of its logical complement $\overline{Q}$. The co-repressed nature of the toggle switch means that when either gene is active, the toggle enters into a stable steady state. In the context of the *BioD*, only the SELECTION genes can perturb the stability of the SELECTION genes.

Two important points need to be made here. First, the SELECTION genes can affect the STATE genes, *independently* of the current state of the *BioD*. Second, genes 4 and 5 are mutually exclusive, which renders it impossible for the SELECTION genes to set the state of the STATE genes to both ON *and* OFF, simultaneously. Which of the two genes (4 or 5) is activated depends on the state of the INPUT genes at the time the *CLK* signal is turned ON.

3.1.3 SELECTION Genes

The SELECTION genes are always OFF until turned ON by FR light (the *CLK* input). In the absence of FR light, genes 4 and 5 are always repressed by the phosphorylated version of *OmpR*, i.e. *OmpRP*. Gene 3 is constitutively expressed and produces *OmpR*. *OmpR* is phosphorylated in the presence of the *EnvZ* enzyme. *EnvZ* is connected to *Cph1*, which in the presence of FR light, induces a conformational change in *EnvZ* preventing the phosphorylation of *OmpR*. The genes that produce *EnvZ* and *Cph1* (and others needed for the light response system) are not shown in Figure 3-2. See reference (Levskaya et al., 2005) for a fully detailed explanation.

The phosphorylation of *OmpR* is dominant in the absence of FR light and negligible in its presence. Therefore, the FR light signal causes a drop in *OmpRP* levels and a corresponding rise in *OmpR* levels. This drop results in *partial* lifting of the repression of both genes 4 and 5, as their promoter *ompf*, is both repressed by *OmpRP* and activated by *OmpR*. Both the functionality of *ompf* and the complementary levels of *OmpR* and *OmpRP* result in a system that is quick to start or stop transcription of both genes 4 and 5.

The SELECTION genes also respond to and affect the INPUT genes. As previously stated, the *BioD* is an edge-triggered device, i.e. it responds to the input when the *CLK* signal *turns* ON, but not when the CLK signal *is* ON. If the CLK signal is ON and either gene 4 (or 5) is ON, then gene 4 (or 5) would be repressing the genes that could potentially repress it. Namely, gene 4 would repress genes 2 and 5, and gene 5 would repress genes 1 and 4. As a result, any change due to input *D,* when the CLK signal is already ON, does not propagate to the SELECTION genes. For a toggle (ON) input signal to affect the current state of the SELECTION genes, the *CLK* signal must first turn OFF for a period then ON again.

Table 3-1. State Transition Table
"X" is don't care. "=" is no change.

CLK	*D*	CURRENT STATE						NEXT STATE					
		S_1	S_2	S_4	S_5	S_6	S_7	S_1	S_2	S_4	S_5	S_6	S_7
$\underline{0}$	0	0	1	0	0	X	X	0	1	0	0	=	=
$\underline{0}$	1	1	0	0	0	X	X	1	0	0	0	=	=
↑	0	0	1	0	0	X	X	0	1	0	1	1	0
↑	1	1	0	0	0	X	X	1	0	1	0	0	1
$\overline{1}$	0	X	X	X	X	X	X	=	=	=	=	=	=
$\overline{1}$	1	X	X	X	X	X	X	=	=	=	=	=	=
↓	0	X	X	X	X	X	X	0	1	0	0	=	=
↓	1	X	X	X	X	X	X	1	0	0	0	=	=

Given that the dynamics of such a gene network are non-trivial, we provide a summary of its operation using a state transition table (Table 3-1) plus a single fully detailed scenario, tracing through one typical sequence of transitions. The scenario is that of a change of state, from OFF to ON, in response to a turned ON input (*D*), whose level must stabilize, prior to the introduction of the *CLK* signal (FR light).

When the state of the *BioD* is OFF, gene 6 is ON, expressing two products. Since one of them (*TetR*) is repressing gene 7, gene 7 is considered OFF. In the absence of FR light, the constitutively expressed (and subsequently phosphorylated) repressor (*OmpRP*) blocks any production from the SELECTION genes (4 and 5). Hence, the status quo of the STATE genes is maintained. Lastly, gene 1 is ON, induced by input (*D*), while gene 2 is OFF, repressed by the product of gene 1, *cI*. After clocking, the concentration of *OmpRP* (which was repressing genes 4 and 5) starts falling. The only other repressor of gene 4 (i.e. *cII* from gene 2) is already OFF. So gene 4 can start producing, and as such, it starts repressing gene 5, which is still repressed by *cI* from gene 1. At this point in time, gene 1 is ON, gene 2 is OFF, gene 4 is ON, gene 5 is OFF, while gene 6 is still ON and gene 7 is still OFF. Turning our attention to gene 4, note that one of the repressors it produces is identical to the one generated by gene 7, namely *LacI*. Its production starts switching off gene 6, resulting in a gradual increase in the expression of gene 7. Once gene 7 is fully expressed, its product (*LacI*) represses gene 6, ensuring the continuation of gene 7’s new ON state. Hence, we have achieved a change of network state (indicated by GFP) from OFF to ON (following the value of the input (*D*)).

As long as the *CLK* signal is ON, the new state will be maintained. If a significant change in the input level occurs while the clock is ON, the repressions of genes 2 and 5 would not disappear, since gene 4 is ON and produces *cI*. Indeed, as long as gene 4 is ON, it has the ability to keep itself from being repressed by other genes, that is, by repressing them. It is only when the *CLK* signal is removed and both genes 4 and 5 are OFF that the system is again free to respond to input (*D*), upon the re-introduction of the *CLK* signal.

Model

The gene regulatory network of Figure 3-2 is simulated deterministically and stochastically, following a mathematical model. The model is shown below as (a) a system of ordinary differential equations (ODEs) modelling the production of mRNA transcripts, and (b) a system of ODEs modelling the translation of the transcripts into their respective proteins.

The same system of ODEs constitutes the basis of the stochastic simulation used to generate the data for Figure 4-1, Figure 4-5 and Figure 4-6. We used the tau-leaping algorithm (Cao et al., 2007), which achieves fast and accurate simulation by taking large time steps that leap over individual reactions. We chose to show the results of the stochastic simulation because they are similar to, but are more realistic than the deterministic ones.

We define the following terms and chemical species: $mCIcr$ is the *cis*-repressed mRNA transcript of gene 1; mX is the mRNA transcript for the protein X; $prod_{GeneX}$ is the amount of transcripts produced by gene X at any given time; ρ_X is the maximum transcription rate of the promoter of gene X; while ω_X, n_X, K_X and $[X]$ are, respectively, the degradation constant, the Hill cooperativity coefficient, the dissociation constant and the concentration of substance X.

Transcription ODEs

$$\frac{d[mCIcr]}{dt} = prod_{Gene1} - \omega_{mRNA}.[mCIcr] \qquad (3.1)$$

$$\frac{d[mCI]}{dt} = prod_{Gene4} - \omega_{mRNA}.[mCI] \qquad (3.2)$$

$$\frac{d[mCII]}{dt} = prod_{Gene2} + prod_{Gene5} - \omega_{mRNA}.[mCII] \qquad (3.3)$$

$$\frac{d[mOmpR]}{dt} = prod_{Gene3} - \omega_{mRNA}.[mOmpR] \qquad (3.4)$$

$$\frac{d[mGal4]}{dt} = prod_{Gene5} - \omega_{mRNA}.[mGal4] \qquad (3.5)$$

$$\frac{d[mTetR]}{dt} = prod_{Gene5} + prod_{Gene6} - \omega_{mRNA}.[mTetR] \tag{3.6}$$

$$\frac{d[mLacI]}{dt} = prod_{Gene4} + prod_{Gene7} - \omega_{mRNA}.[mLacI] \tag{3.7}$$

where gene 1 is repressed by *Gal4*,

$$prod_{Gene1} = \frac{\rho_1}{1 + \left(\frac{[Gal4]}{K_{Gal4}}\right)^{n_{Gal4}}} \tag{3.8}$$

gene 2 is repressed by *cI*,

$$prod_{Gene2} = \frac{\rho_2}{1 + \left(\frac{[CI]}{K_{CI}}\right)^{n_{CI}}} \tag{3.9}$$

gene 3 is constitutively expressed,

$$prod_{Gene3} = \rho_3 \tag{3.10}$$

gene 4 is repressed by both *cII* and *OmpRP*, while being activated by *OmpR*,

$$prod_{Gene4} = \frac{\rho_4}{1 + \left(\frac{[CII]}{K_{CII}}\right)^{n_{CII}}} * \frac{1}{1 + \left(\frac{[OmpRP]}{K_{OmpRP}}\right)^{n_{OmpRP}}} * \frac{\left(\frac{[OmpR]}{K_{OmpR}}\right)^{n_{OmpR}}}{1 + \left(\frac{[OmpR]}{K_{OmpR}}\right)^{n_{OmpR}}} \tag{3.11}$$

gene 5 is repressed by both *cI* and *OmpRP*, while being activated by *OmpR*,

$$prod_{Gene5} = \frac{\rho_5}{1+\left(\frac{[CI]}{K_{CI}}\right)^{n_{CI}}} * \frac{1}{1+\left(\frac{[OmpRP]}{K_{OmpRP}}\right)^{n_{OmpRP}}} * \frac{\left(\frac{[OmpR]}{K_{OmpR}}\right)^{n_{OmpR}}}{1+\left(\frac{[OmpR]}{K_{OmpR}}\right)^{n_{OmpR}}} \tag{3.12}$$

gene 6 is repressed by *LacI*,

$$prod_{Gene6} = \frac{\rho_6}{1+\left(\frac{[LacI]}{K_{LacI}}\right)^{n_{LacI}}} \tag{3.13}$$

and gene 7 is repressed by *TetR*,

$$prod_{Gene7} = \frac{\rho_7}{1+\left(\frac{[TetR]}{K_{TetR}}\right)^{n_{TetR}}} \tag{3.14}$$

<u>Translation ODEs</u>

$$\frac{d[CI]}{dt} = \gamma_{CI}.K_{taR12}.[taR12].[mCIcr] + \gamma_{CI}.[mCI] - \omega_{CI}.[CI] \tag{3.15}$$

$$\frac{d[CII]}{dt} = \gamma_{CII}.[mCII] - \omega_{CII}.[CII] \tag{3.16}$$

The $(1 - L)$ term inserted in the Michaelis-Menten expressions of equations (3.17) and (3.18) enables phosphorylation in the absence of FR light, i.e. when $L = 0$.

$$\frac{d[OmpR]}{dt} = \gamma_{OmpR}.[mOmpR] - \frac{v_{phos}.(1-L).[OmpR]}{k_{phos}+[OmpR]} + v_{dePhos}.[OmpRP] - \omega_{OmpR}.[OmpR] \tag{3.17}$$

$$\frac{d[OmpRP]}{dt} = \frac{v_{phos}.(1-L).[OmpR]}{k_{phos}+[OmpR]} - v_{dePhos}.[OmpRP] - \omega_{OmpRP}.[OmpRP] \tag{3.18}$$

$$\frac{d[Gal4]}{dt} = \gamma_{Gal4}.[mGal4] - \omega_{Gal4}.[Gal4] \tag{3.19}$$

$$\frac{d[TetR]}{dt} = \gamma_{TetR}.[mTetR] - \omega_{TetR}.[TetR] \tag{3.20}$$

$$\frac{d[LacI]}{dt} = \gamma_{LacI}.[mLacI] - \omega_{LacI}.[LacI] \tag{3.21}$$

Parameters values are as shown in Table 3-2. Please refer to the *Discussion* section for a discussion of the dissociation constants. The degradation rates of various molecules are not known, so we use the rates arising from dilution by cell-growth. Somewhat elevated rates are used for ω_{CI} and ω_{TetR} in order to avoid lingering production of *cI*, and *TetR*, when the state is not favourable. That is feasible because protein degradation rates can be artificially increased by adding to the protein coding sequence an SsrA tag, making the modified protein a target of various proteases in the cell (Elowitz and Leibler, 2000).

Table 3-2. Nominal Values of the Parameters of Transcription and Translation Equations

Parameter	Description	Value	Reference
GENERAL PARAMETERS			
ρ_1	max. transcription rate of promoter of gene 1	0.680 [nM/s]	Estimate
ρ_2	max. transcription rate of promoter of gene 2	0.595 nM/s]	Estimate
ρ_3	max. transcription rate of constitutive promoter of gene 3	0.085 [nM/s]	Estimate
ρ_4	max. transcription rate of promoter of gene 4	0.255 [nM/s]	Estimate
ρ_5	max. transcription rate of promoter of gene 5	0.255 [nM/s]	Estimate
ρ_6	max. transcription rate of promoter of gene 6	0.765 [nM/s]	Estimate
$\rho_{7,8,9,10}$	max. transcription rate of promoters of genes 7, 8, 9 and 10	0.850 [nM/s]	Estimate
γ_X	Translation rate of gene X (any gene)	0.1	Estimate
υ_{phos}	Rate of *OmpR* phosphorylation	20.0	Estimate
υ_{dePhos}	Rate of *OmpRP* de-phosphorylation	0.01	Estimate
k_{phos}	Kinetic phosphorylation constant	1.0	Estimate
DEGRADATION CONSTANTS			
ω_{LacI}	degradation of *LacI*	2.31e-3 [1/s]	Estimate
ω_{TetR}	degradation of *TetR*	2.3e-2 [1/s]	(Baumeister et al., 1991)
ω_{CI}	degradation of *cI*	7e-4 [1/s]	(Reinitz and Vaisnys, 1990)
ω_{CII}	degradation of *cII*	6.9e-3 [1/s]	(Vohradsky, 2001)
ω_{OmpR}	degradation of *OmpR*	0.13e-2 [1/s]	(Zhu et al., 2000)
ω_{Gal4}	degradation of *Gal4*	2.88e-2 [1/s]	Estimate
ω_{LexA}	degradation of *LexA*	0.0115 [1/min]	(Camas et al., 2006) (half-life of ~60mins)
$\omega_{LuxR\cdot AHL}$	degradation of *LuxR·AHL*	1e-3 [1/s]	(Goryachev et al., 2006)
$\omega_{RhlR\cdot AHL}$	degradation of *RhlR·AHL*	1e-3 [1/s]	Estimate
ω_{GFP}	degradation of *GFP*	0.012 [1/min]	(de Jong et al., 2010)
ω_{taR12}	degradation of *taR12*	1.96e-3 [1/s]	(Friedland et al., 2009)
ω_{mRNA}	degradation of an mRNA transcript	2.88e-3 [1/s]	(Alon, 2006)
DISSOCIATION CONSTANTS			
K_{LacI}	*LacI* repressor dissociation constant	10 [nM]	(Wang et al., 2005)
K_{TetR}	*TetR* repressor dissociation constant	5.6 [nM]	(Stekel and Jenkins, 2008)
K_{CI}	*cI* repressor dissociation constant	8 [nM]	(Basu et al., 2005)
K_{CII}	*cII* repressor dissociation constant	50 [nM]	Estimate
K_{OmpR}	*OmpR* repressor dissociation constant	151 [nM]	(Head et al., 1998)
K_{OmpRP}	*OmpRP* repressor dissociation constant	6 [nM]	(Head et al., 1998)
K_{Gal4}	*Gal4* repressor dissociation constant	24 [nM]	(Hong et al., 2008)
K_{LexA}	*LexA* repressor dissociation constant	20 [nM]	(Kuhner et al., 2004)
$K_{LuxR\cdot AHL}$	*LuxR·AHL* affinity	10 [nM]	(Basu et al., 2005)
$K_{RhlR\cdot AHL}$	*RhlR·AHL* affinity	10 [nM]	Estimate
K_{taR12}	*taR12* repressor dissociation constant	80 [nM]	(Isaacs et al., 2004)

HILL COOPERATIVITY			
n_{LacI}	`LacI` repressor Hill cooperativity	2	(Basu et al., 2005)
n_{TetR}	`TetR` repressor Hill cooperativity	2	Estimate
n_{CI}	`cI` repressor Hill cooperativity	2	(Basu et al., 2005)
n_{CII}	`cII` repressor Hill cooperativity	2	(Shih and Gussin, 1984)
n_{OmpR}	`OmpR` repressor Hill cooperativity	2	Estimate
n_{OmpRP}	`OmpRP` repressor Hill cooperativity	2	Estimate
n_{Gal4}	`Gal4` repressor Hill cooperativity	2	Estimate
n_{LexA}	`LexA` repressor Hill cooperativity	2	(Aksenov, 1999)
$n_{LuxR \cdot AHL}$	`LuxR·AHL` Hill cooperativity	1	(Basu et al., 2005)
$n_{RhlR \cdot AHL}$	`RhlR·AHL` Hill cooperativity	1	Estimate
n_{taR12}	`taR12` repressor Hill cooperativity	2	Estimate

3.2 BioFSM

From a computational point of view, a logical next step to the *BioD* is the design of a GRN embodying a finite state machine, which uses the *BioD* as a 1-bit memory module. We call this design a *BioFSM*, which is also a stand-alone module that can be modified to carry out different logical functions and/or to communicate with other modules via inter-cellular signalling.

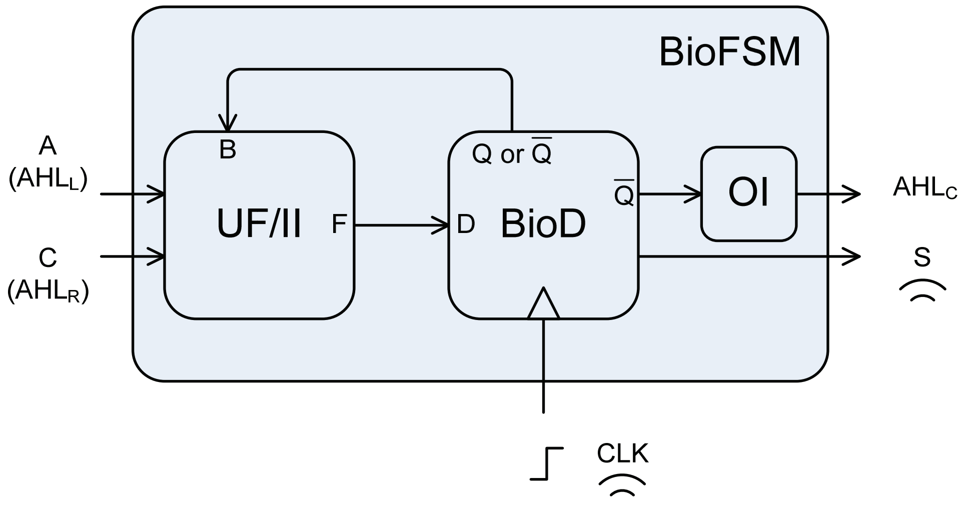


Figure 3-4. The Logical Block Diagram for *BioFSM*.

The *BioFSM* has the following specification, characterized by its inputs, clock and current state. When the clock is OFF, there is no change in the state of the device. However, when the clock turns ON, the next state of the *BioFSM* is determined by a state update function (the UF), which is a function of its external inputs and its own current state.

The design of the *BioFSM* is shown in Figure 3-4. It consists of 3 modules: (a) a *BioD*, which holds the state of the *BioFSM*; (b) the UpdateFunction/InputInterface (or UF/II) module. The UF determines the next state of the *BioD*. The genes implementing the UF/II implicitly include the input interface, as changing any of the two external inputs requires a change to the promoter side of the genes (see Figure 3-5b); (c) the OutputInterface (OI) module, which is used to enable a chosen acyl-homoserine lactone (AHL) molecule as the output of the *BioFSM* (Figure 3-5c); AHLs are a class of small molecules capable of inter-cellular signalling in *E. coli* and other bacteria (Fuqua et al., 2001). In fact, the two external inputs to the *BioFSM* are also AHLs. The modular design of the *BioFSM* allows us to alter its logic/inputs or output only by changing only its UF/II or OI, respectively.

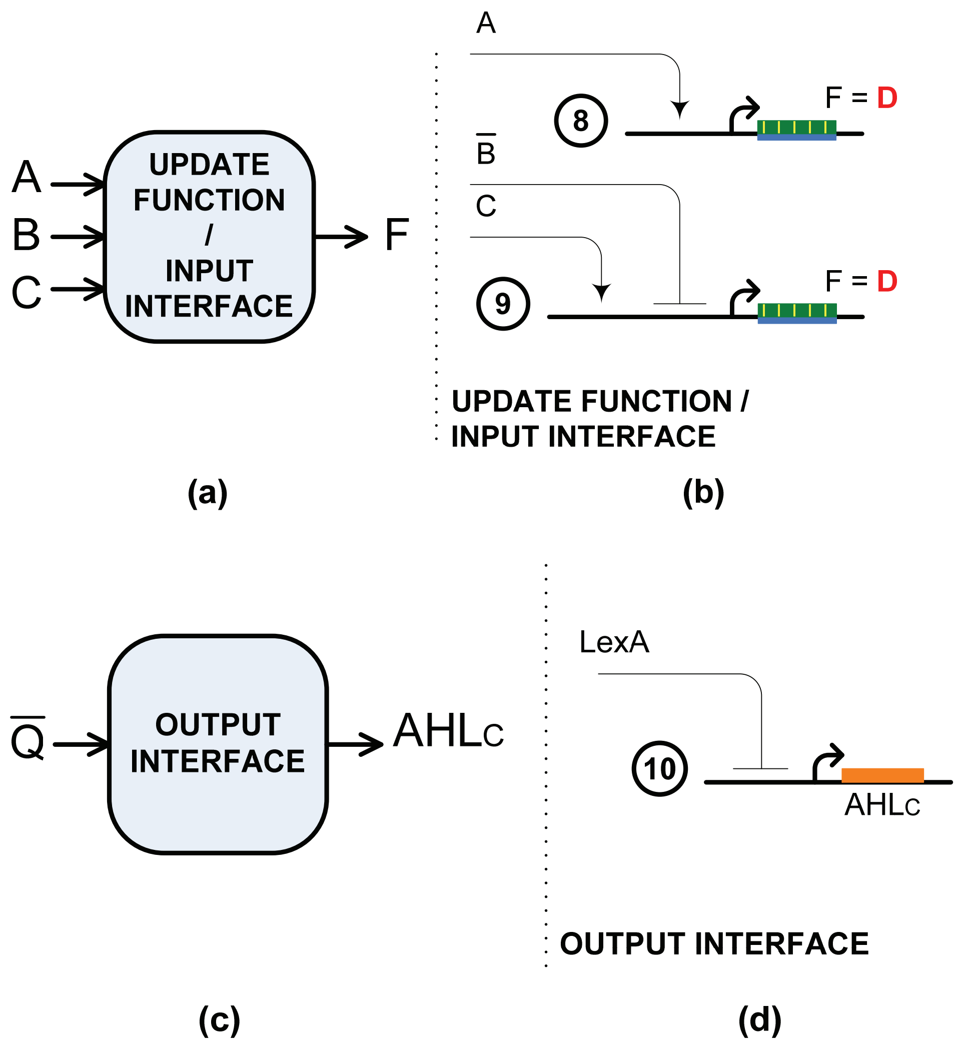


Figure 3-5. UF/II and OI implementing F=A+BC.
a. Logical block diagram for the UpdateFunction/InputInterface (UF/II). Inputs A and C are AHLs coming from neighbouring *BioFSMs*. They are the left and right inputs, so A=AHLL and C=AHLR. Input B is a repressor that reflects the state of the *BioFSM* and comes directly from the STATE genes of the embedded *BioD*. **b.** The gene regulatory network for the UF/II, where $F = A + BC$. The output *F* is the input *D* (or *taR12*) to the *BioD*. **c.** Logical block diagram for the OutputInterface (OI). AHLC (centre) is the particular AHL assigned to this *BioFSM*. It is used to transmit the state of the device to its neighbours. The presence of *LexA* reflects the OFF state (i.e. the $\overline{QQ}$ output) of the *BioD*. The OI stops production of the AHL when the *BioD* is in an OFF state. **d.** The gene regulatory network realizing the OI; it is made of one gene.

The example shown in Figure 3-5 illustrates a particular UF/II and OI. The update function is $A + BC$. Inputs *A* and *C* are the two originating from external sources, and are both AHLs, while *B* represents the state of the *BioD*, and is a repressor. It is worth noting here that AHLs can be activators or repressors based on the positioning of the binding site of the R-protein/AHL complex within the promoter region of the AHL-regulated gene (Anderson et al., 1999; Medina et al., 2003). Hence, the logical complements of the external inputs, $\overline{A}$ and $\overline{C}$, are readily available, while the state *B* = $LacI$ and its logical complement $\overline{B}$ = $TetR$ are made available by the *BioD*. This flexibility often allows for the reduction in the number of genes required for the implementation of the UpdateFunction. As to the OutputInterface, all possible realizations are driven by the *BioD*'s $LexA$ output, but would have different (AHL) products, depending on the application.

Before providing the model, a word about AHLs and the way they function. AHLs are capable of inter-cellular signalling partly because they are small molecules capable of diffusion across membranes. Even though they are small, they are capable of being indirectly used as activators or repressors by forming complexes with larger proteins called R-proteins. The resulting R-protein-AHL complex can activate or repress production of genes by binding to specific operator sites in the promoter region of those genes. There exists many different types of AHLs (Fuqua et al., 2001; Shrout and Parsek, 2006; Steindler and Venturi, 2007) and each AHL has a particular R-protein that it activates. In the design of the *BioFSM*, we have two external AHL input signals and one AHL output signal. Specifically, we use the `RhlI/RhlR` and `LasI/LasR` pairs for input, and the `LuxI/LuxR` pair for output.

Model

The model used for the UF/II and the OI simulations for $F = A + BC$ is presented below. The production of the R-proteins is not considered here because they are constitutively produced proteins, generated without regulation. The protein translation ODEs are not shown because there is no post transcriptional regulation.

Transcription ODEs

$$\frac{d[taR12]}{dt} = prod_{Gene8} + prod_{Gene9} - \omega_{mRNA}.[taR12] \tag{3.22}$$

$$\frac{d[mLasI]}{dt} = prod_{Gene10} - \omega_{mRNA}.[mLasI] \tag{3.23}$$

where gene 8 is activated by the $LuxR \cdot AHL$ complex,

$$prod_{Gene8} = \rho_8 * \frac{\left(\frac{[LuxR \cdot AHL]}{K_{LuxR \cdot AHL}}\right)^{n_{LuxR \cdot AHL}}}{1 + \left(\frac{[LuxR \cdot AHL]}{K_{LuxR \cdot AHL}}\right)^{n_{LuxR \cdot AHL}}} \tag{3.24}$$

gene 9 is repressed by `TetR` and activated by the `RhlR·AHL` complex,

$$prod_{Gene9} = \frac{\rho_9}{1 + \left(\frac{[TetR]}{K_{TetR}}\right)^{n_{TetR}}} * \frac{\left(\frac{[RhlR \cdot AHL]}{K_{RhlR \cdot AHL}}\right)^{n_{RhlR \cdot AHL}}}{1 + \left(\frac{[RhlR \cdot AHL]}{K_{RhlR \cdot AHL}}\right)^{n_{RhlR \cdot AHL}}} \tag{3.25}$$

and gene 10 is repressed by `LexA`,

$$prod_{Gene10} = \frac{\rho_{10}}{1 + \left(\frac{[LexA]}{K_{LexA}}\right)^{n_{LexA}}} \tag{3.26}$$

3.3 Simulation Methodology

3.3.1 Language

The above systems of ODEs were solved using our own implementation (written in the C++ programming language) of the common forth-order Runge-Kutta method (Kaps and Rentrop, 1979). The source code is available in the appendix.

3.3.2 Inputs and Outputs

As it stands, when simulating the *BioD*, the *D* and *CLK* inputs are manually set to a particular value before the beginning of every new simulation. When simulating the BioFSM, only the *CLK* input and the external AHL inputs are manually assigned.

The output generated by the program is a matrix of tab delimited values representing the state of every differential equation, at each step. Every line starts with the time step and is followed by the values of every ODE in the system at that time step.

3.3.3 Tools

This matrix of values serves as one of two inputs to the plotting program, Gnuplot (Janert, 2009), which is used for both results plotting and viewing. The other input is a file that holds the details of the plot; *e.g.* plot area, axes, zoom, plot colours and highlighted areas (given in appendix). We manually included two types of highlighting regions (a red-hue and diagonal-stripes) in the output generated by Gnuplot to indicate the time during which the inputs are present. The program itself always generates the complete list of values for any given simulation. However, in our Gnuplot generated figures, we choose to plot only the values of interest.

3.3.4 Stochastic run

The systems of ODEs presented in the previous section are the basis of the stochastic simulation used to generate Figure 4-1, Figure 4-5 and

Figure 4-6. We used the tau-leaping algorithm (Cao et al., 2007), which achieves fast and accurate stochastic simulation by taking large time steps that leap over individual reactions. During a leap interval (t, t + τ) in tau-leaping, each reaction channel operates as a Poisson process with a constant intensity.

3.3.5 Parameters

The values of τ used in our simulations varied from τ = 5, τ = 10 up to τ = 20 in an effort to display the most relevant plots. Increasing the value of τ reduces the resolution of the results but reduces the computing time of the simulation.

Table 3-3. CPU time (performance) needed to simulate one hour of biological time.

Simulation Step	**Time (in seconds)**
τ = **5**	*1.181*
τ = **10**	*0.636*
τ = **20**	*0.273*

In the deterministic run, τ also represents the time step. No noticeable performance change was observed between the deterministic and stochastic runs of equal time steps.

CHAPTER 4.

SIMULATION RESULTS AND DISCUSSION

In the sequel, we present the results of simulating the device using a system of rate equations. The results confirm our expectation that the device will toggle when and only when required – though its speed can still be improved.

4.1 BioD

The core functionality of our *BioD* device is illustrated in Figure 4-1. The highlighted areas indicate the presence of an input. The reddish hue reflects the presence of the clock input (*CLK*), while the grey diagonal pattern reflects the presence of the data input (*D*). The examples provided have two different data cycles intersecting (or not) with four different clock cycles. This setting allows us to show that the device can indeed go from one state to the other in response to nothing more than the introduction of the inputs it was designed to respond to. Furthermore, this setting also goes through the various permutations of the inputs seen in Table 3-1.

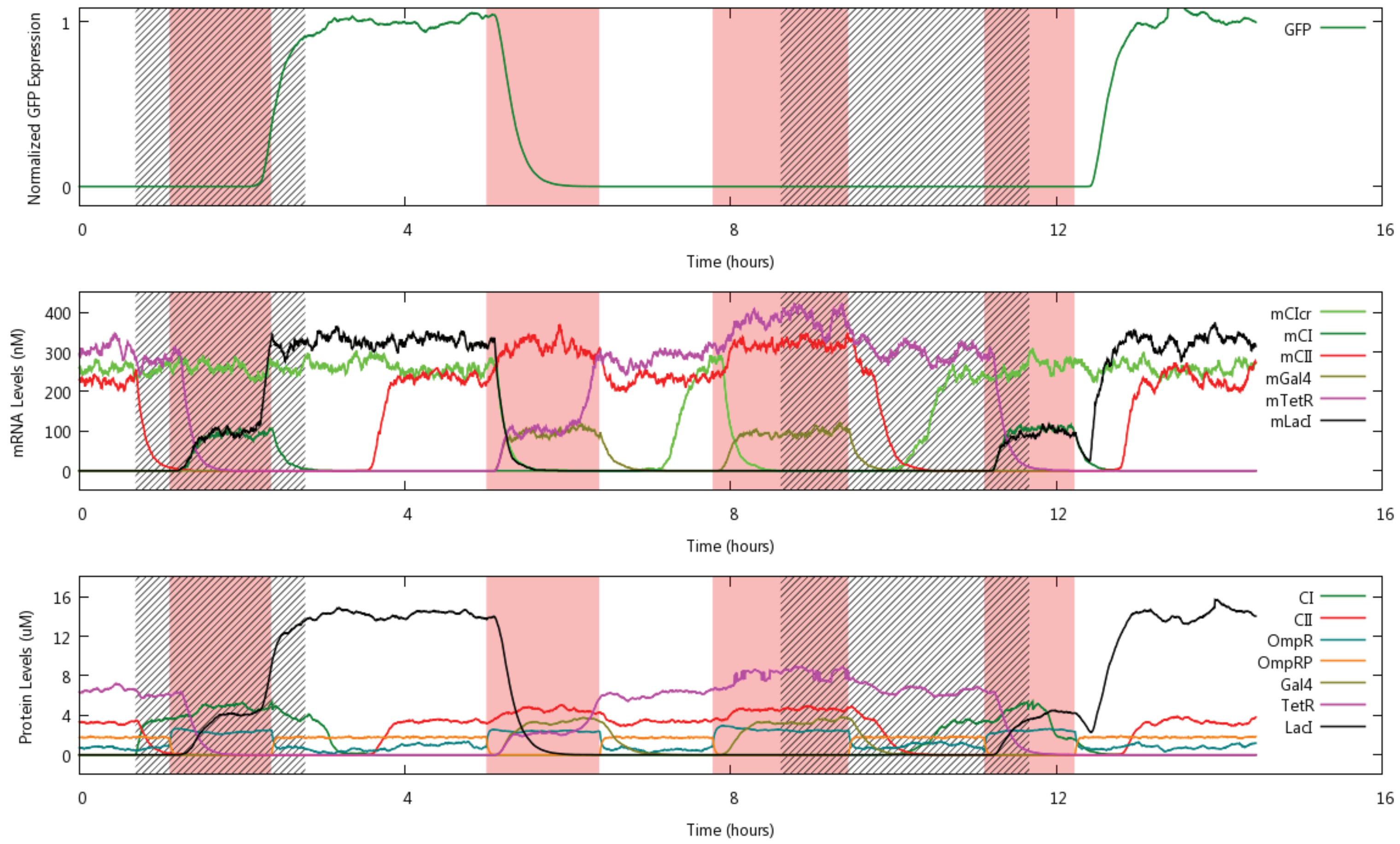


Figure 4-1. Stochastic simulation of *BioD*.

The three timing diagrams are displaying different signals of the same run. The highlighted areas indicate the presence of an input. The red hue indicates the *CLK* signal (FR light). The grey diagonal pattern indicates the presence of the input *D*. **a.** Normalized GFP expression **b.** mRNA levels **c.** Protein levels.

Ideally, with four separate *CLK* inputs, the state of the device should follow the *D* input four times. In this case, the state should turn ON, then OFF, and then OFF again and finally ON. Figure 4-1a displays those exact state changes in a stochastic run whose initial condition is an OFF state. The normalized *GFP* expression output follows the input only at the rising edge of the clock. However, while the clock is ON or is OFF, any changes in the input do not propagate to the output. Figure 4-1b shows the changes in the concentrations of the mRNA transcripts of the various substances involved. Please note that the concentration level of *mOmpR* is not displayed because this transcript is constitutively expressed. Figure 4-1c shows the changes in the protein levels; the levels of *LexA* and *GFP* were not displayed because they do not affect the behaviour of the device. Changes in protein concentrations follow changes in corresponding mRNA concentrations, except in situations where post-transcriptional regulation is in effect. In particular, when *mCIcr* is expressed in the absence of input *D*, the level of the *cI* repressor does not subsequently increase. Because of this highly correlated relationship between transcript and protein, the protein levels are not shown for the rest of the examples. Rather, the *GFP* figure is used to demonstrate the overall input/output relationship.

The concentration of a molecule is decided, mainly, by its rates of synthesis and degradation. Some transcripts have multiple stable levels of expression. Since `cI`, `cII`, `LacI` and `TetR` are not *only* produced by the SLECTION genes (but can also be produced by some of the INPUT or STATE genes) the production of their transcripts is significantly increased in the presence of the *CLK* signal. `mTetR` has four levels of expression: (i) all the genes that can produce it are OFF, (ii) gene 6 is ON, (iii) gene 5 is ON, and (iv) genes 5 and 6 are ON. `mLacI` has similar multiple levels of expression, using genes 4 and 7. In the case of `mCI`, however, since gene 4 can only turn ON when gene 1 is ON, it only has three levels of expression. The case of `mCII` is analogous to that of `mCI`.

Tracing the various signals in Figure 4-1b shows that, the simulation starts with three active transcripts, `mTetR` (the state of the device is OFF), `mCIcr` (unrepressed since the *CLK* and therefore `Gal4` are OFF) and `mCII` (unrepressed since input *D* is OFF). Following, is a step-by-step explanation of the changes shown in the timing diagram (Figure 4-1b).

First, input *D* is introduced, causing the repression of gene 2 (or `mCII`). Since the transcript of gene 1 is translated and gene 2 is OFF, gene 4 is on a hair-trigger to be turned ON, while gene 5 is doubly repressed by `OmpRP` and `cI`. The *CLK* signal is introduced, stopping the phosphorylation of `OmpR` and activating gene 4. This raises the level of `mCI` and `mLacI`. The latter represses gene 6 and starts turning the state of the device ON. As *TetR* degrades, *GFP* levels increase. Then, the *CLK* signal is turned OFF followed by input *D*. These two actions turn OFF gene 4 and disable gene 1, respectively. With both inputs OFF, the *cI* repressor produced by genes 1 and 4 degrades without replacement, allowing *mCII* to return to its previous level. *mLacI,* which is now produced by gene 7, reaches its unrepressed (ON) state equilibrium.

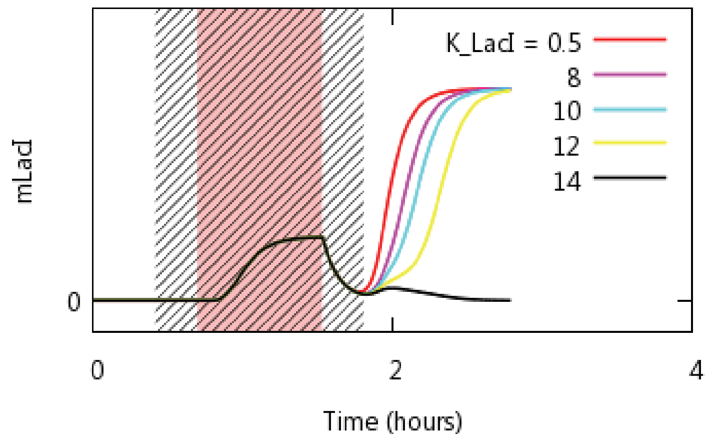
The second state change occurs when the *CLK* signal is turned ON again. Since *mCII* is expressed at that time (no input *D*), gene 5 turns ON, causing the repression of gene 1 (through *mGal4*), the repression of gene 7 (through *mTetR*), and an increase in the level of *mCII* (as it is produced by both genes 2 and 5). When the *CLK* is removed, gene 5 is turned OFF, but *mCII* and *mTetR* remain high, while *mGal4* is repressed. This allows the production of *mCIcr* to start again (after *Gal4* degrades). Note, however, that *mTetR* is now produced by gene 6, and not by gene 5.

The third *CLK* signal starts now. Gene 5 is again turned ON; the levels of *mCII*, *mGal4* and *mTetR* climb; the level of *mCIcr* drops (repressed by *Gal4*). In the middle of the *CLK* pulse, input *D* is introduced. This causes no change in the network. Since input *D* only affects gene 1, its effects are muzzled because the clock has already turned on gene 5 which repressed gene 1. It is only after the clock is turned OFF that the repression of gene 1 is lifted. At this point, even though the *CLK* signal is removed, input *D* is still present, and since gene 1 is no longer repressed by gene 5 (or *Gal4*), *cI* is synthesized, which proceeds to represses gene 2. The state of the device, however, does not change since the STATE genes are not directly affected by the INPUT genes.

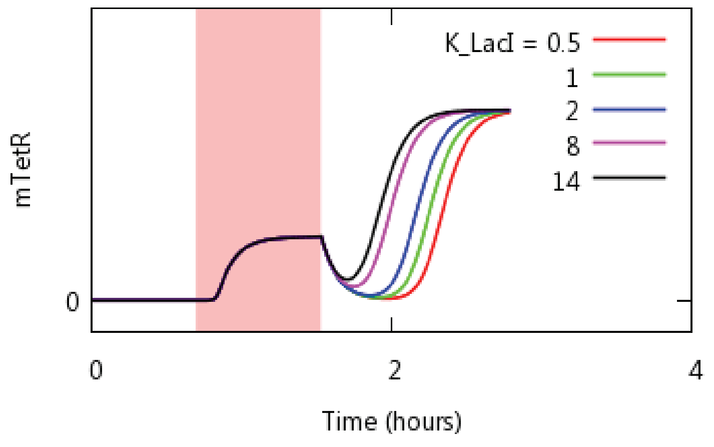
The fourth *CLK* signal turns the state of the device back ON. In the presence of input *D*, the *CLK* turns gene 4 ON causing a similar sequence of events to the one witnessed following the first *CLK* signal.

4.1.1 Model Constraints

An important factor in the design of any gene network is the choice of regulatory sequences, promoters and coding sequences, which make up the various genes. The specific genes used for the realization of the *BioD* are just an example, meaning that other genes can be used to realize the logical design (shown in Figure 3-3) of the *BioD*. It must be noted that any alternate set of genes will very likely have a different set of model parameters. The variation of these parameters changes the behaviour of the network, possibly making it faster or slower in responding to the inputs or in reaching a steady state.



a.



b.

Figure 4-2. Effect of Dissociation constant on input response time.

The *BioD* network described above is left unchanged except for one variable, K_{LacI}. Its effect on the input response time is highlighted for two complementary genes, *LacI* and *TetR*. **a.** Increasing the K_{LacI} value from 0.5nM to 14.0nM increases the time it takes to start production of the *mLacI* transcript in response to the proper input sequence. **b.** As expected, increasing the K_{LacI} value has the opposite effect on the production of the *mTetR* transcript.

In more detail, any gene chosen or constructed for the *BioD* comes with a set of parameters: the dissociation constant K_d reflects the affinity of a repressor binding to its operator site; the Hill coefficient n_d reflects the cooperativity of repression of the constituent molecules of a multimer; the degradation rate ω_d depends on the chemical and spatial properties of the substance but can be modified using certain well-studied methods (such as the addition of an SsrA tag to speed-up degradation).

In a network where two genes repress each other (such as the two STATE genes), a small increase in the dissociation constant (K_d) of one of the two repressors, affects the network's response time to the input in two separate ways: (i) it significantly reduces the response time of the target gene, and (ii) it increases the response time of the gene that produces it. The *state genes* are used to illustrate this issue.

We chose to record the effect of separately varying K_{LacI} on the dynamic behavior of $mLacI$ and $mTetR$. Figure 4-2a illustrates the effect of changing K_{LacI} from 0.5 nM to 14.0 nM on $[mLacI]$, leading - or not - to a change of state from OFF to ON. Similarly, Figure 4-2b illustrates the effect of changing the value of K_{LacI} on $[mTetR]$, leading to a change of state from ON to OFF.

Generally speaking, the K_d value is not the only parameter defining a repressor, nor can this value be changed at will, because it is dependent on the chemical and conformational properties of both the repressor and its corresponding binding site. Therefore, any change in one gene's parameters might have effects beyond those intended. This must always be taken into consideration during design or optimization of gene regulatory networks.

It is noteworthy that in Figure 4-2a, there is one K_d value where the expected change of state fails to happen. This occurs because the *CLK* signal becomes too short for the state change to occur at this K_d value. A more detailed discussion of the relationship between K_{TetR}, K_{LacI} and the *CLK* signal is provided in the following section 4.1.4.

4.1.2 Clock Input (CLK)

When the input and output states are at opposite levels, the length of the *CLK* signal must be large enough to allow a change of state to occur. As an example, when input = ON and output = OFF, the *CLK* signal has to be sustained for a time greater than the minimum time needed for the cell concentration of $mTetR$ (or $[mTetR]$) and for $[TetR]$ to degrade below $[mLacI]$ and $[LacI]$, respectively. If the *CLK* signal is removed too soon, the production of $mLacI$ from the SELECTION gene 4 is cut too quickly. The output responds to its short presence and reduces the production of $mTetR$, seemingly heading towards a state change. However, when the *CLK* signal is removed, the $mTetR$

production is simply reasserted, because gene 7 has not yet begun the production of *mLacI*, and the state of the device fails to toggle. As seen in Figure 4-3a, the *GFP* levels do not rise even though input *D* was present at the rising edge of the clock. In point of fact, the *CLK* signal enabled the transcription of *mLacI* from gene 4 (which is not repressed by *TetR*). This causes the levels of *TetR* to fall rapidly. However, the *CLK* signal is removed before they could fall low enough to turn gene 7 ON. Gene 4 is then turned OFF on the *CLK*, and gene 6 is reasserted. This situation explains the need for the *CLK* signal to remain active until the target STATE gene is activated.

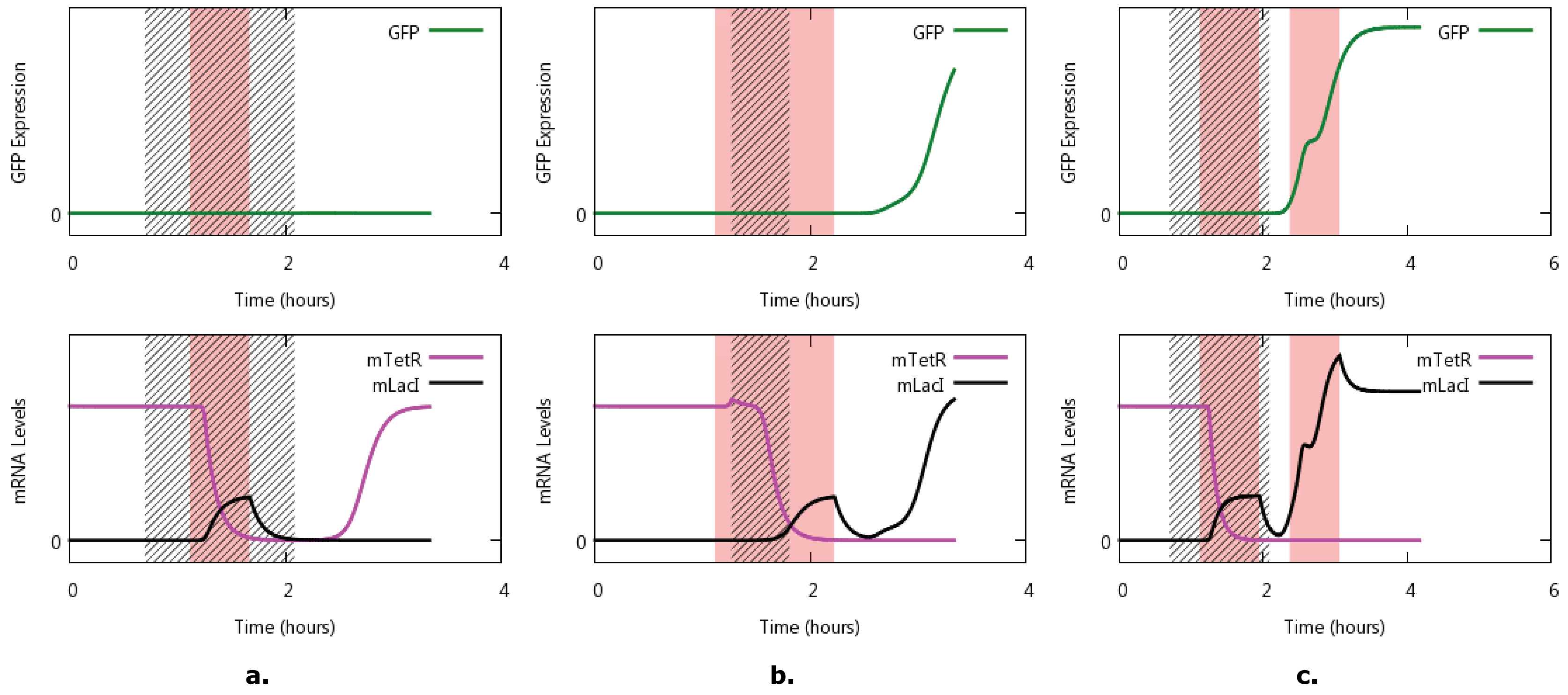


Figure 4-3. Constraints and failures.

The graphs show only the `GFP` and the transcripts of the STATE genes. **a.** The clock pulse is too short. The state of the *BioD* does not have enough time to change **b.** Input *D* introduced shortly after the clock turns the state of the output ON. **c.** A clock pulse that occurs *shortly* after the end of the input *D*, acts as if input *D* was still ON, resulting in a change of state.

4.1.3 Data Input (D)

The data input (*D*) introduces two more timing constraints. The first prohibits the introduction of the input too soon after the start of the *CLK* signal. While this might seem odd, it is in fact consistent with network behaviour. Since gene 1 is only repressed by *Gal4*, it can only be repressed when the clock is ON. Therefore, when the clock is OFF, gene 1 is not prevented from continuously transcribing *mCIcr*. Since translation on its own is faster than transcription followed by translation, when input *D* is introduced, it quickly induces the translation of *mCIcr*, now unlocked. During that time, the *CLK* signal selects gene 5, but before *Gal4* has had a chance to be transcribed and then translated, the direct translation of the transcript of gene 1 into its corresponding (repressor) protein causes the repression of *cII* (by way of genes 2 and 5) and hence, the activation of gene 4. This ultimately results in an erroneous change of state as illustrated in Figure 4-3b.

The second timing constraint occurs when input *D* is turned OFF. Indeed the level of expression of protein *cII* does not climb immediately. Time is needed to allow for the degradation of the *cI* protein, the *taR12* molecule, and the unlocked mRNA molecule that are still in the system, in order to stop the production of more *cI* and allow the production (transcription and translation) of *cII*. Figure 4-3c shows the *CLK* signal being activated too soon after input *D* is turned OFF. Since the system has not had enough time to reach equilibrium, it reacts to the clock as if its input was still ON.

The clock pulses must be sufficiently apart to allow the system to go to equilibrium (steady state) before the next pulse. Which SELECTION gene gets enabled depends heavily on that. Essentially, the input signal must stabilize (as ON or OFF), then the levels of *cI* and *cII* must stabilize as well, allowing the selection of one of the SELECTION genes, which must occur prior to the start of the clock pulse.

4.1.4 Bi-stability

A necessary feature of the *BioD* is its bi-stability. Bi-stability means that the network is capable of being in any one of two steady states for as long as the inputs remain unchanged. This is a crucial feature because we do not want a *BioD* that is in (say) an ON state to autonomously switch to the OFF state, without any prompting from its input. Furthermore, we want these two steady states to be stable. Dynamically, a *stable* steady state is a basin of attraction with all nearby trajectories leading into it. In other words, the effect of small, non-sustained and/or noisy perturbations in the inputs are absorbed and do not prevent a return to the original stable steady state. This does not only apply to the STATE genes, but also to the SELECTION genes (which also form a toggle switch).

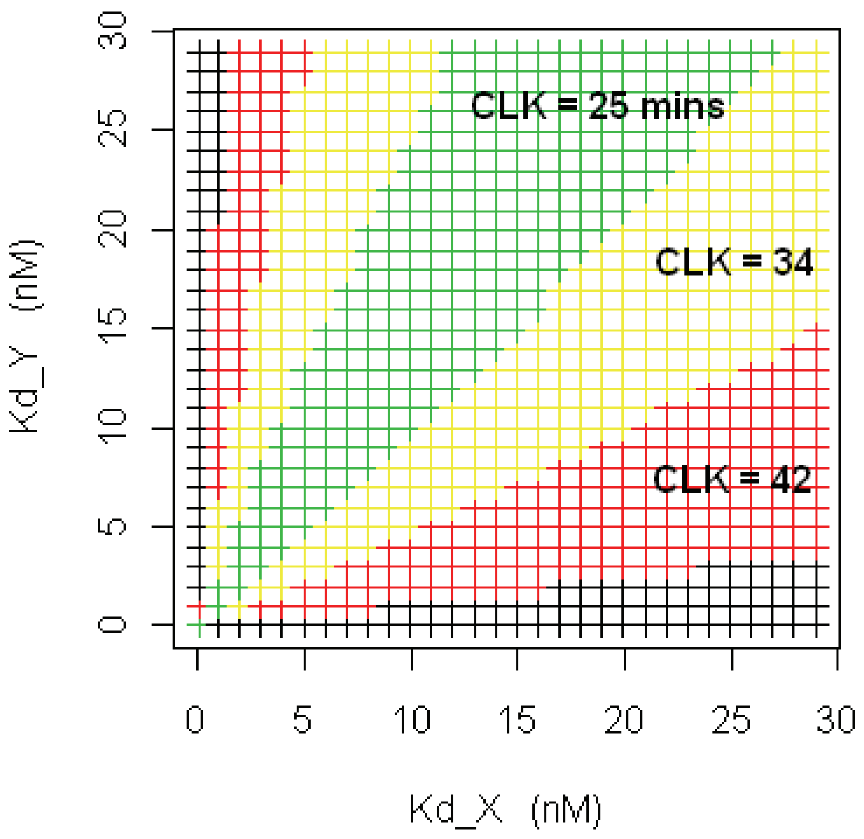


Figure 4-4. Bi-stable region relative to *CLK* pulse width.

Varying the K_d values of the toggle switch genes while keeping all other parameters constant results in the above functional plot of the *BioD*. The *BioD* is said to be bi-stable (or functional) when it is able to toggle from one state to the other on the right inputs and is able to hold on to that state indefinitely if unperturbed. The green zone, which is included in the yellow zone, which itself is included in the red zone all define the bi-stability regions of the *BioD* at *CLK* pulse widths of 25, 34 and 42 minutes, respectively. The black region denotes results of simulations that did not lead to a bi-stable network.

The conditions for toggle switch bi-stability have been discussed by Gardner *et al.* [2], asserting that (i) the gene products must have a cooperative repression of transcription (Hill cooperativity) that is greater than 1; (ii) the rates of synthesis of the two repressors must be balanced (approximately equal). According to Gardner *et al.*, these two conditions decide the size of the bi-stability region; where larger cooperative repressions and larger synthesis rates result in larger bi-stability areas. We add to these findings by including the effect of our *CLK* signal in relation to the genes used in the network. The results of our investigation resulted in a delineation of the region of bi-stability identical in general shape to the one discovered by Gardner *et al.*, but having different exact boundaries.

In more detail, we varied the two K_d values of the two STATE genes as well as the length the *CLK* pulse, while keeping all other parameter values constant. For every pair of K_d values, we sought a minimum *CLK* pulse width that would result in a bi-stable network. In some cases, we found it, such as the green, yellow and red regions of Figure 4-4, but in others – the black area – we did not. It is worth noting that for all of these regions – except the black one – a clock pulse whose length is equal or *greater* than the noted values would ensure a bi-stable behaviour.

As can be seen, a smaller *CLK* pulse significantly reduces the range of K_d values (and hence potential genes) that can used to construct a bi-stable *BioD*. Extending the length of the *CLK* pulse too much, however, would not only be highly impractical, but would also mean that the state change is occurring across multiple reproductive cycles of an *E. coli* cell. It is therefore important to balance speed, practicality and the absolute need for bi-stability.

4.1.5 Dissociation Constants

The dissociation constants, or K_d values, measure the propensity of a complex molecule to separate (or dissociate) reversibly into its component molecules. The vast majority of reported values for the dissociation constants of some well known transcription factors were unrealistically low. This issue becomes quite apparent when the values are investigated.

Substitution of various K_d values found in literature (for $LacI$, $TetR$ and cI) in equation (4.1) yields concentrations of far less than one molecule per cell. In other words, K_{TetR} = 179 pM (Weber et al., 2007) means that seven hundredth (0.07) of a $TetR$ molecule in a cell would somehow be enough to repress half its operators. Arguing it further, and rounding the number of $TetR$ molecules up (then multiplying it by 4) and assuming four repressor molecules existed in the cell, the probability of them colliding with each other to form a tetramer is negligibly small. More to the point, the probability that a single tetramer in the entire cell would collide with and bind to the operator site is effectively zero, making these numbers quite problematic.

If the complex molecules in question is made up of one ligand molecule and one receptor molecule only (as is frequently the case) then K_d is also defined as the ligand concentration at which half the receptors are occupied at equilibrium. This allows a meaningful conversion of K_d from moles-per-liter to molecules-per-cell (K_d now becomes the number of ligand molecules per cell at which half the receptors are occupied). This is achieved as follows

$$\frac{Molecules}{Cell} = Molarity * N_A * V_{cyto} \tag{4.1}$$

where $V_{cyto} = 6.7 * 10^{-16}$ liters is the *E.coli* cytoplasm volume and is taken from the *CyberCell* Database (CCDB) (Sundararaj et al., 2004), and where $N_A = 6.022 * 10^{23}\ \mathrm{mol}^{-1}$ is the Avogadro constant.

The 2009 University of Aberdeen iGEM team illustrates a method to generate more realistic K_d values estimations. The method uses known repressor molecule numbers present in the cell at a given state, to extrapolate the number of repressor molecules needed to halve the overall production of its target gene (in this case of repressors and operator sites). In most cases, this number would be a better approximation of the real K_d value.

4.2 BioFSM

As previously described, the *BioFSM* is built by connecting the *BioD* to the UF/II and OI. Of these, the OI is the simplest module. It is in effect just an inverter that uses $\overline{Q}$ from the *BioD* to generate an AHL version of Q that is meant for inter-cellular signalling.

When $LexA$ is ON ($\overline{Q}$ is ON) the AHL production is stopped; while when $LexA$ is OFF (Q is ON) AHL production is resumed. The UF/II is a variable module whose complexity depends on the desired functionality of the *BioFSM*. It can be as simple as the OI inverter or it can be an elaborate network that handles numerous inputs and performs complex combinatorial logic.

Figure 4-5 and Figure 4-6 display the stochastic simulations of all eight possible inputs to two UF/IIs implementing $F = A + BC$ and $F = A\bar{B}\bar{C} + \bar{A}B + \bar{A}C$, respectively. The output F of the UF/II is the input *D* (or $taR12$) to the *BioD*. The inputs A, B (or $\bar{B}$) and C of the UF/II are the AHLL, $LacI$ (or $TetR$), and AHLR, respectively. The core functionality of our *BioFSM* hinges on the manipulation of the input to the *BioD* incorporated within the *BioFSM*. We therefore highlight the proper functionality of the UF/IIs that provide these inputs.

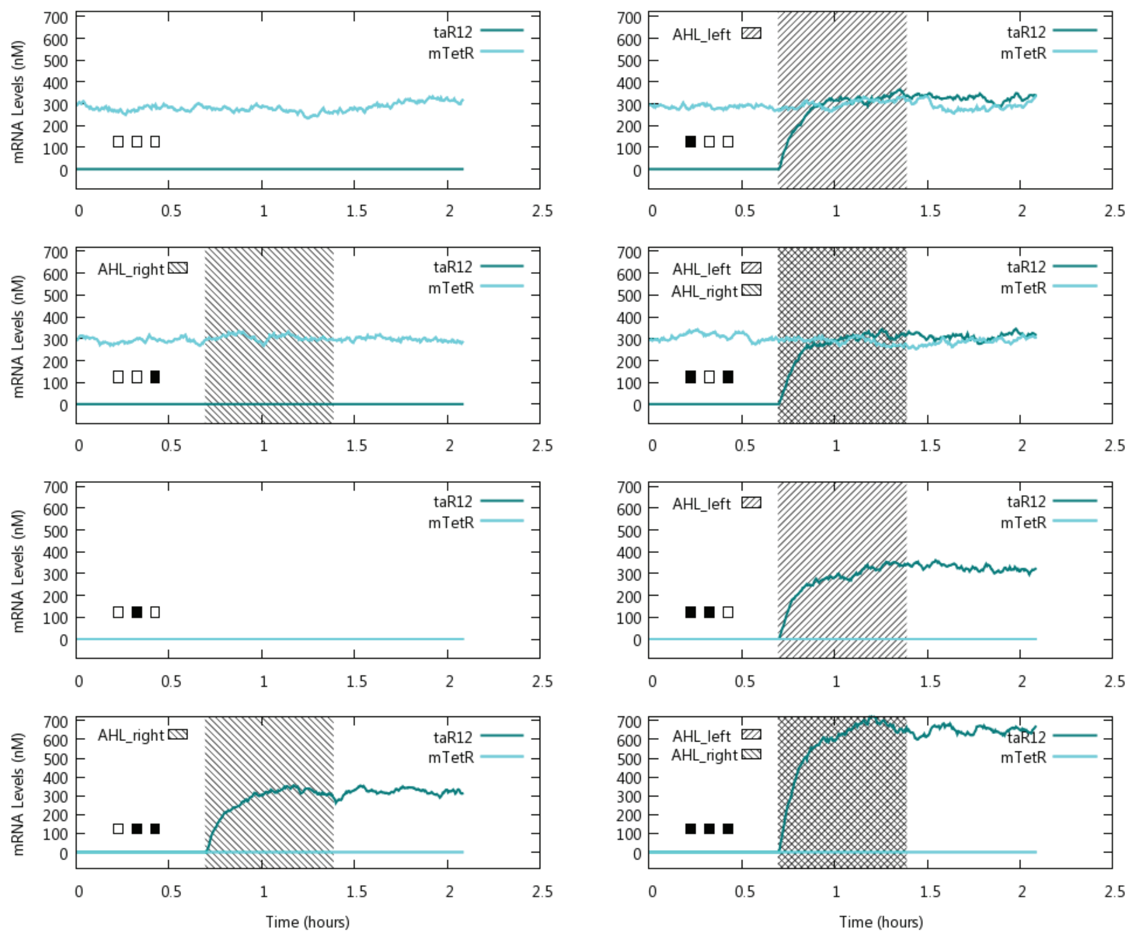


Figure 4-5. Stochastic simulation of all 8 possible inputs to function $F = A + BC$ (or *Rule 248* (Wolfram, 2002)).

The external inputs A and C are AHL_left (AHLL) and AHL_right (AHLR), respectively. Their presence is highlighted by the grey diagonal patterns. The internal input B comes from the *BioD*. In this case, $\overline{B}$ was needed for the implementation of the UF (see Figure 3-5b), so *TetR* was used and its respective mRNA level is displayed.

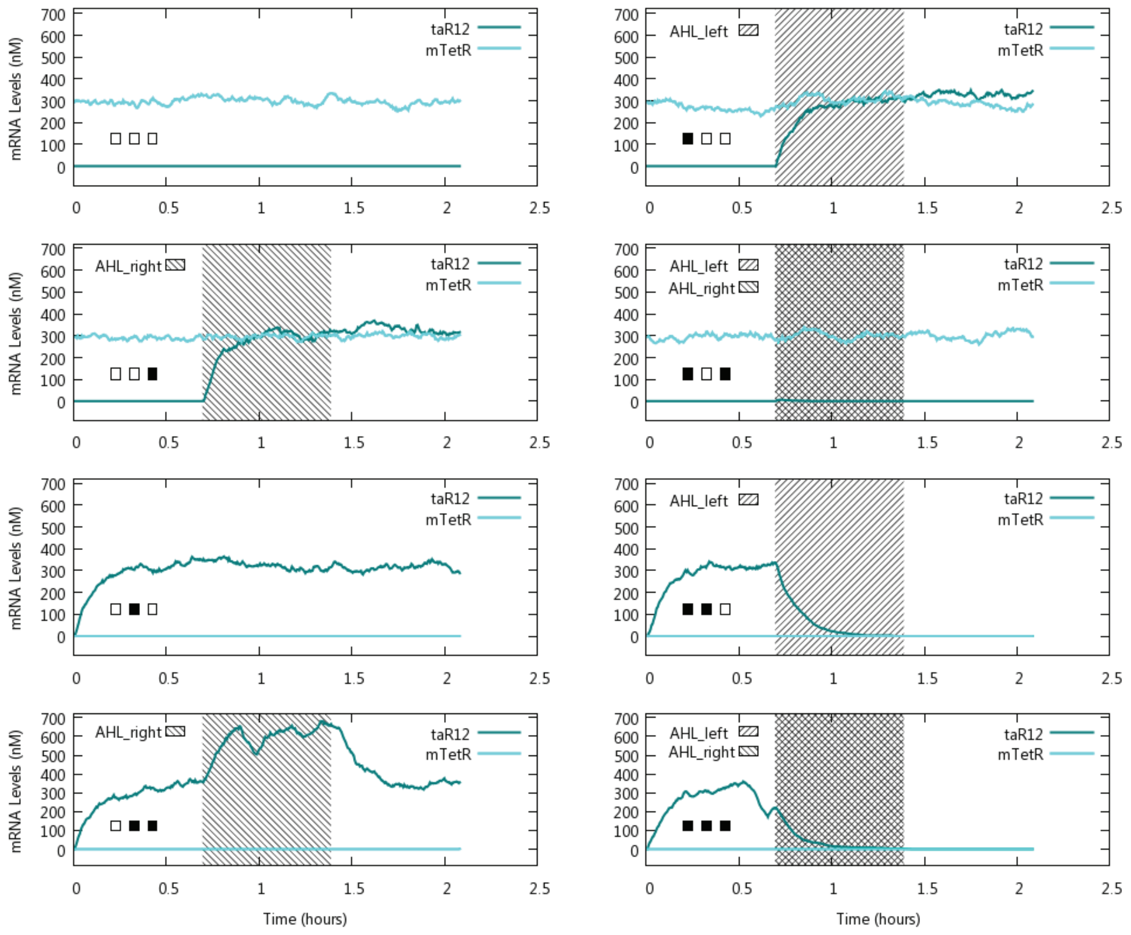


Figure 4-6. Stochastic simulation of all 8 possible inputs to function $F = A\overline{B}\overline{C} + \overline{A}B + \overline{A}C$ (or *Rule 30* (Wolfram, 2002)).

The external inputs A, B and C are defined and highlighted as above. Note that in the $ABC = 011, 110$ *and* 111 cases, the output F is affected before the introduction of either AHL. This is because before the introduction of the AHLs, all these cases are in effect $ABC = 010$ and in *Rule 30*, this input results in $F = 1$. This behavior is not unwanted because the design of the UF/II is asynchronous. As can be seen, in the first case, F is doubly asserted when AHLR is introduced. While in the other two cases, F is turned OFF when the AHLs are present. This flexibility insures that the *BioD* always receives the most updated input from the UF/II, regardless of the *CLK* signal.

Here, we have one internal and two external inputs. The highlighted areas indicate the presence of an external input, while $mTetR$ reflects the internal input $\overline{B}$ (= $\overline{Q}$). The two opposing diagonal patterns reflect the presence of the left and right inputs A and C.

Eight stochastic simulations are provided, covering every possible UF/II input permutation (given three inputs, one internal and two external). These simulations are presented as a table of diagrams, sorted by input presence, top to bottom, starting with the left column. We say $A = 0$ when AHLL is not present, while $A = 1$ means that AHLL is present in high quantities. Similarly, the values of B and C denote the presence and absence of $LacI$ and AHLR respectively. As previously described in *BioD*, in our design $LacI$ and $TetR$ are complimentary signals, which is why we consider $\overline{B} = TetR$, and why we used it in the diagrams below.

The top left diagram displays the UF/II level at input $ABC = 000$ while the bottom right diagram displays that level at input $ABC = 111$. The top left diagram has no highlighted areas (i.e. no diagonal patterns) denoting the absence of the external inputs (AHLL and AHLR). $mTetR$ is present however, meaning $\overline{B} = 1$ (or $B = 0$), denoting the absence of $LacI$. Hence, this diagram displays the value of the UF/II, namely $F = A + BC$, with zero inputs, which is zero itself. The bottom right diagram has two areas of diagonal patterns (overlapping) denoting the presence of the external inputs. $mTetR$ is absent meaning $\overline{B} = 0$ (or $B = 1$), denoting the presence of $LacI$. Hence, this diagram displays the value of the UF/II, namely $F = A + BC$, with all inputs present. It is in fact doubly asserted by both A and BC and results in a higher production of $taR12$ (representing F) than the other cases where is it asserted; at $ABC = 011, 100, 101, and\ 110$. The value of the input is highlighted in each diagram by three little squares in the bottom left area.

The UF/II module is designed as a non-synchronous module, but the *BioFSM* still functions synchronously using the embedded *BioD* clock.

4.3 Extension: *BioCell*

Using multiple strains of *BioFSM*s connected in sequence to build circular cellular automata (CA), or *BioCell*.

A *BioCell* is a ring of *N* colonies of *E. coli*. Each colony consists of clones of one of three strains, genetically modified to realize a *BioFSM*. The three strains implement the same logical functionality (same *BioD* and UF rules) but have different input and output interfaces (for inter-colony communications). We chose to connect these *BioFSM*s as a ring cellular automata, i.e. each *BioFSM* is connected to its left and right

BioFSM neighbours only (see Figure 4-7). In effect, each colony will implement one type of *BioFSM*, and will communicate with its neighbour colonies via AHLs.

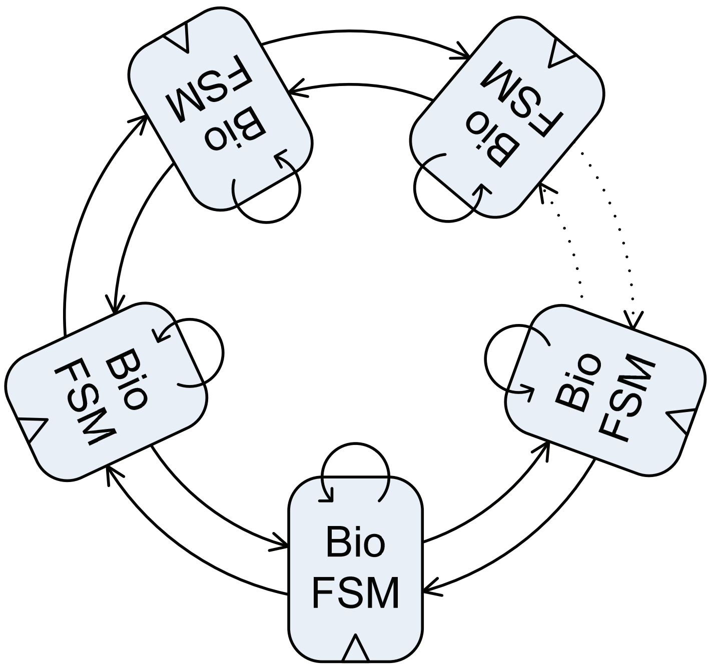


Figure 4-7. The Logical Block Diagram for *BioCell*.

Therefore the UF must have three inputs (two from its immediate neighbours, and one from itself). Each strain (*BioFSM*) needs to be able to recognize the origin of its inputs (to the UF/II), and to broadcast a recognizable output (from its OI). The left-hand strain produces AHLL and responds to AHLC and AHLR, while the centre strain produces AHLC and responds to AHLL and AHLR, and so on. In order to function as expected a colony processes its inputs to decide whether to alter its state, upon the application of a global clock pulse (FR light). The decision to change the state is made following the rules implemented by the UF. Those rules are the same for all strains, though with variations merely reflecting the chemical nature of the inputs with which each strain is confronted. A colony exhibits its state by expressing (or not) a florescent protein.

The *BioCell* will have the following dynamic behaviour, determined by its inputs states of its *N* colonies (collectively making up the *BioCell*'s state). When there is no FR light (i.e. $CLK = 0$), there is no change in the state of the *BioCell*. In contrast, when the device receives a FR light pulse (i.e. $CLK = 1$) applied to all the colonies simultaneously, the next states of the colonies follow the outputs of their UF/II (by processing its own and the neighbours' states). The next clock pulse has to wait until the *BioCell* is back in equilibrium. Equilibrium, after an OFF to ON state change, comes after the colonies have had a chance to produce enough AHLs and after those AHLs have diffused to the neighbours. Equilibrium, after an ON to OFF state change, comes after the AHLs produced by the colonies have had a chance to degrade. This is critical because the AHLs are the only indicator of the neighbours' states. A clock pulse that comes before equilibrium might cause an erroneous change of state of the *BioCell.*

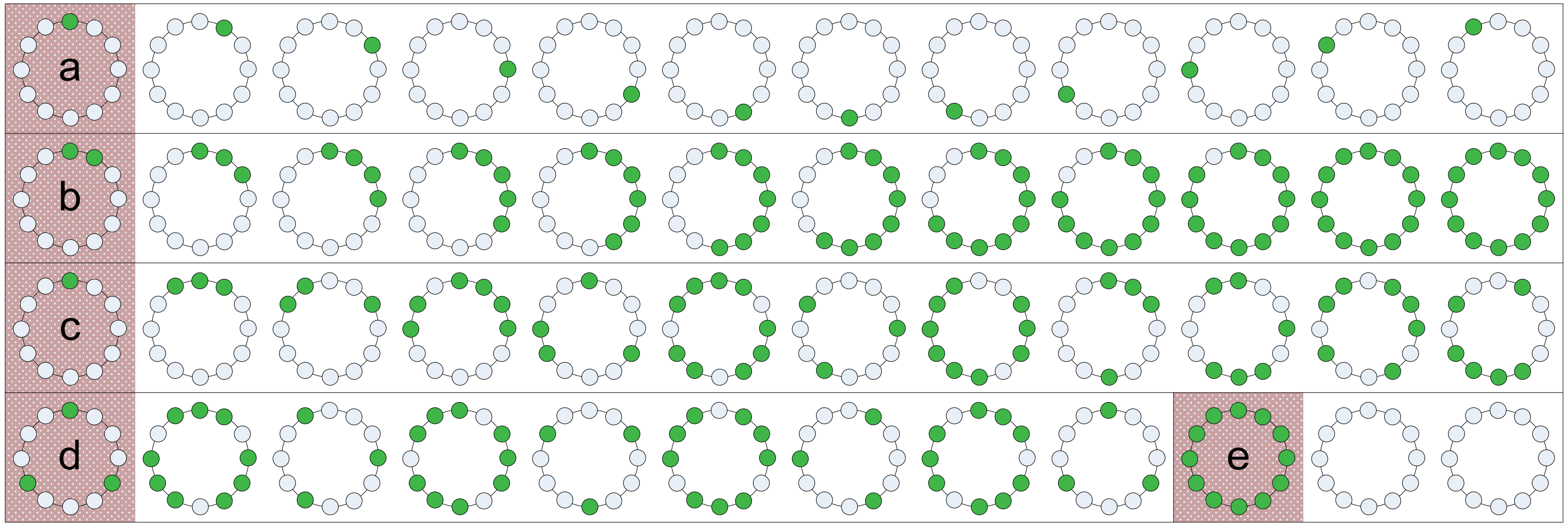


Figure 4-8. Ring topology CA run of two rules with changing initial conditions.

Rule 248 with two different initial conditions: **a.** demonstrates signal propagation and **b.** exhibits counting. *Rule 30* is shown with three different initial conditions, resulting in: **c.** chaotic behaviour, **d.** cyclical behaviour and **e.** fixed behaviour

From a computational point of view, the *BioCell* device is a synchronous ring of cellular automata implemented as a ring of *N* communicating colonies of three new strains of *E. coli*. Given this setup (three binary inputs and one binary output) for every *BioFSM* in the *BioCell*, there exists $2^{2^3} = 256$ possible functions (or rules) that can be implemented by the UF. We chose two such rules to implement: (a) *rule 248*, as defined by Wolfram (Wolfram, 2002), allows us to demonstrate signal propagation and counting behaviours, depending on the initial state of the ring; (b) *rule 30*, can be used as a *pseudo random number generator* or to exhibit *cyclical* behaviour, depending on the initial state of the ring.

Some of the power of cellular automata is emphasized when rules exhibit different dynamic behaviours, i.e. chaotic, cyclical or fixed, by merely varying the initial conditions of the cellular automata. The particular UF used when introducing *BioFSM* above, implements *rule 248*. Figure 4-8 (a, b) displays runs of this rule on a *BioCell* of 12 colonies. The change in the initial state results in two different behaviours, namely signal propagation in (a) and counting in (b). *Rule 30*, whose runs are displayed in Figure 4-8 (c-e), is an interesting rule that can result in either chaotic behaviour as in (c), various cyclical behaviours such as (d) or simply lead the ring to a fixed state, as in (e).

This is a device that can be configured to perform many different functions using simple or no modifications (via change in initial conditions). Many cellular automata are capable of universal computation (Wolfram, 2002).

CHAPTER 5.

CONCLUSION

In this thesis, we present a mathematical model and simulation results of a synchronous single-input delay flip-flop, realized as a gene regulatory network for implementation in *E. coli*. The simulation we present provides evidence that the device can toggle from the ON state to the OFF state and back, according to its intended functionality. The inherent symmetry of the design reduces the number of genes used, but introduces some complexity, which is palpable when tracing the various changes the device goes through when toggling.

The *BioD* is effectively a 1-bit memory element that can operate synchronously with any number of other elements. As such, it can be used to hold the state of a finite state machine, as it does in the *BioFSM*. It could also be used to build a memory bank, an event sequence detector/effector, a decision-making system, and numerous other memory-requiring devices. The *BioFSM* is made of three modules: the *BioD*, the Update Function/Input Interface (UF/II) and the Output Interface (OI). The modular design of the *BioFSM* allows us to hold the *BioD* constant while changing the UF/II or/and OI, if and when the time-dependant behaviour of the *BioFSM*, or its input/output interfaces require alteration. Then, there is the *BioCell*, which is made of a number of *BioFSM* colonies, and is capable of exhibiting a large number of computational, communicational and pattern formation behaviours depending on the particular UF and/or initial states of its constituent *BioFSM*s.

Speed is a main area of improvement. Indeed, the slowest reactions in a cell are the ones involving regulated transcription and translation. The time it takes to execute these operations depends on many factors, including various binding affinities, generation and degradation rates. For example, the impact of a repressor is delayed until a mature protein is formed and manages to interact with its corresponding operator site on the DNA. Using post-transcriptional regulation like *ta*RNA or RNA interference (RNAi) - where possible - to affect regulation in the *BioD* will make the system significantly faster. One possible location for such a change would be where the SELECTION genes interact with the STATE genes. Instead of producing repressors for genes 6 or 7, the use of RNAi molecules to prevent the translation of repressor proteins would make the entire system significantly faster. However, since we already make use of `taR12` for input sensing, we would have to use two more riboregulators that do *not* interfere with `taR12` or with each other.

Another notable property of genetic networks is that the building blocks tend to vary significantly from one another, whether they be promoters, operators, or coding sequences – to name a few. That is to say, when designing a gene regulatory network, the choice of the building blocks is not easily exchangeable. In fact, the literature does not provide much in the way of “acceptable ranges” because most networks are presented as they are. In the case of dynamic and extendable circuits like *BioD* or *BioFSM*, that need is reasserted. Gene networks constitute highly interconnected graphs such that, for example, a repressor contributes to the functioning of the designed network by means of its dissociation constant (for a given operator), its rates of synthesis, diffusion and degradation, as well as the possibility of unintended (and often unexpected) cross-talk with the native DNA and constitutively generated molecules. We attempted to provide such “ranges” for our design (= constraints), and identified failure points and tendencies that help greatly when selecting different genes (or parts thereof) to ensure correct performance of the *BioD*. However, we admit that much more work can and should be done in that area to provide standardized sets of devices, information sheets and design approaches for future gene regulatory networks.

In fact, there are two main areas that need to be standardised in order to design and/or implement and successfully replicate gene regulatory networks: (i) standard building blocks and (ii) standard and complete parameters (or measurements) relating to the building blocks.

A successful and growing database of biological parts was started in 2003 at MIT is called the “Registry of Standard Biological Parts” (Knight et al., 2004). This is a database that grows yearly with new simple or complex parts, designed following a standard framework.

A database that includes the parameters needed for simulation of every gene is yet to be realized however. The discrepancies in simulation methodologies and in results have become too pronounced for meaningful claims on complicated network designs. A standardized parameters’ database is required to deal with those discrepancies.

References

Ordered by: Author/Date

Adleman,L.M. (1994). Molecular computation of solutions to combinatorial problems. Science *266*, 1021-1024.

Aksenov,S.V. (1999). Induction of the SOS Response in Ultraviolet-Irradiated Escherichia coli Analyzed by Dynamics of LexA, RecA and SulA Proteins. Journal of Biological Physics *25*, 263-277.

Alon,U. (2006). An Introduction to Systems Biology Design Principles of Biological Circuits. (London: Chapman & Hall/CRC/Taylor & Francis).

Anderson,J.C., Clarke,E.J., Arkin,A.P., and Voigt,C.A. (2006). Environmentally controlled invasion of cancer cells by engineered bacteria. J. Mol. Biol. *355*, 619-627.

Anderson,R.M., Zimprich,C.A., and Rust,L. (1999). A second operator is involved in Pseudomonas aeruginosa elastase (lasB) activation. J. Bacteriol. *181*, 6264-6270.

Balagadde,F.K., Song,H., Ozaki,J., Collins,C.H., Barnet,M., Arnold,F.H., Quake,S.R., and You,L. (2008). A synthetic Escherichia coli predator-prey ecosystem. Mol. Syst. Biol. *4*, 187.

Basu,S., Gerchman,Y., Collins,C.H., Arnold,F.H., and Weiss,R. (2005). A synthetic multicellular system for programmed pattern formation. Nature *434*, 1130-1134.

Basu,S., Mehreja,R., Thiberge,S., Chen,M.T., and Weiss,R. (2004). Spatiotemporal control of gene expression with pulse-generating networks. Proc. Natl. Acad. Sci. U. S. A *101*, 6355-6360.

Baumeister,R., Flache,P., Melefors,O., von,G.A., and Hillen,W. (1991). Lack of a 5' non-coding region in Tn1721 encoded tetR mRNA is associated with a low efficiency of translation and a short half-life in Escherichia coli. Nucleic Acids Res. *19*, 4595-4600.

Bayer,T.S. and Smolke,C.D. (2005). Programmable ligand-controlled riboregulators of eukaryotic gene expression. Nat. Biotechnol. *23*, 337-343.

Becskei,A., Seraphin,B., and Serrano,L. (2001). Positive feedback in eukaryotic gene networks: cell differentiation by graded to binary response conversion. EMBO J. *20*, 2528-2535.

Boyle,P.M. and Silver,P.A. (2009). Harnessing nature's toolbox: regulatory elements for synthetic biology. J. R. Soc. Interface *6 Suppl 4*, S535-S546.

Camas,F.M., Blazquez,J., and Poyatos,J.F. (2006). Autogenous and nonautogenous control of response in a genetic network. Proc. Natl. Acad. Sci. U. S. A *103*, 12718-12723.

Cao,Y., Gillespie,D.T., and Petzold,L.R. (2007). Adaptive explicit-implicit tau-leaping method with automatic tau selection. J. Chem. Phys. *126*, 224101.

Cox,R.S.I., Surette,M.G., and Elowitz,M.B. (2007). Programming gene expression with combinatorial promoters. Mol. Syst. Biol. *3*, 145.

de Jong,H. (2002). Modeling and simulation of genetic regulatory systems: a literature review. J. Comput. Biol. *9*, 67-103.

de Jong,H., Ranquet,C., Ropers,D., Pinel,C., and Geiselmann,J. (2010). Experimental and computational validation of models of fluorescent and luminescent reporter genes in bacteria. BMC. Syst. Biol. *4*, 55.

Drubin,D.A., Way,J.C., and Silver,P.A. (2007). Designing biological systems. Genes Dev. *21*, 242-254.

Dueber,J.E., Yeh,B.J., Chak,K., and Lim,W.A. (2003). Reprogramming control of an allosteric signaling switch through modular recombination. Science *301*, 1904-1908.

Elowitz,M.B. and Leibler,S. (2000). A synthetic oscillatory network of transcriptional regulators. Nature *403*, 335-338.

Friedland,A.E., Lu,T.K., Wang,X., Shi,D., Church,G., and Collins,J.J. (2009). Synthetic gene networks that count. Science *324*, 1199-1202.

Fuqua,C., Parsek,M.R., and Greenberg,E.P. (2001). Regulation of gene expression by cell-to-cell communication: acyl-homoserine lactone quorum sensing. Annu. Rev. Genet. *35*, 439-468.

Gardner,T.S., Cantor,C.R., and Collins,J.J. (2000). Construction of a genetic toggle switch in Escherichia coli. Nature *403*, 339-342.

Gillespie,D.T. (1977). Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry *81*, 2340-2361.

Goryachev,A.B., Toh,D.J., and Lee,T. (2006). Systems analysis of a quorum sensing network: design constraints imposed by the functional requirements, network topology and kinetic constants. Biosystems *83*, 178-187.

Ham,T.S., Lee,S.K., Keasling,J.D., and Arkin,A.P. (2006). A tightly regulated inducible expression system utilizing the fim inversion recombination switch. Biotechnol. Bioeng. *94*, 1-4.

Ham,T.S., Lee,S.K., Keasling,J.D., and Arkin,A.P. (2008). Design and construction of a double inversion recombination switch for heritable sequential genetic memory. PLoS. One. *3*, e2815.

Hartwell,L., Hood,L., Goldberg,M., Reynolds,A., and Silver,L. (2010). Genetics: From Genes to Genomes. McGraw-Hill Higher Education).

Haynes,K.A. and Silver,P.A. (2009). Eukaryotic systems broaden the scope of synthetic biology. J. Cell Biol. *187*, 589-596.

Head,C.G., Tardy,A., and Kenney,L.J. (1998). Relative binding affinities of OmpR and OmpR-phosphate at the ompF and ompC regulatory sites. J. Mol. Biol. *281*, 857-870.

Hindmarsh,A.C., Brown,P.N., Grant,K.E., Lee,S.L., Serban,R., Shumaker,D.E., and Woodward,C.S. (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans. Math. Softw *31*, 363-396.

Hong,M., Fitzgerald,M.X., Harper,S., Luo,C., Speicher,D.W., and Marmorstein,R. (2008). Structural basis for dimerization in DNA recognition by Gal4. Structure. *16*, 1019-1026.

Isaacs,F.J., Dwyer,D.J., Ding,C., Pervouchine,D.D., Cantor,C.R., and Collins,J.J. (2004). Engineered riboregulators enable post-transcriptional control of gene expression. Nat. Biotechnol. *22*, 841-847.

Janert,P.K. (2009). Gnuplot in Action. Manning Publications).

Kaps,P. and Rentrop,P. (1979). Generalized Runge-Kutta methods of order four with stepsize control for stiff ordinary differential equations. Numerische Mathematik *33*, 55-68.

Knight, T., Endy, D., and Voight, C. The BioBricks Foundation. 2004.

Ref Type: Online Source

Knight, T. F., Jr. and Sussman, G. J. Cellular gate technology. UMC98: First International Conference On Unconventional Models Of Computation. [1], 257-272. 1-1-1998.

Ref Type: Conference Proceeding

Kobayashi,H., Kaern,M., Araki,M., Chung,K., Gardner,T.S., Cantor,C.R., and Collins,J.J. (2004). Programmable cells: interfacing natural and engineered gene networks. Proc. Natl. Acad. Sci. U. S. A *101*, 8414-8419.

Kramer,B.P., Fischer,C., and Fussenegger,M. (2004). BioLogic gates enable logical transcription control in mammalian cells. Biotechnol. Bioeng. *87*, 478-484.

Kuhner,F., Costa,L.T., Bisch,P.M., Thalhammer,S., Heckl,W.M., and Gaub,H.E. (2004). LexA-DNA bond strength by single molecule force spectroscopy. Biophys. J. *87*, 2683-2690.

Levskaya,A., Chevalier,A.A., Tabor,J.J., Simpson,Z.B., Lavery,L.A., Levy,M., Davidson,E.A., Scouras,A., Ellington,A.D., Marcotte,E.M., and Voigt,C.A. (2005). Synthetic biology: engineering Escherichia coli to see light. Nature *438*, 441-442.

Lou,C., Liu,X., Ni,M., Huang,Y., Huang,Q., Huang,L., Jiang,L., Lu,D., Wang,M., Liu,C., Chen,D., Chen,C., Chen,X., Yang,L., Ma,H., Chen,J., and Ouyang,Q. (2010). Synthesizing a novel genetic sequential logic circuit: a push-on push-off switch. Mol. Syst. Biol. *6*, 350.

Medina,G., Juarez,K., Valderrama,B., and Soberon-Chavez,G. (2003). Mechanism of Pseudomonas aeruginosa RhlR transcriptional regulation of the rhlAB promoter. J. Bacteriol. *185*, 5976-5983.

Mullock,B.M. and Luzio,J.P. (2005). Theory of Organelle Biogenesis: AHistorical Perspective. Springer US), pp. 1-18.

Olivier,B.G., Rohwer,J.M., and Hofmeyr,J.H. (2005). Modelling cellular systems with PySCeS. Bioinformatics *21*, 560-561.

Orth,J.D., Thiele,I., and Palsson,B.O. (2010). What is flux balance analysis? Nat. Biotechnol. *28*, 245-248.

Reinitz,J. and Vaisnys,J.R. (1990). Theoretical and experimental analysis of the phage lambda genetic switch implies missing levels of co-operativity. J. Theor. Biol. *145*, 295-318.

Shih,M.C. and Gussin,G.N. (1984). Kinetic analysis of mutations affecting the cII activation site at the PRE promoter of bacteriophage lambda. Proc. Natl. Acad. Sci. U. S. A *81*, 6432-6436.

Shrout,J.D. and Parsek,M.R. (2006). Quorum Sensing: Coordinating Group Behavior Through Intercellular Signals. In Molecular Paradigms of Infectious Disease, C.A.Nickerson and M.J.Schurr, eds. Springer US), pp. 404-437.

Steindler,L. and Venturi,V. (2007). Detection of quorum-sensing N-acyl homoserine lactone signal molecules by bacterial biosensors. FEMS Microbiol. Lett. *266*, 1-9.

Stekel,D.J. and Jenkins,D.J. (2008). Strong negative self regulation of prokaryotic transcription factors increases the intrinsic noise of protein expression. BMC. Syst. Biol. *2*, 6.

Stricker,J., Cookson,S., Bennett,M.R., Mather,W.H., Tsimring,L.S., and Hasty,J. (2008). A fast, robust and tunable synthetic gene oscillator. Nature *456*, 516-519.

Sundararaj,S., Guo,A., Habibi-Nazhad,B., Rouani,M., Stothard,P., Ellison,M., and Wishart,D.S. (2004). The CyberCell Database (CCDB): a comprehensive, self-updating, relational database to coordinate and facilitate in silico modeling of Escherichia coli. Nucleic Acids Res. *32*, D293-D295.

Swinburne,I.A., Miguez,D.G., Landgraf,D., and Silver,P.A. (2008). Intron length increases oscillatory periods of gene expression in animal cells. Genes Dev. *22*, 2342-2346.

Tecon,R., Wells,M., and van der Meer,J.R. (2006). A new green fluorescent protein-based bacterial biosensor for analysing phenanthrene fluxes. Environ. Microbiol. *8*, 697-708.

Tigges,M., Marquez-Lago,T.T., Stelling,J., and Fussenegger,M. (2009). A tunable synthetic mammalian oscillator. Nature *457*, 309-312.

Vohradsky,J. (2001). Neural model of the genetic network. J. Biol. Chem. *276*, 36168-36173.

Wang,Y.M., Tegenfeldt,J.O., Reisner,W., Riehn,R., Guan,X.J., Guo,L., Golding,I., Cox,E.C., Sturm,J., and Austin,R.H. (2005). Single-molecule studies of repressor-DNA interactions show long-range interactions. Proc. Natl. Acad. Sci. U. S. A *102*, 9796-9801.

Weber,W., Stelling,J., Rimann,M., Keller,B., Daoud-El,B.M., Weber,C.C., Aubel,D., and Fussenegger,M. (2007). A synthetic time-delay circuit in mammalian cells and mice. Proc. Natl. Acad. Sci. U. S. A *104*, 2643-2648.

Weiss, R., Homsy, G., and Nagpal, R. Programming biological cells. Eighth International Conference on Architectural Support for Programming Languages and Operating Systems. Wild and Crazy Ideas Session [8]. 1998.

Ref Type: Conference Proceeding

Wolfram,S. (2002). A New Kind Of Science.

Zhu,Y., Qin,L., Yoshida,T., and Inouye,M. (2000). Phosphatase activity of histidine kinase EnvZ without kinase catalytic domain. Proc. Natl. Acad. Sci. U. S. A *97*, 7808-7813.

Appendix

Deterministic Run:

MyDet.cpp

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <stdexcept>
#include <sstream>
#include <string>
#include <cmath>
#include <ctime>
#include <cstdlib>

using namespace std;

#define  dim  25

#define  abt  1 /*sampling rate*/

// number of iterations
//#define  N  39600          // 11 hours (make arbitrarily high)
//#define  N  108000    // whatever, just testing
//#define  N  72000          // whatever, just testing
//#define  N  20000          // whatever, just testing
//#define  N  28000          // whatever, just testing
//#define  N  52000          // whatever, just testing
#define  N  10000       // whatever, just testing
//#define  N  7500           // whatever, just testing

#define  MAX_DATE_LEN  12

/*T=N*tau, where T is the real time. This is independent of abt.*/


      /***********************************************************/
      /********************** PARAMETERS ************************/
      /***********************************************************/


      // as per K_LacI = 10nM
      double K_taR12                            = 80; //*
      double K_cI                               = 8;
      double K_cII                              = 50;
      double K_ompR                             = 151;
      double K_ompRP                            = 6;
      double K_Gal4                             = 24;
      double K_TetR                             = 0.6;
      double K_LexA                             = 20; //*
//    double K_LacI                             = 10;
      double K_LacI                             = 14;
      double K_AHL_LEFT                   = 20; //*
      double K_AHL_RIGHT                        = 20; //*

//    // as per K_LacI = 1.7uM
//    double K_taR12                            = 1.7; //*
//    double K_cI                                 = 17;
//    double K_cII                                = 17;
//    double K_ompR                               = 1.7; //*
//    double K_ompRP                              = 1.7; //*
//    double K_Gal4                               = 1.7; //*
//    double K_TetR                               = 17;
//    double K_LacI                               = 1.7;

//    // as per K_LacI = 700 Molecules/cell
//    double K_taR12                              = 700; //*
//    double K_cI                                 = 7000;
//    double K_cII                                = 7000;
//    double K_ompR                               = 700; //*
//    double K_ompRP                              = 700; //*
//    double K_Gal4                               = 700; //*
//    double K_TetR                               = 7000;
//    double K_LacI                               = 700;

//    // as per K_LacI = 15 Molecules/cell
//    double K_taR12                              = 15; //*
//    double K_cI                                 = 150;
//    double K_cII                                = 150;
//    double K_ompR                               = 15; //*
//    double K_ompRP                              = 15; //*
//    double K_Gal4                               = 15; //*
//    double K_TetR                               = 150;
//    double K_LacI                               = 15;

      double n_taR12                              = 2;
      double n_cI                                 = 2;
      double n_cII                                = 2;
      double n_ompR                               = 2;
      double n_ompRP                              = 2;
      double n_Gal4                               = 2;
      double n_TetR                               = 3;
      double n_LexA                               = 2;
      double n_LacI                               = 2;
      double n_AHL_LEFT                     = 2;
      double n_AHL_RIGHT                          = 2;

//    double d_taR12                              = 0.006; //*
//    double d_mRNA                               = 0.006;
//    double d_cI                                 = 0.002888;
//    double d_cII                                = 0.002888; //*
//    double d_ompR                               = 0.002888; //*
//    double d_ompRP                              = 0.002888; //*
//    double d_Gal4                               = 0.002888; //*
//    double d_TetR                               = 0.002888;
//    double d_LexA                               = 0.002888; //*
//    double d_LacI                               = 0.002888;
//    double d_GFP                                = 0.002888; //*

      double d_taR12                              = 0.0026; //*
      double d_mRNA                               = 0.0026;
//    double d_mRNA                               = 0.006;
      double d_cI                                 = 0.0007*10;
      double d_cII                                = 0.0069;
      double d_ompR                              = 0.00132;
      double d_ompRP                             = 0.00132;
      double d_Gal4                              = 0.002888; //*
      double d_TetR                              = 0.00231*2; //*
      double d_LexA                              = 0.00231; //*
      double d_LacI                              = 0.00231;
      double d_GFP                               = 0.0002*10;
      double d_AHL_LEFT                    = 0.001; //*
      double d_AHL_RIGHT                         = 0.001; //*
      double d_AHL_CENTER                        = 0.001; //*

      double gp                                  = 0.1;
      double a                                   = 0.00;
      double T                                   = 0;
      double L                                   = 0;
      double AHL_LEFT                            = 0;
      double AHL_RIGHT                     = 0;

//    double V_phos                              = 20.0;     // Rate of
OmpR phosphorylation
//    double K_phos                              = 5.0;      // Kinetic
constant
//    double V_dephos                            = 0.01;     // Rate of
OmpRP dephosphorylation
//    double V_phos                              = 0.75;     // Rate of
OmpR phosphorylation
//    double K_phos                              = 0.25;     // Kinetic
constant
//    double V_dephos                            = 0.001;    // Rate of
OmpRP dephosphorylation
      double V_phos                              = 20.0;     // Rate of
OmpR phosphorylation
      double K_phos                              = 1.0;      // Kinetic
constant
      double V_dephos                            = 0.01;     // Rate of
OmpRP dephosphorylation

      double     K_y[dim];
      double     n_y[dim];
      double     d_y[dim];
      double     cmax[14];


      const string path = "T:/workspace/C++/MyDet/";
      const string path2 = "C:/Documents and Settings/Administrator/"
                                "Desktop/BioSym/May
2nd/Paper1/Results/";
//    const string path = "D:/Imad/workspace/C++/MyDet/";
//    const string path2 = "D:/Imad/workspace/C++/ResultsDet/";

      const string gPlot = "gnuplot.exe " + path + "test.gp";
//    const string gPlot = "gnuplot.exe " + path + "test2.gp -persist";
//    const string gPlot = "gnuplot.exe " + path + "test3.gp -persist";

      const char* gnuPlot = gPlot.c_str();
```

```
/************************************************************/
/*********************** FUNCTIONS *************************/
/************************************************************/


void RungeKutta(double y[],double h,double dy[]);
double *diff_eq(double y[], double dy[]);

bool fexists(const char *filename);
string getDate();
string nextFileName();
string itime(const double diff);


/************************************************************/
/********************* PROGRAM START ***********************/
/************************************************************/


int main()
{
      time_t start, end;
      double diff;

      time(&start);

      cout << "Starting Deterministic...\n" << endl;

      double tau = 10; //20; //5; // step used
      double t = 0;
      double y[dim], dy[dim];




      // Initial Conditions
      // Initializing ODEs
      for(int i=0; i<dim; ++i)
      {
            y[i]  = 0.0;
            dy[i] = 0.0;
            K_y[i]      = 0.0;
            n_y[i]      = 0.0;
            d_y[i]      = 0.0;
      }

//          y[0]  = 10;
//          y[1]  = 10;
//          y[2]  = 10;
//          y[3]  = 50000;
//          y[4]  = 50000;
//          y[5]  = 10;
//          y[6]  = 50000;
//          y[7]  = 50000;
//          y[8]  = 10;
//          y[9]  = 10;
//          y[10] = 10;
//          y[11] = 50000;
//          y[12] = 10;
//          y[13] = 50000;
//          y[14] = 10;
//          y[15] = 50000;
//          y[16] = 50000;
//          y[17] = 10;
//          y[18] = 10;
//          y[19] = 10;
//          y[20] = 10;
//          y[21] = 10;
//          y[22] = 10;

            // start with ON state (BioD)
            y[0]  = 0;         // 04
            y[1]  = 325;       // 06
            y[2]  = 0;         // 06
            y[3]  = 325;       // 06
            y[4]  = 33;        // 06
            y[5]  = 0.01;      // 06
            y[6]  = 0.01;      // 06
            y[7]  = 0;         // 06
            y[8]  = 326;       // 06
            y[9]  = 326;       // 06
            y[10] = 0;         // 04
            y[11] = 4750;      // 06
            y[12] = 2105;      // 06
            y[13] = 322;       // 06
            y[14] = 0.4;       // 06
            y[15] = 0.05;      // 06
            y[16] = 0.007;     // 06
            y[17] = 14150;     // 06
            y[18] = 16350;     // 06
            y[19] = 0;         // 06
            y[20] = 0.01;      // 06
            y[21] = 0.01;      // 06
            y[22] = 0;         // 06
            y[23] = 0;         // 06
            y[24] = 0;         // 06

            // start with ON state (BioFSM)
//          y[0]  = 0;         // 04
//          y[1]  = 262;       // 06
//          y[2]  = 0;         // 06
//          y[3]  = 229;       // 06
//          y[4]  = 33;        // 06
//          y[5]  = 0.001;     // 06
//          y[6]  = 0.001;     // 06
//          y[7]  = 0;         // 06
//          y[8]  = 327;       // 06
//          y[9]  = 327;       // 06
//          y[10] = 0;         // 04
//          y[11] = 3317;      // 06
//          y[12] = 712;       // 06
//          y[13] = 1764;      // 06
//          y[14] = 0.03;      // 06
//          y[15] = 0.03;      // 06
//          y[16] = 0.006;     // 06
//          y[17] = 14151;     // 06
//          y[18] = 16345;     // 06
//          y[19] = 0;         // 06
//          y[20] = 0;         // 06
//          y[21] = 327;       // 06
//          y[22] = 0;         // 06
//          y[23] = 0;         // 06
//          y[24] = 32663;     // 06

            // start with OFF state (BioD)
            y[0]  = 0;         // 04
            y[1]  = 260;       // 06
            y[2]  = 0;         // 06
            y[3]  = 230;       // 06
            y[4]  = 33;        // 06
            y[5]  = 0.001;     // 06
            y[6]  = 295;       // 06
            y[7]  = 295;       // 06
            y[8]  = 0;         // 06
            y[9]  = 0;         // 06
            y[10] = 0;         // 04
            y[11] = 3320;      // 06
            y[12] = 715;       // 06
            y[13] = 1750;      // 06
            y[14] = 0.04;      // 06
            y[15] = 6370;      // 06
            y[16] = 12750;     // 06
            y[17] = 0;         // 06
            y[18] = 0;         // 06
            y[19] = 0;         // 06
            y[20] = 0;         // 06
            y[21] = 0;         // 06
            y[22] = 0;         // 06
            y[23] = 0;         // 06
            y[24] = 0;         // 06

            // start with OFF state (BioFSM)
//          y[0]  = 0;         // 04
//          y[1]  = 262;       // 06
//          y[2]  = 0;         // 06
//          y[3]  = 229;       // 06
//          y[4]  = 33;        // 06
//          y[5]  = 0.001;     // 06
//          y[6]  = 294;       // 06
//          y[7]  = 294;       // 06
//          y[8]  = 0;         // 06
//          y[9]  = 0;         // 06
//          y[10] = 0;         // 04
//          y[11] = 3316;      // 06
//          y[12] = 712;       // 06
//          y[13] = 1764;      // 06
//          y[14] = 0.04;      // 06
//          y[15] = 6369;      // 06
//          y[16] = 12737;     // 06
//          y[17] = 0;         // 06
//          y[18] = 0;        // 06
//          y[19] = 0;        // 06
//          y[20] = 0;        // 06
//          y[21] = 0;        // 06
//          y[22] = 0;        // 06
//          y[23] = 0;        // 06
//          y[24] = 0.08;     // 06

            K_y[0]  = K_taR12;
            K_y[10] = K_cI;
            K_y[11] = K_cII;
            K_y[12] = K_ompR;
            K_y[13] = K_ompRP;
            K_y[14] = K_Gal4;
            K_y[15] = K_TetR;
            K_y[16] = K_LexA;
            K_y[17] = K_LacI;
            K_y[22] = K_AHL_LEFT;
            K_y[23] = K_AHL_RIGHT;

            n_y[0]  = n_taR12;
            n_y[10] = n_cI;
            n_y[11] = n_cII;
            n_y[12] = n_ompR;
            n_y[13] = n_ompRP;
            n_y[14] = n_Gal4;
            n_y[15] = n_TetR;
            n_y[16] = n_LexA;
            n_y[17] = n_LacI;
            n_y[22] = n_AHL_LEFT;
            n_y[23] = n_AHL_RIGHT;

            d_y[0]  = d_taR12;
            d_y[1]  = d_mRNA;
            d_y[2]  = d_mRNA;
            d_y[3]  = d_mRNA;
            d_y[4]  = d_mRNA;
            d_y[5]  = d_mRNA;
            d_y[6]  = d_mRNA;
            d_y[7]  = d_mRNA;
            d_y[8]  = d_mRNA;
            d_y[9]  = d_mRNA;
            d_y[10] = d_cI;
            d_y[11] = d_cII;
            d_y[12] = d_ompR;
            d_y[13] = d_ompRP;
            d_y[14] = d_Gal4;
            d_y[15] = d_TetR;
            d_y[16] = d_LexA;
            d_y[17] = d_LacI;
            d_y[18] = d_GFP;
            d_y[19] = d_mRNA;
            d_y[20] = d_mRNA;
            d_y[21] = d_mRNA;
            d_y[22] = d_AHL_LEFT;
            d_y[23] = d_AHL_RIGHT;
            d_y[24] = d_AHL_CENTER;
```

```
//          // nM/min
//          cmax[0] = 1.67e+1;
//          cmax[1] = 1.67e+2;
//          cmax[2] = 1.67e+2;
//          cmax[3] = 0.25e-0;
//          cmax[4] = 1.67e+2;
//          cmax[5] = 1.67e+2;
//          cmax[6] = 1.67e+2;
//          cmax[7] = 1.67e+2;

//          // uM/min
//          cmax[0] = 1.67e-2;
//          cmax[1] = 1.67e-1;
//          cmax[2] = 1.67e-1;
//          cmax[3] = 0.25e-3;
//          cmax[4] = 1.67e-1;
//          cmax[5] = 1.67e-1;
//          cmax[6] = 1.67e-1;
//          cmax[7] = 1.67e-1;

            // nM/s
            // average is 3.06 uM/h
//          cmax[0]  = 0.425e-1;
//          cmax[3]  = 0.125e-2;
            cmax[0]  = 0.1 * 0.85e-0;
            cmax[1]  = 0.8 * 0.85e-0;
            cmax[2]  = 0.7 * 0.85e-0;
            cmax[3]  = 0.1 * 0.85e-0;
            cmax[4]  = 0.3 * 0.85e-0;
            cmax[5]  = 0.3 * 0.85e-0;
            cmax[6]  = 0.9 * 0.85e-0;
            cmax[7]  = 1.0 * 0.85e-0;
            cmax[8]  = 1.0 * 0.85e-0;
            cmax[9]  = 1.0 * 0.85e-0;
            cmax[10] = 1.0 * 0.85e-0;
            cmax[11] = 1.0 * 0.85e-0;
            cmax[12] = 1.0 * 0.85e-0;
            cmax[13] = 1.0 * 0.85e-0;

            // Set up file

            remove((path + "deterministic.dat").c_str());
//          remove("T:/workspace/C++/MyDet/deterministic2.dat");
//          remove("T:/workspace/C++/MyDet/deterministic3.dat");

            ofstream outF1((path + "deterministic.dat").c_str());
//          ofstream
outF2("T:/workspace/C++/MyDet/deterministic2.dat");
//          ofstream
outF3("T:/workspace/C++/MyDet/deterministic3.dat");

            outF1 << "t\ttaR12\tmCIcr\tmCI\tmCII\tmOmpR\tmGal4\t" <<
                         "mTetR\tmLexA\tmLacI\tmGFP\tCI\tCII\tOmpR\t"
<<
                         "OmpRP\tGal4\tTetR\tLexA\tLacI\tGFP\t" <<
                         "mAHL_LEFT\tmAHL_RIGHT\tmAHL_CENTER\t" <<
```

```
                                "AHL_LEFT\tAHL_RIGHT\tAHL_CENTER\n";
//            outF1 <<
"t\ttaR12\tmCIcr\tmCI\tmCII\tmOmpR\tmGal4\tmTetR\t" <<
//
"mLexA\tmLacI\tmGFP\tCI\tCII\tOmpR\tOmpRP\tGal4\tTetR\t" <<
//
"LexA\tLacI\tGFP\tmCI_4\tmCII_5\tmTetR_5\tmLacI_4\n";
//            outF2 << "mCIcr\tmCI\tmCII\tCI\tCII\n";
//            outF3 << "mOmpR\tOmpR\tOmpRP\n";

              string filename = nextFileName();

              ofstream outF(filename.c_str());

              outF << "t\ttaR12\tmCIcr\tmCI\tmCII\tmOmpR\tmGal4\t" <<
                          "mTetR\tmLexA\tmLacI\tmGFP\tCI\tCII\tOmpR\t" <<
                          "OmpRP\tGal4\tTetR\tLexA\tLacI\tGFP\t" <<
                          "mAHL_LEFT\tmAHL_RIGHT\tmAHL_CENTER\t" <<
                          "AHL_LEFT\tAHL_RIGHT\tAHL_CENTER\n";
//            outF << "t\ttaR12\tmCIcr\tmCI\tmCII\tmOmpR\tmGal4\tmTetR\t"
<<
//
      "mLexA\tmLacI\tmGFP\tCI\tCII\tOmpR\tOmpRP\tGal4\tTetR\t" <<
//
      "LexA\tLacI\tGFP\tmCI_4\tmCII_5\tmTetR_5\tmLacI_4\n";


              //Define Input

              while (t < N)
              {
                    t += tau;

                    //INPUT SIGNALS

                    // taRNA input (taR12)
//                  T = ((t<500) || ((t>10000)&&(t<25000)) || (t>35000))
? 0 : 1;
//                  T = ((t<10000) || (t>15000)) ? 0 : 1;
//                  T = (t<10000) ? 0 : 1;  // 01 N=39600
//                  T = (t<2500) ? 0 : 1;   // 02 N=10000
//                  T = ((t<9000) || ((t>36000)&&(t<72000)) ||
//                         (t>99000)) ? 0 : 1;     // 03 N=108000
//                  T = ((t<36000) || (t>63000)) ? 0 : 1;     // 04
N=72000
//                  T = ((t<13000) || (t>17000)) ? 0 : 1;     // 05
N=20000
//                  T = ((t<2500) ||
//                         ((t>7000)&&(t<16500)) ||
//                         (t>23000)) ? 0 : 1;     // 06&07 N=28000
//                  T = ((t<2500) || (t>7000)) ? 0 : 1; // 06&07 N=28000
//                  T = ((t<2500) ||
//                         ((t>10000)&&(t<31000)) ||
//                         (t>42000)) ? 0 : 1;     // 08 N=52000
                    T = ((t<1500) || (t>6500)) ? 0 : 1; // 09 N=10000
      LAST ONE!
//                  T = ((t<10000) || ((t>15000)&&(t<22000))) ? 0 : 1;
//                T = (t<2000) ? 0 : 1;
//                T = 0;

                  // Red light
//                L = ((t<4000) || ((t>7000)&&(t<13000)) ||
//                      ((t>18000)&&(t<22000)) ||
//                      ((t>28000)&&(t<32000))) ? 0 : 1;
//                L = ((t<10000) || (t>20000)) ? 0 : 1;
//                L = ((t<5000) || ((t>15000)&&(t<25000)) ||
//                      (t>35000)) ? 0 : 1;      // 01 N=39600
//                L = ((t<1750) || (t>9000)) ? 0 : 1;  // 02 N=10000
//                L = ((t<18000) ||
//                      ((t>27000)&&(t<45000)) ||
//                      ((t>54000)&&(t<63000)) ||
//                      ((t>81000)&&(t<90000)) ||
//                      (t>104000)) ? 0 : 1;     // 03 N=108000
//                L = ((t<9000) || ((t>18000)&&(t<27000)) ||
//                      ((t>45000)&&(t<54000)) ||
//                      (t>67000)) ? 0 : 1;      // 04 N=72000
//                L = ((t<4000) || ((t>10000)&&(t<13100)) ||
//                      (t>17100)) ? 0 : 1;      // 05 N=20000
//                L = ((t<4000) ||
//                      ((t>5500)&&(t<10000)) ||
//                      ((t>13000)&&(t<15000)) ||
//                      ((t>18000)&&(t<22000)) ||
//                      (t>24000)) ? 0 : 1;      // 06 N=28000
//                L = ((t<4000) || (t>10000)) ? 0 : 1;      // 06
N=28000
//                L = ((t<4000) ||
//                      ((t>5500)&&(t<10000)) ||
//                      ((t>11000)&&(t<15000)) ||
//                      ((t>18000)&&(t<22000)) ||
//                      (t>24000)) ? 0 : 1;      // 07 N=28000
//                L = ((t<4000) ||
//                      ((t>6500)&&(t<18000)) ||
//                      ((t>23000)&&(t<28000)) ||
//                      ((t>34000)&&(t<40000)) ||
//                      (t>44000)) ? 0 : 1;      // 08 N=52000
                  L = ((t<2500) || (t>5500)) ? 0 : 1; // 09 N=10000
//                L = ((t<2500) || (t>7000)) ? 0 : 1; // 09 N=15000
//                L = 0;

//                T=0;L=0;

                  // AHL_LEFT
//                AHL_LEFT =  ((t<2400) ||
//                                  ((t>9900)&&(t<30900)) ||
//                                  (t>41900)) ? 0 : 1;
                  AHL_LEFT =  ((t<2500) || (t>5000)) ? 0 : 1;
                  AHL_LEFT = 0;

                  // AHL_RIGHT
//                AHL_RIGHT = ((t<2400) ||
//                                  ((t>9900)&&(t<30900)) ||
//                                  (t>41900)) ? 0 : 1;
                  AHL_RIGHT = ((t<2500) || (t>5000)) ? 0 : 1;
//                AHL_RIGHT = 0;
```

```
                        // Print Output

                        outF << t     <<"\t"<<y[0] <<"\t"<<y[1] <<"\t"<<y[2]
<<"\t"<<
                                y[3] <<"\t"<<y[4] <<"\t"<<y[5]
<<"\t"<<y[6] <<"\t"<<
                                y[7] <<"\t"<<y[8] <<"\t"<<y[9]
<<"\t"<<y[10]<<"\t"<<

    y[11]<<"\t"<<y[12]<<"\t"<<y[13]<<"\t"<<y[14]<<"\t"<<

    y[15]<<"\t"<<y[16]<<"\t"<<y[17]<<"\t"<<y[18]<<"\t"<<

    y[19]<<"\t"<<y[20]<<"\t"<<y[21]<<"\t"<<y[22]<<"\t"<<//"\n";
                                y[23]<<"\t"<<y[24]<<"\n";

                        outF1 << t     <<"\t"<<y[0] <<"\t"<<y[1] <<"\t"<<y[2]
<<"\t"<<
                                y[3] <<"\t"<<y[4] <<"\t"<<y[5]
<<"\t"<<y[6] <<"\t"<<
                                y[7] <<"\t"<<y[8] <<"\t"<<y[9]
<<"\t"<<y[10]<<"\t"<<

y[11]<<"\t"<<y[12]<<"\t"<<y[13]<<"\t"<<y[14]<<"\t"<<

y[15]<<"\t"<<y[16]<<"\t"<<y[17]<<"\t"<<y[18]<<"\t"<<

y[19]<<"\t"<<y[20]<<"\t"<<y[21]<<"\t"<<y[22]<<"\t"<<//"\n";
                                y[23]<<"\t"<<y[24]<<"\n";
//                      outF2 << t     <<"\t"<<y[1] <<"\t"<<y[2] <<"\t"<<y[3]
<<"\t"<<
//                              y[10]<<"\t"<<y[11]<<"\n";
//                      outF3 << t     <<"\t"<<y[4]
<<"\t"<<y[12]<<"\t"<<y[13]<<"\n";


                        // Integrating

                        RungeKutta(y,tau,dy);

                }

                outF.close();

                outF1.close();
//              outF2.close();
//              outF3.close();

                cout << "Done!\n" << endl;

                time (&end);
                diff = difftime(end, start);

                cout << "Time Elapsed: " << itime(diff) << endl << endl;

                char ans;
```

```
            cout << "Plot Graph(s)? (Yes/No)" << endl;
            cin >> ans;

            if (ans!='y' && ans!='Y')
            {
                  cout << endl << "End!" << endl;
                  return 0;
            }

            cout << "Plotting...\n" << endl;

//          remove("test.gp");
//
//          ofstream plotFile("T:/workspace/C++/MyDet/test.gp");
//
//          plotFile    << "set terminal wxt 0" << endl
//                            << "load
'T:/workspace/C++/MyDet/deterministic.plt'" << endl
//                            << "print \"Done!\\n\\n\"" << endl
//                            << "print \"Plotting the second
plot...!\\n\\n\"" << endl
//                            << "set terminal wxt 1" << endl
//                            << "load
'T:/workspace/C++/MyDet/deterministic_.plt'" << endl
//                            << "print \"Done!\\n\\n\"" << endl
//                            << "print \"Press Enter To Terminate
Program...!\\n\"" << endl
//                            << "pause -1" << endl;


            try
            {
                  // gnuPlot is "gnuplot.exe test.gp"
                  system(gnuPlot);
            }
            catch (invalid_argument& e)
            {
                  cerr << "ERROR: " << e.what();
            }
            catch (...)
            {
                  cerr << "Something Happened..!" << endl;
            }

            cout << endl << "End!" << endl;

            return 0;
      }


      /************************************************************/
      /*********************** FUNCTIONS *************************/
      /************************************************************/


      void RungeKutta(double y[],double h,double dy[])
      {
            double k1[dim],k2[dim],k3[dim],k4[dim];
            double y1[dim],y2[dim],y3[dim];
            double *p_dy;

            p_dy=diff_eq(y,dy);
            for(int i=0; i<dim; ++i)
            {
                  k1[i]=*(p_dy+i)*h;
                  y1[i]=y[i]+0.5*k1[i];
            }

            p_dy=diff_eq(y1,dy);
            for(int i=0; i<dim; ++i)
            {
                  k2[i]=*(p_dy+i)*h;
                  y2[i]=y[i]+0.5*k2[i];
            }

            p_dy=diff_eq(y2,dy);
            for(int i=0; i<dim; ++i)
            {
                  k3[i]=*(p_dy+i)*h;
                  y3[i]=y[i]+k3[i];
            }

            p_dy=diff_eq(y3,dy);
            for(int i=0; i<dim; ++i)
            {
                  k4[i]=*(p_dy+i)*h;
            }

            for(int i=0; i<dim; ++i)
                  y[i]=y[i]+(k1[i]+2.*k2[i]+2.*k3[i]+k4[i])/6.;

      }


      double *diff_eq(double y[], double dy[])
      {


      /***********************************************************/
            /******************** MRNA EQUATIONS
***********************/

      /***********************************************************/

            // d[taR12]/dt
            dy[0] = cmax[0]*T - d_y[0]*y[0];
//          dy[0] = cmax[8]*(a+(1-a)*(pow((y[22]/K_y[22]),n_y[22]) /
//
      (1+pow((y[22]/K_y[22]),n_y[22])))) +
//                      cmax[9]*(a+(1-
a)*(1/(1+pow((y[15]/K_y[15]),n_y[15])))
//
      *(pow((y[23]/K_y[23]),n_y[23]) /
//
       (1+pow((y[23]/K_y[23]),n_y[23])))) -
//                          d_y[0]*y[0];
//           dy[0] = cmax[8]*(a+(1-
a)*(1/(1+pow((y[17]/K_y[17]),n_y[17])))
//
       *(1/(1+pow((y[23]/K_y[23]),n_y[23])))
//
       *(pow((y[22]/K_y[22]),n_y[22]) /
//
       (1+pow((y[22]/K_y[22]),n_y[22])))) +
//                          cmax[9]*(a+(1-
a)*(1/(1+pow((y[22]/K_y[22]),n_y[22])))
//
       *(pow((y[23]/K_y[23]),n_y[23]) /
//
       (1+pow((y[23]/K_y[23]),n_y[23])))) +
//                          cmax[13]*(a+(1-
a)*(1/(1+pow((y[15]/K_y[15]),n_y[15])))
//
*(1/(1+pow((y[22]/K_y[22]),n_y[22])))) -
//                          d_y[0]*y[0];

             // d[mCIcr]/dt
             dy[1] = cmax[1]*(a+(1-
a)*(1/(1+pow((y[14]/K_y[14]),n_y[14])))) -
                          d_y[1]*y[1];

             // d[mCI]/dt
             dy[2] = cmax[4]*(a+(1-
a)*(1/(1+pow((y[11]/K_y[11]),n_y[11])))

       *(1/(1+pow((y[13]/K_y[13]),n_y[13])))

       *(pow((y[12]/K_y[12]),n_y[12]) /

       (1+pow((y[12]/K_y[12]),n_y[12])))) -
                          d_y[2]*y[2];

             // d[mCII]/dt
             dy[3] = cmax[2]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))) +
                          cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))

       *(1/(1+pow((y[13]/K_y[13]),n_y[13])))

       *(pow((y[12]/K_y[12]),n_y[12]) /

       (1+pow((y[12]/K_y[12]),n_y[12])))) -
                          d_y[3]*y[3];

             // d[mOmpR]/dt
             dy[4] = cmax[3] - d_y[4]*y[4];

             // d[mGal4]/dt
```

```
            dy[5] = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))

      *(1/(1+pow((y[13]/K_y[13]),n_y[13])))

      *(pow((y[12]/K_y[12]),n_y[12]) /

      (1+pow((y[12]/K_y[12]),n_y[12])))) -
                        d_y[5]*y[5];

            // d[mTetR]/dt
            dy[6] = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))

      *(1/(1+pow((y[13]/K_y[13]),n_y[13])))

      *(pow((y[12]/K_y[12]),n_y[12]) /

      (1+pow((y[12]/K_y[12]),n_y[12])))) +
                        cmax[6]*(a+(1-
a)*(1/(1+pow((y[17]/K_y[17]),n_y[17])))) -
                        d_y[6]*y[6];

            // d[mLexA]/dt
            dy[7] = cmax[6]*(a+(1-
a)*(1/(1+pow((y[17]/K_y[17]),n_y[17])))) -
                        d_y[7]*y[7];

            // d[mLacI]/dt
            dy[8] = cmax[4]*(a+(1-
a)*(1/(1+pow((y[11]/K_y[11]),n_y[11])))

      *(1/(1+pow((y[13]/K_y[13]),n_y[13])))

      *(pow((y[12]/K_y[12]),n_y[12]) /

      (1+pow((y[12]/K_y[12]),n_y[12])))) +
                        cmax[7]*(a+(1-
a)*(1/(1+pow((y[15]/K_y[15]),n_y[15])))) -
                        d_y[8]*y[8];

            // d[mGFP]/dt
            dy[9] = cmax[7]*(a+(1-
a)*(1/(1+pow((y[15]/K_y[15]),n_y[15])))) -
                        d_y[9]*y[9];

            // d[mCI_4]/dt
//          dy[19] = cmax[4]*(a+(1-
a)*(1/(1+pow((y[11]/K_y[11]),n_y[11])))
//
*(1/(1+pow((y[13]/K_y[13]),n_y[13])))
//
*(pow((y[12]/K_y[12]),n_y[12]) /
//
(1+pow((y[12]/K_y[12]),n_y[12])))) -
//                      d_y[1]*y[19];
            // d[mAHL_LEFT]/dt
            dy[19] = cmax[11]*AHL_LEFT - d_y[19]*y[19];

            // d[mCII_5]/dt
//          dy[20] = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))
//
*(1/(1+pow((y[13]/K_y[13]),n_y[13])))
//
*(pow((y[12]/K_y[12]),n_y[12]) /
//
(1+pow((y[12]/K_y[12]),n_y[12])))) -
//                         d_y[2]*y[20];
            // d[mAHL_RIGHT]/dt
            dy[20] = cmax[12]*AHL_RIGHT - d_y[20]*y[20];

            // d[mTetR_5]/dt
//          dy[21] = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))
//
*(1/(1+pow((y[13]/K_y[13]),n_y[13])))
//
*(pow((y[12]/K_y[12]),n_y[12]) /
//
(1+pow((y[12]/K_y[12]),n_y[12])))) -
//                         d_y[6]*y[21];
            // d[mAHL_CENTER]/dt
            dy[21] = cmax[10]*(a+(1-
a)*(1/(1+pow((y[16]/K_y[16]),n_y[16])))) -
                        d_y[21]*y[21];

            // d[mLacI_4]/dt
//          dy[22] = cmax[4]*(a+(1-
a)*(1/(1+pow((y[11]/K_y[11]),n_y[11])))
//
*(1/(1+pow((y[13]/K_y[13]),n_y[13])))
//
*(pow((y[12]/K_y[12]),n_y[12]) /
//
(1+pow((y[12]/K_y[12]),n_y[12])))) -
//                         d_y[8]*y[22];


      /************************************************************/
            /****************** PROTEIN EQUATIONS
**********************/

      /************************************************************/

//          // d[CI]/dt
//          dy[10] = gp*y[1]*(a+(1-a)*(pow((y[0]/K_y[0]),n_y[0]) /
//
(1+pow((y[0]/K_y[0]),n_y[0])))) +
//                         gp*y[2] -
//                         d_y[10]*y[10];

            // d[CI]/dt
```

```
        dy[10] = gp*y[1]*(a+(1-a)*y[0]*0.04) +
                     gp*y[2] -
                     d_y[10]*y[10];

        // d[CII]/dt
        dy[11] = gp*y[3] - d_y[11]*y[11];

        // d[OmpR]/dt
        dy[12] = gp*y[4] + V_dephos * y[13] -
                     (V_phos*(1-L)*y[12])/(K_phos + y[12]) -
                     d_y[12]*y[12];

        // d[OmpRP]/dt
        dy[13] = (V_phos*(1-L)*y[12])/(K_phos + y[12]) -
                     V_dephos * y[13] -
                     d_y[13]*y[13];

        // d[Gal4]/dt
        dy[14] = gp*y[5] - d_y[14]*y[14];

        // d[TetR]/dt
        dy[15] = gp*y[6] - d_y[15]*y[15];

        // d[LexA]/dt
        dy[16] = gp*y[7] - d_y[16]*y[16];

        // d[LacI]/dt
        dy[17] = gp*y[8] - d_y[17]*y[17];

        // d[GFP]/dt
        dy[18] = gp*y[9] - d_y[18]*y[18];

        // d[AHL_LEFT]/dt
        dy[22] = gp*y[19] - d_y[22]*y[22];

        // d[AHL_RIGHT]/dt
        dy[23] = gp*y[20] - d_y[23]*y[23];

        // d[AHL_CENTER]/dt
        dy[24] = gp*y[21] - d_y[24]*y[24];


        return dy;
}

bool fexists(const char *filename)
{
  ifstream ifile(filename);
  return ifile;
}

string getDate()
{
   time_t now;
   char theDate[MAX_DATE_LEN];

   theDate[0] = '\0';
```

```
          now = time(0);

          if (now != -1)
          {
//              strftime(theDate, MAX_DATE_LEN,
//                           "%Y%h%d_%Hh%Mm%Ss", localtime(&now));
                strftime(theDate, MAX_DATE_LEN, "%Y%m%d",
localtime(&now));
          }

          return string(theDate);
      }

      string nextFileName()
      {
            int count=0;
            stringstream ss;
            string date = getDate();

            do
            {
                  count++;
                  ss.str("");

                  ss    << path2
                        << date
                        << "_DET_"
                        << setw(3) << setfill('0') << count
                        << ".xls";
            }
            while (fexists(ss.str().c_str()));

            return ss.str();
      }

      string itime(const double diff)
      {
            stringstream ss;

            int hrs = int (diff/3600);
            int mins = int ((diff - hrs*3600)/60);
            int secs = int ((diff - hrs*3600 - mins*60));

            ss    << setw(2) << setfill('0')
                  << hrs << ":"
                  << setw(2) << setfill('0')
                  << mins << ":"
                  << setw(2) << setfill('0')
                  << secs;

            return ss.str();
      }
```

GnuPlot File:

deterministc.plt

```
#!/gnuplot
#
#
#     G N U P L O T
#     Version 4.2 patchlevel 5
#     last modified Mar 2009
#     System: MS-Windows 32 bit
#
#     Copyright (C) 1986 - 1993, 1998, 2004, 2007 - 2009
#     Thomas Williams, Colin Kelley and many others
#
#     Type `help` to access the on-line reference manual.
#     The gnuplot FAQ is available from http://www.gnuplot.info/faq/
#
#     Send bug reports and suggestions to
<http://sourceforge.net/projects/gnuplot>
#
# set terminal windows color noenhanced
# set output
reset
GNUTERM = "win"

set xlabel "Time (hours)"
set ylabel "Protein Levels (uM)"

set lmargin at screen 0.055

set yrange [-2:18]

set ytics 4
set xtics 4

set object 1 rectangle from 4000/3600.0, graph 0 to 8500/3600.0, graph
1 fc lt 1 fs transparent solid 0.25 noborder
set object 2 rectangle from 18000/3600.0, graph 0 to 23000/3600.0,
graph 1 fc lt 1 fs transparent solid 0.25 noborder
set object 3 rectangle from 28000/3600.0, graph 0 to 34000/3600.0,
graph 1 fc lt 1 fs transparent solid 0.25 noborder
set object 4 rectangle from 40000/3600.0, graph 0 to 44000/3600.0,
graph 1 fc lt 1 fs transparent solid 0.25 noborder

set object 5 rectangle from 2500/3600.0, graph 0 to 10000/3600.0, graph
1 fc lt -1 fs transparent pattern 7 noborder
set object 6 rectangle from 31000/3600.0, graph 0 to 42000/3600.0,
graph 1 fc lt -1 fs transparent pattern 7 noborder

plot " deterministic.dat" using ($1/3600.0):($12/1000.0) t "CI" w l lc
rgb "#008000" lw 2, "deterministic.dat" u ($1/3600.0):($13/1000.0) t
"CII" w l lt 1 lw 2, " deterministic.dat" u ($1/3600.0):($14/1000.0) t
"OmpR" w l lc rgb "#008080" lw 2, " deterministic.dat" u
($1/3600.0):($15/1000.0) t "OmpRP" w l lc rgb "#FF8000" lw 2, "
deterministic.dat" using ($1/3600.0):($16/1000.0) t "Gal4" w l lc rgb
```

```
"#808000" lw 2, " deterministic.dat" using ($1/3600.0):($17/1000.0) t
"TetR" w l lt 4 lw 2.5, " deterministic.dat" using
($1/3600.0):($19/1000.0) t "LacI" w l lt -1 lw 2.5

#    EOF
```

Stochastic Run:

MySto.cpp

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <stdexcept>
#include <sstream>
#include <string>
#include <cmath>
#include <ctime>
#include <cstdlib>

using namespace std;

#define  dim  25

#define  abt  1 /*sampling rate*/

// number of iterations
//#define  N  39600          // 11 hours (make arbitrarily high)
//#define  N  108000    // whatever, just testing
//#define  N  72000          // whatever, just testing
//#define  N  20000          // whatever, just testing
//#define  N  28000          // whatever, just testing
//#define  N  52000          // whatever, just testing
//#define  N  10000          // whatever, just testing
#define  N  7500        // whatever, just testing

#define  MAX_DATE_LEN  12

/*T=N*tau, where T is the real time. This is independent of abt.*/


      /***********************************************************/
      /********************** PARAMETERS ************************/
      /***********************************************************/


      // as per K_LacI = 10nM
      double K_taR12                              = 80; //*
      double K_cI                                 = 8;
      double K_cII                                = 50;
      double K_ompR                               = 151;
      double K_ompRP                              = 6;
      double K_Gal4                               = 24;
      double K_TetR                               = 0.6;
      double K_LexA                               = 20; //*
      double K_LacI                               = 10;
      double K_AHL_LEFT                     = 20; //*
      double K_AHL_RIGHT                          = 20; //*

//    // as per K_LacI = 1.7uM
//    double K_taR12                              = 1.7; //*
//    double K_cI                                 = 17;
//    double K_cII                                 = 17;
//    double K_ompR                                = 1.7; //*
//    double K_ompRP                               = 1.7; //*
//    double K_Gal4                                = 1.7; //*
//    double K_TetR                                = 17;
//    double K_LacI                                = 1.7;

//    // as per K_LacI = 700 Molecules/cell
//    double K_taR12                               = 700; //*
//    double K_cI                                  = 7000;
//    double K_cII                                 = 7000;
//    double K_ompR                                = 700; //*
//    double K_ompRP                               = 700; //*
//    double K_Gal4                                = 700; //*
//    double K_TetR                                = 7000;
//    double K_LacI                                = 700;

//    // as per K_LacI = 15 Molecules/cell
//    double K_taR12                               = 15; //*
//    double K_cI                                  = 150;
//    double K_cII                                 = 150;
//    double K_ompR                                = 15; //*
//    double K_ompRP                               = 15; //*
//    double K_Gal4                                = 15; //*
//    double K_TetR                                = 150;
//    double K_LacI                                = 15;

      double n_taR12                               = 2;
      double n_cI                                  = 2;
      double n_cII                                 = 2;
      double n_ompR                                = 2;
      double n_ompRP                               = 2;
      double n_Gal4                                = 2;
      double n_TetR                                = 3;
      double n_LexA                                = 2;
      double n_LacI                                = 2;
      double n_AHL_LEFT                      = 2;
      double n_AHL_RIGHT                           = 2;

//    double d_taR12                               = 0.006; //*
//    double d_mRNA                                = 0.006;
//    double d_cI                                  = 0.002888;
//    double d_cII                                 = 0.002888; //*
//    double d_ompR                                = 0.002888; //*
//    double d_ompRP                               = 0.002888; //*
//    double d_Gal4                                = 0.002888; //*
//    double d_TetR                                = 0.002888;
//    double d_LexA                                = 0.002888; //*
//    double d_LacI                                = 0.002888;
//    double d_GFP                                 = 0.002888; //*

      double d_taR12                               = 0.0026; //*
      double d_mRNA                                = 0.0026;
//    double d_mRNA                                = 0.006;
      double d_cI                                  = 0.0007*10;
      double d_cII                                 = 0.0069;
      double d_ompR                                = 0.00132;
```

```
	double d_ompRP                                  = 0.00132;
	double d_Gal4                                   = 0.002888; //*
	double d_TetR                                   = 0.00231*2; //*
	double d_LexA                                   = 0.00231; //*
	double d_LacI                                   = 0.00231;
	double d_GFP                                    = 0.0002*10;
	double d_AHL_LEFT                         = 0.001; //*
	double d_AHL_RIGHT                              = 0.001; //*
	double d_AHL_CENTER                             = 0.001; //*

	double gp                                       = 0.1;
	double a                                        = 0.00;
	double T                                        = 0;
	double L                                        = 0;
	double AHL_LEFT                                 = 0;
	double AHL_RIGHT                          = 0;

//	double V_phos                                   = 20.0;      // Rate of
OmpR phosphorylation
//	double K_phos                                   = 5.0;       // Kinetic
constant
//	double V_dephos                                 = 0.01;      // Rate of
OmpRP dephosphorylation
//	double V_phos                                   = 0.75;      // Rate of
OmpR phosphorylation
//	double K_phos                                   = 0.25;      // Kinetic
constant
//	double V_dephos                                 = 0.001;     // Rate of
OmpRP dephosphorylation
	double V_phos                                   = 20.0;      // Rate of
OmpR phosphorylation
	double K_phos                                   = 1.0;       // Kinetic
constant
	double V_dephos                                 = 0.01;      // Rate of
OmpRP dephosphorylation

	double     K_y[dim];
	double     n_y[dim];
	double     d_y[dim];
	double     cmax[14];


	const string path = "T:/workspace/C++/MySto/";
	const string path2 = "C:/Documents and Settings/Administrator/"
	                               "Desktop/BioSym/May
2nd/Paper1/Results/";
//	const string path = "D:/Imad/workspace/C++/MySto/";
//	const string path2 = "D:/Imad/workspace/C++/ResultsSto/";

	const string gPlot = "gnuplot.exe " + path + "test.gp";
//	const string gPlot = "gnuplot.exe " + path + "test2.gp -persist";
//	const string gPlot = "gnuplot.exe " + path + "test3.gp -persist";

	const char* gnuPlot = gPlot.c_str();
```

```
/**********************************************************/
/*********************** FUNCTIONS ************************/
/**********************************************************/


const int PoissonRandomNumber(const double lambda);

bool fexists(const char *filename);
string getDate();
string nextFileName();
string itime(const double diff);


/**********************************************************/
/********************* PROGRAM START **********************/
/**********************************************************/


int main()
{
      time_t start, end;
      double diff;

      time(&start);

      cout << "Starting Stochastic...\n" << endl;

      srand( (unsigned int)time(NULL) );  //initialize random
generator

      double tau = 10; //20; //5; // step used
      double t = 0;
      double lambda;
      double y[dim],dy[dim];
      double d1,d2,d3,d4;

      remove((path + "stochastic.dat").c_str());

      ofstream outFS((path + "stochastic.dat").c_str());
      outFS << "t\ttaR12\tmCIcr\tmCI\tmCII\tmOmpR\tmGal4\t" <<
                  "mTetR\tmLexA\tmLacI\tmGFP\tCI\tCII\tOmpR\t"
<<
                  "OmpRP\tGal4\tTetR\tLexA\tLacI\tGFP\t" <<
                  "mAHL_LEFT\tmAHL_RIGHT\tmAHL_CENTER\t" <<
                  "AHL_LEFT\tAHL_RIGHT\tAHL_CENTER\n";
//          outFS <<
"t\ttaR12\tmCIcr\tmCI\tmCII\tmOmpR\tmGal4\tmTetR\t" <<
//
"mLexA\tmLacI\tmGFP\tCI\tCII\tOmpR\tOmpRP\tGal4\tTetR\t" <<
//
"LexA\tLacI\tGFP\tmCI_4\tmCII_5\tmTetR_5\tmLacI_4\n";


      string filename = nextFileName();
      ofstream outF(filename.c_str());

      outF << "t\ttaR12\tmCIcr\tmCI\tmCII\tmOmpR\tmGal4\t" <<
```

```
                            "mTetR\tmLexA\tmLacI\tmGFP\tCI\tCII\tOmpR\t" <<
                            "OmpRP\tGal4\tTetR\tLexA\tLacI\tGFP\t" <<
                            "mAHL_LEFT\tmAHL_RIGHT\tmAHL_CENTER\t" <<
                            "AHL_LEFT\tAHL_RIGHT\tAHL_CENTER\n";
//              outF << "t\ttaR12\tmCIcr\tmCI\tmCII\tmOmpR\tmGal4\tmTetR\t"
<<
//
        "mLexA\tmLacI\tmGFP\tCI\tCII\tOmpR\tOmpRP\tGal4\tTetR\t" <<
//
        "LexA\tLacI\tGFP\tmCI_4\tmCII_5\tmTetR_5\tmLacI_4\n";


                // Initial Conditions
                // Initializing ODEs
                for(int i=0; i<dim; ++i)
                {
                        y[i]  = 0.0;
                        dy[i] = 0.0;
                        K_y[i]      = 0.0;
                        n_y[i]      = 0.0;
                        d_y[i]      = 0.0;
                }

//              y[0]  = 10;
//              y[1]  = 10;
//              y[2]  = 10;
//              y[3]  = 50000;
//              y[4]  = 50000;
//              y[5]  = 10;
//              y[6]  = 50000;
//              y[7]  = 50000;
//              y[8]  = 10;
//              y[9]  = 10;
//              y[10] = 10;
//              y[11] = 50000;
//              y[12] = 10;
//              y[13] = 50000;
//              y[14] = 10;
//              y[15] = 50000;
//              y[16] = 50000;
//              y[17] = 10;
//              y[18] = 10;
//              y[19] = 10;
//              y[20] = 10;
//              y[21] = 10;
//              y[22] = 10;

                // start with ON state (BioD)
//              y[0]  = 0;        // 04
//              y[1]  = 325;      // 06
//              y[2]  = 0;        // 06
//              y[3]  = 325;      // 06
//              y[4]  = 33;       // 06
//              y[5]  = 0.01;     // 06
//              y[6]  = 0.01;     // 06
//              y[7]  = 0;        // 06
//              y[8]  = 326;      // 06
//          y[9]  = 326;      // 06
//          y[10] = 0;        // 04
//          y[11] = 4750;     // 06
//          y[12] = 2105;     // 06
//          y[13] = 322;      // 06
//          y[14] = 0.4;      // 06
//          y[15] = 0.05;     // 06
//          y[16] = 0.007;    // 06
//          y[17] = 14150;    // 06
//          y[18] = 16350;    // 06
//          y[19] = 0;        // 06
//          y[20] = 0.01;     // 06
//          y[21] = 0.01;     // 06
//          y[22] = 0;        // 06
//          y[23] = 0;        // 06
//          y[24] = 0;        // 06

            // start with ON state (BioFSM)
            y[0]  = 0;        // 04
            y[1]  = 262;      // 06
            y[2]  = 0;        // 06
            y[3]  = 229;      // 06
            y[4]  = 33;       // 06
            y[5]  = 0.001;    // 06
            y[6]  = 0.001;    // 06
            y[7]  = 0;        // 06
            y[8]  = 327;      // 06
            y[9]  = 327;      // 06
            y[10] = 0;        // 04
            y[11] = 3317;     // 06
            y[12] = 712;      // 06
            y[13] = 1764;     // 06
            y[14] = 0.03;     // 06
            y[15] = 0.03;     // 06
            y[16] = 0.006;    // 06
            y[17] = 14151;    // 06
            y[18] = 16345;    // 06
            y[19] = 0;        // 06
            y[20] = 0;        // 06
            y[21] = 327;      // 06
            y[22] = 0;        // 06
            y[23] = 0;        // 06
            y[24] = 32663;    // 06

            // start with OFF state (BioD)
//          y[0]  = 0;        // 04
//          y[1]  = 260;      // 06
//          y[2]  = 0;        // 06
//          y[3]  = 230;      // 06
//          y[4]  = 33;       // 06
//          y[5]  = 0.001;    // 06
//          y[6]  = 295;      // 06
//          y[7]  = 295;      // 06
//          y[8]  = 0;        // 06
//          y[9]  = 0;        // 06
//          y[10] = 0;        // 04
//          y[11] = 3320;     // 06
//          y[12] = 715;       // 06
//          y[13] = 1750;      // 06
//          y[14] = 0.04;      // 06
//          y[15] = 6370;      // 06
//          y[16] = 12750;     // 06
//          y[17] = 0;         // 06
//          y[18] = 0;         // 06
//          y[19] = 0;         // 06
//          y[20] = 0;         // 06
//          y[21] = 0;         // 06
//          y[22] = 0;         // 06
//          y[23] = 0;         // 06
//          y[24] = 0;         // 06

            // start with OFF state (BioFSM)
//          y[0]  = 0;         // 04
//          y[1]  = 262;       // 06
//          y[2]  = 0;         // 06
//          y[3]  = 229;       // 06
//          y[4]  = 33;        // 06
//          y[5]  = 0.001;     // 06
//          y[6]  = 294;       // 06
//          y[7]  = 294;       // 06
//          y[8]  = 0;         // 06
//          y[9]  = 0;         // 06
//          y[10] = 0;         // 04
//          y[11] = 3316;      // 06
//          y[12] = 712;       // 06
//          y[13] = 1764;      // 06
//          y[14] = 0.04;      // 06
//          y[15] = 6369;      // 06
//          y[16] = 12737;     // 06
//          y[17] = 0;         // 06
//          y[18] = 0;         // 06
//          y[19] = 0;         // 06
//          y[20] = 0;         // 06
//          y[21] = 0;         // 06
//          y[22] = 0;         // 06
//          y[23] = 0;         // 06
//          y[24] = 0.08;      // 06

            K_y[0]  = K_taR12;
            K_y[10] = K_cI;
            K_y[11] = K_cII;
            K_y[12] = K_ompR;
            K_y[13] = K_ompRP;
            K_y[14] = K_Gal4;
            K_y[15] = K_TetR;
            K_y[16] = K_LexA;
            K_y[17] = K_LacI;
            K_y[22] = K_AHL_LEFT;
            K_y[23] = K_AHL_RIGHT;

            n_y[0]  = n_taR12;
            n_y[10] = n_cI;
            n_y[11] = n_cII;
            n_y[12] = n_ompR;
            n_y[13] = n_ompRP;
            n_y[14] = n_Gal4;
            n_y[15] = n_TetR;
            n_y[16] = n_LexA;
            n_y[17] = n_LacI;
            n_y[22] = n_AHL_LEFT;
            n_y[23] = n_AHL_RIGHT;

            d_y[0]  = d_taR12;
            d_y[1]  = d_mRNA;
            d_y[2]  = d_mRNA;
            d_y[3]  = d_mRNA;
            d_y[4]  = d_mRNA;
            d_y[5]  = d_mRNA;
            d_y[6]  = d_mRNA;
            d_y[7]  = d_mRNA;
            d_y[8]  = d_mRNA;
            d_y[9]  = d_mRNA;
            d_y[10] = d_cI;
            d_y[11] = d_cII;
            d_y[12] = d_ompR;
            d_y[13] = d_ompRP;
            d_y[14] = d_Gal4;
            d_y[15] = d_TetR;
            d_y[16] = d_LexA;
            d_y[17] = d_LacI;
            d_y[18] = d_GFP;
            d_y[19] = d_mRNA;
            d_y[20] = d_mRNA;
            d_y[21] = d_mRNA;
            d_y[22] = d_AHL_LEFT;
            d_y[23] = d_AHL_RIGHT;
            d_y[24] = d_AHL_CENTER;

//          // nM/min
//          cmax[0] = 1.67e+1;
//          cmax[1] = 1.67e+2;
//          cmax[2] = 1.67e+2;
//          cmax[3] = 0.25e-0;
//          cmax[4] = 1.67e+2;
//          cmax[5] = 1.67e+2;
//          cmax[6] = 1.67e+2;
//          cmax[7] = 1.67e+2;

//          // uM/min
//          cmax[0] = 1.67e-2;
//          cmax[1] = 1.67e-1;
//          cmax[2] = 1.67e-1;
//          cmax[3] = 0.25e-3;
//          cmax[4] = 1.67e-1;
//          cmax[5] = 1.67e-1;
//          cmax[6] = 1.67e-1;
//          cmax[7] = 1.67e-1;

//          // nM/s
//          // average is 3.06 uM/h
////        cmax[0] = 0.425e-1;
```

```
//          cmax[0] = 0.85e-1;
//          cmax[1] = 0.85e-0;
//          cmax[2] = 0.85e-0;
////        cmax[3] = 0.125e-2;
//          cmax[3] = 0.85e-1;
//          cmax[4] = 0.85e-1;
//          cmax[5] = 0.85e-1;
//          cmax[6] = 0.85e-0;
//          cmax[7] = 0.85e-0;

            // nM/s
            // average is 3.06 uM/h
//          cmax[0]  = 0.425e-1;
//          cmax[3]  = 0.125e-2;
            cmax[0]  = 0.1 * 0.85e-0;
            cmax[1]  = 0.8 * 0.85e-0;
            cmax[2]  = 0.7 * 0.85e-0;
            cmax[3]  = 0.1 * 0.85e-0;
            cmax[4]  = 0.3 * 0.85e-0;
            cmax[5]  = 0.3 * 0.85e-0;
            cmax[6]  = 0.9 * 0.85e-0;
            cmax[7]  = 1.0 * 0.85e-0;
            cmax[8]  = 1.0 * 0.85e-0;
            cmax[9]  = 1.0 * 0.85e-0;
            cmax[10] = 1.0 * 0.85e-0;
            cmax[11] = 1.0 * 0.85e-0;
            cmax[12] = 1.0 * 0.85e-0;
            cmax[13] = 1.0 * 0.85e-0;

            while (t < N)
            {
                  t += tau;

                  //INPUT SIGNALS

                  // taRNA input (taR12)
//                T = ((t<500) || ((t>10000)&&(t<25000)) || (t>35000))
? 0 : 1;
//                T = ((t<10000) || (t>15000)) ? 0 : 1;
//                T = (t<10000) ? 0 : 1;  // 01 N=39600
//                T = (t<2500) ? 0 : 1;   // 02 N=10000
//                T = ((t<9000) || ((t>36000)&&(t<72000)) ||
//                      (t>99000)) ? 0 : 1;     // 03 N=108000
//                T = ((t<36000) || (t>63000)) ? 0 : 1;     // 04
N=72000
//                T = ((t<13000) || (t>17000)) ? 0 : 1;     // 05
N=20000
//                T = ((t<2500) ||
//                      ((t>7000)&&(t<16500)) ||
//                      (t>23000)) ? 0 : 1;     // 06&07 N=28000
//                T = ((t<2500) || (t>7000)) ? 0 : 1; // 06&07 N=28000
//                T = ((t<2500) ||
//                      ((t>10000)&&(t<31000)) ||
//                      (t>42000)) ? 0 : 1;     // 08 N=52000
//                T = ((t<1500) || (t>6500)) ? 0 : 1; // 09 N=12000
      LAST ONE!
//                T = ((t<10000) || ((t>15000)&&(t<22000))) ? 0 : 1;
//                  T = (t<2000) ? 0 : 1;
//                  T = 0;

                    // Red light
//                  L = ((t<4000) || ((t>7000)&&(t<13000)) ||
//                        ((t>18000)&&(t<22000)) ||
//                        ((t>28000)&&(t<32000))) ? 0 : 1;
//                  L = ((t<10000) || (t>20000)) ? 0 : 1;
//                  L = ((t<5000) || ((t>15000)&&(t<25000)) ||
//                        (t>35000)) ? 0 : 1;      // 01 N=39600
//                  L = ((t<1750) || (t>9000)) ? 0 : 1;  // 02 N=10000
//                  L = ((t<18000) ||
//                        ((t>27000)&&(t<45000)) ||
//                        ((t>54000)&&(t<63000)) ||
//                        ((t>81000)&&(t<90000)) ||
//                        (t>104000)) ? 0 : 1;     // 03 N=108000
//                  L = ((t<9000) || ((t>18000)&&(t<27000)) ||
//                        ((t>45000)&&(t<54000)) ||
//                        (t>67000)) ? 0 : 1;      // 04 N=72000
//                  L = ((t<4000) || ((t>10000)&&(t<13100)) ||
//                        (t>17100)) ? 0 : 1;      // 05 N=20000
//                  L = ((t<4000) ||
//                        ((t>5500)&&(t<10000)) ||
//                        ((t>13000)&&(t<15000)) ||
//                        ((t>18000)&&(t<22000)) ||
//                        (t>24000)) ? 0 : 1;      // 06 N=28000
//                  L = ((t<4000) || (t>10000)) ? 0 : 1;      // 06
N=28000
//                  L = ((t<4000) ||
//                        ((t>5500)&&(t<10000)) ||
//                        ((t>11000)&&(t<15000)) ||
//                        ((t>18000)&&(t<22000)) ||
//                        (t>24000)) ? 0 : 1;      // 07 N=28000
//                  L = ((t<4000) ||
//                        ((t>6500)&&(t<18000)) ||
//                        ((t>23000)&&(t<28000)) ||
//                        ((t>34000)&&(t<40000)) ||
//                        (t>44000)) ? 0 : 1;      // 08 N=52000
                    L = ((t<2500) || (t>7000)) ? 0 : 1; // 09 N=15000
                    L = 0;

//                  T=0;L=0;

                    // AHL_LEFT
//                  AHL_LEFT =  ((t<2400) ||
//                                    ((t>9900)&&(t<30900)) ||
//                                    (t>41900)) ? 0 : 1;
                    AHL_LEFT =  ((t<2500) || (t>5000)) ? 0 : 1;
                    AHL_LEFT = 0;

                    // AHL_RIGHT
//                  AHL_RIGHT = ((t<2400) ||
//                                    ((t>9900)&&(t<30900)) ||
//                                    (t>41900)) ? 0 : 1;
                    AHL_RIGHT = ((t<2500) || (t>5000)) ? 0 : 1;
                    AHL_RIGHT = 0;
                    // Print Output

                    outF << t    <<"\t"<<y[0] <<"\t"<<y[1] <<"\t"<<y[2]
<<"\t"<<
                                 y[3] <<"\t"<<y[4] <<"\t"<<y[5]
<<"\t"<<y[6] <<"\t"<<
                                 y[7] <<"\t"<<y[8] <<"\t"<<y[9]
<<"\t"<<y[10]<<"\t"<<

      y[11]<<"\t"<<y[12]<<"\t"<<y[13]<<"\t"<<y[14]<<"\t"<<

      y[15]<<"\t"<<y[16]<<"\t"<<y[17]<<"\t"<<y[18]<<"\t"<<

      y[19]<<"\t"<<y[20]<<"\t"<<y[21]<<"\t"<<y[22]<<"\t"<<//"\n";
                                 y[23]<<"\t"<<y[24]<<"\n";

                    outFS << t    <<"\t"<<y[0] <<"\t"<<y[1] <<"\t"<<y[2]
<<"\t"<<
                                  y[3] <<"\t"<<y[4] <<"\t"<<y[5]
<<"\t"<<y[6] <<"\t"<<
                                  y[7] <<"\t"<<y[8] <<"\t"<<y[9]
<<"\t"<<y[10]<<"\t"<<

y[11]<<"\t"<<y[12]<<"\t"<<y[13]<<"\t"<<y[14]<<"\t"<<

y[15]<<"\t"<<y[16]<<"\t"<<y[17]<<"\t"<<y[18]<<"\t"<<

y[19]<<"\t"<<y[20]<<"\t"<<y[21]<<"\t"<<y[22]<<"\t"<<//"\n";
                                  y[23]<<"\t"<<y[24]<<"\n";


      /*************************************************************/
      /******************** MRNA EQUATIONS **********************/
      /*************************************************************/


//                  // d[taR12]/dt - first term of the equation
//                  lambda = cmax[0]*T*tau;
//                  d1=PoissonRandomNumber(lambda);
//
//                  // d[taR12]/dt - second term of the equation
//                  lambda = d_y[0]*y[0]*tau;
//                  d2=PoissonRandomNumber(lambda);
//
//                  // d[taR12]/dt - equation
//                  y[0] = y[0] + d1 - d2;

//                  // d[taR12]/dt - first term of the equation
//                  lambda = cmax[8]*(a+(1-
a)*(pow((y[22]/K_y[22]),n_y[22]) /
//
(1+pow((y[22]/K_y[22]),n_y[22]))))
//                                                    *tau;
//                  d1=PoissonRandomNumber(lambda);
//
//                  // d[taR12]/dt - second term of the equation
//                  lambda = cmax[9]*(a+(1-
a)*(1/(1+pow((y[15]/K_y[15]),n_y[15])))
//
*(pow((y[23]/K_y[23]),n_y[23]) /
//
(1+pow((y[23]/K_y[23]),n_y[23]))))
//                                                          *tau;
//                  d2=PoissonRandomNumber(lambda);
//
//                  // d[taR12]/dt - third term of the equation
//                  lambda = d_y[0]*y[0]*tau;
//                  d3=PoissonRandomNumber(lambda);
//
//                  // d[taR12]/dt - equation
//                  y[0] = y[0] + d1 + d2 - d3;

                    // d[taR12]/dt - first term of the equation
                    lambda = cmax[8]*(a+(1-
a)*(1/(1+pow((y[17]/K_y[17]),n_y[17])))

*(1/(1+pow((y[23]/K_y[23]),n_y[23])))

*(pow((y[22]/K_y[22]),n_y[22]) /

      (1+pow((y[22]/K_y[22]),n_y[22]))))
                                                            *tau;
                    d1=PoissonRandomNumber(lambda);

                    // d[taR12]/dt - second term of the equation
                    lambda = cmax[9]*(a+(1-
a)*(1/(1+pow((y[22]/K_y[22]),n_y[22])))

*(pow((y[23]/K_y[23]),n_y[23]) /

      (1+pow((y[23]/K_y[23]),n_y[23]))))
                                                            *tau;
                    d2=PoissonRandomNumber(lambda);

                    // d[taR12]/dt - third term of the equation
                    lambda = cmax[13]*(a+(1-
a)*(1/(1+pow((y[15]/K_y[15]),n_y[15])))

*(1/(1+pow((y[22]/K_y[22]),n_y[22]))))
                                                             *tau;
                    d3=PoissonRandomNumber(lambda);

                    // d[taR12]/dt - fourth term of the equation
                    lambda = d_y[0]*y[0]*tau;
                    d4=PoissonRandomNumber(lambda);

                    // d[taR12]/dt - equation
                    y[0] = y[0] + d1 + d2 + d3 - d4;

                    // d[mCIcr]/dt - first term of the equation
                    lambda = cmax[1]*(a+(1-
a)*(1/(1+pow((y[14]/K_y[14]),n_y[14]))))*tau;
                    d1=PoissonRandomNumber(lambda);
```

```
                        // d[mCIcr]/dt - second term of the equation
                        lambda = d_y[1]*y[1]*tau;
                        d2=PoissonRandomNumber(lambda);

                        // d[mCIcr]/dt - equation
                        y[1] = y[1] + d1 - d2;

                        // d[mCI]/dt - first term of the equation
                        lambda = cmax[4]*(a+(1-
a)*(1/(1+pow((y[11]/K_y[11]),n_y[11])))

*(1/(1+pow((y[13]/K_y[13]),n_y[13])))

*(pow((y[12]/K_y[12]),n_y[12])/

(1+pow((y[12]/K_y[12]),n_y[12]))))
                                                                     *tau;
                        d1=PoissonRandomNumber(lambda);

                        // d[mCI]/dt - second term of the equation
                        lambda = d_y[2]*y[2]*tau;
                        d2=PoissonRandomNumber(lambda);

                        // d[mCI]/dt - equation
                        y[2] = y[2] + d1 - d2;

                        // d[mCII]/dt - first term of the equation
                        lambda = cmax[2]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))))*tau;
                        d1=PoissonRandomNumber(lambda);

                        // d[mCII]/dt - second term of the equation
                        lambda = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))

*(1/(1+pow((y[13]/K_y[13]),n_y[13])))

*(pow((y[12]/K_y[12]),n_y[12])/

(1+pow((y[12]/K_y[12]),n_y[12]))))
                                                                     *tau;
                        d2=PoissonRandomNumber(lambda);

                        // d[mCII]/dt - third term of the equation
                        lambda = d_y[3]*y[3]*tau;
                        d3=PoissonRandomNumber(lambda);

                        // d[mCII]/dt - equation
                        y[3] = y[3] + d1 + d2 - d3;

                        // d[mOmpR]/dt - first term of the equation
                        lambda = cmax[3]*tau;
                        d1=PoissonRandomNumber(lambda);

                        // d[mOmpR]/dt - second term of the equation
                        lambda = d_y[4]*y[4]*tau;
```

```
                    d2=PoissonRandomNumber(lambda);

                    // d[mOmpR]/dt - equation
                    y[4] = y[4] + d1 - d2;

                    // d[mGal4]/dt - first term of the equation
                    lambda = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))

*(1/(1+pow((y[13]/K_y[13]),n_y[13])))

*(pow((y[12]/K_y[12]),n_y[12])/

(1+pow((y[12]/K_y[12]),n_y[12]))))
                                                                        *tau;
                    d1=PoissonRandomNumber(lambda);

                    // d[mGal4]/dt - second term of the equation
                    lambda = d_y[5]*y[5]*tau;
                    d2=PoissonRandomNumber(lambda);

                    // d[mGal4]/dt - equation
                    y[5] = y[5] + d1 - d2;

                    // d[mTetR]/dt - first term of the equation
                    lambda = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))

*(1/(1+pow((y[13]/K_y[13]),n_y[13])))

*(pow((y[12]/K_y[12]),n_y[12])/

(1+pow((y[12]/K_y[12]),n_y[12]))))
                                                                        *tau;
                    d1=PoissonRandomNumber(lambda);

                    // d[mTetR]/dt - second term of the equation
                    lambda = cmax[6]*(a+(1-
a)*(1/(1+pow((y[17]/K_y[17]),n_y[17]))))*tau;
                    d2=PoissonRandomNumber(lambda);

                    // d[mTetR]/dt - third term of the equation
                    lambda = d_y[6]*y[6]*tau;
                    d3=PoissonRandomNumber(lambda);

                    // d[mTetR]/dt - equation
                    y[6] = y[6] + d1 + d2 - d3;

                    // d[mLexA]/dt - first term of the equation
                    lambda = cmax[6]*(a+(1-
a)*(1/(1+pow((y[17]/K_y[17]),n_y[17]))))*tau;
                    d1=PoissonRandomNumber(lambda);

                    // d[mLexA]/dt - second term of the equation
                    lambda = d_y[7]*y[7]*tau;
                    d2=PoissonRandomNumber(lambda);
                    // d[mLexA]/dt - equation
                    y[7] = y[7] + d1 - d2;

                    // d[mLacI]/dt - first term of the equation
                    lambda = cmax[4]*(a+(1-
a)*(1/(1+pow((y[11]/K_y[11]),n_y[11])))

*(1/(1+pow((y[13]/K_y[13]),n_y[13])))

*(pow((y[12]/K_y[12]),n_y[12])/

(1+pow((y[12]/K_y[12]),n_y[12]))))
                                                            *tau;
                    d1=PoissonRandomNumber(lambda);

                    // d[mLacI]/dt - second term of the equation
                    lambda = cmax[7]*(a+(1-
a)*(1/(1+pow((y[15]/K_y[15]),n_y[15]))))*tau;
                    d2=PoissonRandomNumber(lambda);

                    // d[mLacI]/dt - third term of the equation
                    lambda = d_y[8]*y[8]*tau;
                    d3=PoissonRandomNumber(lambda);

                    // d[mLacI]/dt - equation
                    y[8] = y[8] + d1 + d2 - d3;

                    // d[mGFP]/dt - first term of the equation
                    lambda = cmax[7]*(a+(1-
a)*(1/(1+pow((y[15]/K_y[15]),n_y[15]))))*tau;
                    d1=PoissonRandomNumber(lambda);

                    // d[mGFP]/dt - second term of the equation
                    lambda = d_y[9]*y[9]*tau;
                    d2=PoissonRandomNumber(lambda);

                    // d[mGFP]/dt - equation
                    y[9] = y[9] + d1 - d2;

//                  // d[mCI_4]/dt - first term of the equation
//                  lambda = cmax[4]*(a+(1-
a)*(1/(1+pow((y[11]/K_y[11]),n_y[11])))
//
*(1/(1+pow((y[13]/K_y[13]),n_y[13])))
//
*(pow((y[12]/K_y[12]),n_y[12])/
//
(1+pow((y[12]/K_y[12]),n_y[12]))))
//                                                          *tau;
//                  d1=PoissonRandomNumber(lambda);
//
//                  // d[mCI_4]/dt - second term of the equation
//                  lambda = d_y[19]*y[19]*tau;
//                  d2=PoissonRandomNumber(lambda);
//
//                  // d[mCI_4]/dt - equation
//                  y[19] = y[19] + d1 - d2;
```

```
                  // d[mAHL_LEFT]/dt - first term of the equation
                  lambda = cmax[11]*AHL_LEFT*tau;
                  d1=PoissonRandomNumber(lambda);

                  // d[mAHL_LEFT]/dt - second term of the equation
                  lambda = d_y[19]*y[19]*tau;
                  d2=PoissonRandomNumber(lambda);

                  // d[mAHL_LEFT]/dt - equation
                  y[19] = y[19] + d1 - d2;

//                // d[mCII_5]/dt - first term of the equation
//                lambda = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))
//
*(1/(1+pow((y[13]/K_y[13]),n_y[13])))
//
*(pow((y[12]/K_y[12]),n_y[12])/
//
(1+pow((y[12]/K_y[12]),n_y[12]))))
//                                                    *tau;
//                d1=PoissonRandomNumber(lambda);
//
//                // d[mCII_5]/dt - second term of the equation
//                lambda = d_y[20]*y[20]*tau;
//                d2=PoissonRandomNumber(lambda);
//
//                // d[mCII_5]/dt - equation
//                y[20] = y[20] + d1 - d2;

                  // d[mAHL_RIGHT]/dt - first term of the equation
                  lambda = cmax[12]*AHL_RIGHT*tau;
                  d1=PoissonRandomNumber(lambda);

                  // d[mAHL_RIGHT]/dt - second term of the equation
                  lambda = d_y[20]*y[20]*tau;
                  d2=PoissonRandomNumber(lambda);

                  // d[mAHL_RIGHT]/dt - equation
                  y[20] = y[20] + d1 - d2;

//                // d[mTetR_5]/dt - first term of the equation
//                lambda = cmax[5]*(a+(1-
a)*(1/(1+pow((y[10]/K_y[10]),n_y[10])))
//
*(1/(1+pow((y[13]/K_y[13]),n_y[13])))
//
*(pow((y[12]/K_y[12]),n_y[12])/
//
(1+pow((y[12]/K_y[12]),n_y[12]))))
//                                                    *tau;
//                d1=PoissonRandomNumber(lambda);
//
//                // d[mTetR_5]/dt - second term of the equation
//                lambda = d_y[21]*y[21]*tau;
//                d2=PoissonRandomNumber(lambda);
```

```
//
//                  // d[mTetR_5]/dt - equation
//                  y[21] = y[21] + d1 - d2;

                    // d[mAHL_CENTER]/dt - first term of the equation
                    lambda = cmax[10]*(a+(1-
a)*(1/(1+pow((y[16]/K_y[16]),n_y[16])))))*tau;
                    d1=PoissonRandomNumber(lambda);

                    // d[mAHL_CENTER]/dt - second term of the equation
                    lambda = d_y[21]*y[21]*tau;
                    d2=PoissonRandomNumber(lambda);

                    // d[mAHL_CENTER]/dt - equation
                    y[21] = y[21] + d1 - d2;

//                  // d[mLacI_4]/dt - first term of the equation
//                  lambda = cmax[4]*(a+(1-
a)*(1/(1+pow((y[11]/K_y[11]),n_y[11])))
//
*(1/(1+pow((y[13]/K_y[13]),n_y[13])))
//
*(pow((y[12]/K_y[12]),n_y[12])/
//
(1+pow((y[12]/K_y[12]),n_y[12]))))
//                                                                     *tau;
//                  d1=PoissonRandomNumber(lambda);
//
//                  // d[mLacI_4]/dt - second term of the equation
//                  lambda = d_y[22]*y[22]*tau;
//                  d2=PoissonRandomNumber(lambda);
//
//                  // d[mLacI_4]/dt - equation
//                  y[22] = y[22] + d1 - d2;


      /************************************************************/
      /****************** PROTEIN EQUATIONS *********************/
      /************************************************************/


//                  //[CI] - first term of the equation
//                  lambda = gp*y[1]*(a+(1-a)*(pow((y[0]/K_y[0]),n_y[0])/
//
      (1+pow((y[0]/K_y[0]),n_y[0]))))*tau;
//                  d1=PoissonRandomNumber(lambda);

                    //[CI] - first term of the equation
                    lambda = gp*y[1]*(a+(1-a)*y[0]*0.04)*tau;
                    d1=PoissonRandomNumber(lambda);

                    //[CI] - second term of the equation
                    lambda = gp*y[2]*tau;
                    d2=PoissonRandomNumber(lambda);

                    //[CI] - third term of the equation
                    lambda = d_y[10]*y[10]*tau;
```

```
d3=PoissonRandomNumber(lambda);

//[CI] - equation
y[10] = y[10] + d1 + d2 - d3;

//[CII] - first term of the equation
lambda=gp*y[3]*tau;
d1=PoissonRandomNumber(lambda);

//[CII] - second term of the equation
lambda = d_y[11]*y[11]*tau;
d2=PoissonRandomNumber(lambda);

//[CII] - equation
y[11] = y[11] + d1 - d2;

//[OmpR] - first term of the equation
lambda = gp*y[4]*tau;
d1=PoissonRandomNumber(lambda);

//[OmpR] - second term of the equation
lambda = ((V_phos*(1-L)*y[12])/(K_phos + y[12]))*tau;
d2=PoissonRandomNumber(lambda);

//[OmpR] - third term of the equation
lambda = V_dephos*y[13]*tau;
d3=PoissonRandomNumber(lambda);

//[OmpR] - fourth term of the equation
lambda = d_y[12]*y[12]*tau;
d4=PoissonRandomNumber(lambda);

//[OmpR] - equation
y[12] = y[12] + d1 - d2 + d3 - d4;

//[OmpRP] - first term of the equation
lambda=((V_phos*(1-L)*y[12])/(K_phos + y[12]))*tau;
d1=PoissonRandomNumber(lambda);

//[OmpRP] - second term of the equation
lambda = V_dephos*y[13]*tau;
d2=PoissonRandomNumber(lambda);

//[OmpRP] - third term of the equation
lambda = d_y[13]*y[13]*tau;
d3=PoissonRandomNumber(lambda);

//[OmpRP] - equation
y[13] = y[13] + d1 - d2 - d3;

//[Gal4] - first term of the equation
lambda=gp*y[5]*tau;
d1=PoissonRandomNumber(lambda);

//[Gal4] - second term of the equation
lambda = d_y[14]*y[14]*tau;
d2=PoissonRandomNumber(lambda);
```

```
//[Gal4] - equation
y[14] = y[14] + d1 - d2;

//[TetR] - first term of the equation
lambda=gp*y[6]*tau;
d1=PoissonRandomNumber(lambda);

//[TetR] - second term of the equation
lambda = d_y[15]*y[15]*tau;
d2=PoissonRandomNumber(lambda);

//[TetR] - equation
y[15] = y[15] + d1 - d2;

//[LexA] - first term of the equation
lambda=gp*y[7]*tau;
d1=PoissonRandomNumber(lambda);

//[LexA] - second term of the equation
lambda = d_y[16]*y[16]*tau;
d2=PoissonRandomNumber(lambda);

//[LexA] - equation
y[16] = y[16] + d1 - d2;

//[LacI] - first term of the equation
lambda=gp*y[8]*tau;
d1=PoissonRandomNumber(lambda);

//[LacI] - second term of the equation
lambda = d_y[17]*y[17]*tau;
d2=PoissonRandomNumber(lambda);

//[LacI] - equation
y[17] = y[17] + d1 - d2;

//[GFP] - first term of the equation
lambda=gp*y[9]*tau;
d1=PoissonRandomNumber(lambda);

//[GFP] - second term of the equation
lambda = d_y[18]*y[18]*tau;
d2=PoissonRandomNumber(lambda);

//[GFP] - equation
y[18] = y[18] + d1 - d2;

//[AHL_LEFT] - first term of the equation
lambda=gp*y[19]*tau;
d1=PoissonRandomNumber(lambda);

//[AHL_LEFT] - second term of the equation
lambda = d_y[22]*y[22]*tau;
d2=PoissonRandomNumber(lambda);

//[AHL_LEFT] - equation
```

```
        y[22] = y[22] + d1 - d2;

        //[AHL_RIGHT] - first term of the equation
        lambda=gp*y[20]*tau;
        d1=PoissonRandomNumber(lambda);

        //[AHL_RIGHT] - second term of the equation
        lambda = d_y[23]*y[23]*tau;
        d2=PoissonRandomNumber(lambda);

        //[AHL_RIGHT] - equation
        y[23] = y[23] + d1 - d2;

        //[AHL_CENTER] - first term of the equation
        lambda=gp*y[21]*tau;
        d1=PoissonRandomNumber(lambda);

        //[AHL_CENTER] - second term of the equation
        lambda = d_y[24]*y[24]*tau;
        d2=PoissonRandomNumber(lambda);

        //[AHL_CENTER] - equation
        y[24] = y[24] + d1 - d2;

}

outF.close();
outFS.close();

cout << "Done!\n" << endl;

time (&end);
diff = difftime(end, start);

cout << "Time Elapsed: " << itime(diff) << endl << endl;

char ans;
cout << "Plot Graph(s)? (Yes/No)" << endl;
cin >> ans;
if (ans!='y' && ans!='Y')
{
        cout << endl << "End!" << endl;
        return 0;
}

cout << "Plotting...\n" << endl;

try
{
        // c is "gnuplot.exe test.gp"
        system(gnuPlot);
}
catch (invalid_argument& e)
{
        cerr << "ERROR: " << e.what();
}
```

```
        cout << endl << "End!" << endl;

        return 0;
    }


    /********************************************************/
    /********************** FUNCTIONS ***********************/
    /********************************************************/


    const int PoissonRandomNumber(const double lambda)
    {
        int k=0;                                        //Counter
        const int max_k = int (2 * lambda); //k upper limit
        double p = 1.0*rand()/RAND_MAX;           //uniform random
number
        double P = exp(-lambda);                  //probability
        double sum = P;                                 //cumulant

        if (sum >= p) return 0;                   //done allready

        for (k = 1; k < max_k; ++k)               //Loop over all
k:s
        {
            P*=lambda/(double)k;                  //Calc next prob
            sum+=P;
    //Increase cumulant
            if (sum>=p) break;                          //Leave
loop
        }

        return k;                                       //return
random number
    }

    bool fexists(const char *filename)
    {
      ifstream ifile(filename);
      return ifile;
    }

    string getDate()
    {
       time_t now;
       char theDate[MAX_DATE_LEN];

       theDate[0] = '\0';

       now = time(0);

       if (now != -1)
       {
//            strftime(theDate, MAX_DATE_LEN,
//                        "%Y%h%d_%Hh%Mm%Ss", localtime(&now));
              strftime(theDate, MAX_DATE_LEN, "%Y%m%d",
localtime(&now));
```

```
   }

   return string(theDate);
}

string nextFileName()
{
      int count=0;
      stringstream ss;
      string date = getDate();

      do
      {
            count++;
            ss.str("");

            ss    << path2
                  << date
                  << "_STO_"
                  << setw(3) << setfill('0') << count
                  << ".xls";
      }
      while (fexists(ss.str().c_str()));

      return ss.str();
}

string itime(const double diff)
{
      stringstream ss;

      int hrs = int (diff/3600);
      int mins = int ((diff - hrs*3600)/60);
      int secs = int ((diff - hrs*3600 - mins*60));

      ss    << setw(2) << setfill('0')
            << hrs << ":"
            << setw(2) << setfill('0')
            << mins << ":"
            << setw(2) << setfill('0')
            << secs;

      return ss.str();
}
```

GnuPlot file:

stochastic.plt

```
#!/gnuplot
#
#
#     G N U P L O T
#     Version 4.2 patchlevel 5
#     last modified Mar 2009
#     System: MS-Windows 32 bit
#
#     Copyright (C) 1986 - 1993, 1998, 2004, 2007 - 2009
#     Thomas Williams, Colin Kelley and many others
#
#     Type `help` to access the on-line reference manual.
#     The gnuplot FAQ is available from http://www.gnuplot.info/faq/
#
#     Send bug reports and suggestions to
<http://sourceforge.net/projects/gnuplot>
#
# set terminal windows color noenhanced
# set output
reset
GNUTERM = "win"

set xlabel "Time (hours)"
set ylabel "Protein Levels (uM)"

set lmargin at screen 0.055

set yrange [-2:18]

set ytics 4
set xtics 4

set object 1 rectangle from 4000/3600.0, graph 0 to 8500/3600.0, graph
1 fc lt 1 fs transparent solid 0.25 noborder
set object 2 rectangle from 18000/3600.0, graph 0 to 23000/3600.0,
graph 1 fc lt 1 fs transparent solid 0.25 noborder
set object 3 rectangle from 28000/3600.0, graph 0 to 34000/3600.0,
graph 1 fc lt 1 fs transparent solid 0.25 noborder
set object 4 rectangle from 40000/3600.0, graph 0 to 44000/3600.0,
graph 1 fc lt 1 fs transparent solid 0.25 noborder

set object 5 rectangle from 2500/3600.0, graph 0 to 10000/3600.0, graph
1 fc lt -1 fs transparent pattern 7 noborder
set object 6 rectangle from 31000/3600.0, graph 0 to 42000/3600.0,
graph 1 fc lt -1 fs transparent pattern 7 noborder

plot "stochastic.dat" using ($1/3600.0):($12/1000.0) t "CI" w l lc rgb
"#008000" lw 1.5, "stochastic.dat" u ($1/3600.0):($13/1000.0) t "CII" w
l lt 1 lw 1.5, "stochastic.dat" u ($1/3600.0):($14/1000.0) t "OmpR" w l
lc rgb "#008080" lw 1.5, "stochastic.dat" u ($1/3600.0):($15/1000.0) t
"OmpRP" w l lc rgb "#FF8000" lw 1.5, "stochastic.dat" using
($1/3600.0):($16/1000.0) t "Gal4" w l lc rgb "#808000" lw 1.5,
"stochastic.dat" using ($1/3600.0):($17/1000.0) t "TetR" w l lt 4 lw
1.5, "stochastic.dat" using ($1/3600.0):($19/1000.0) t "LacI" w l lt -1
lw 1.5

#    EOF
```