Methods and Tools for the Analysis, Verification and Synthesis of Genetic Logic Circuits,

Baig, Hasan; Madsen, Jan; Pop, Paul

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Methods and Tools for the Analysis, Verification and Synthesis of Genetic Logic Circuits

Hasan Baig

Kongens Lyngby 2017
PhD-2017-456

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk PhD-2017-456

Summary (English)

Synthetic biology has emerged as an important discipline in which engineers and biologists are working together to design new and useful biological systems composed of genetic circuits. The purpose of developing genetic circuits is to carry out desired logical functions inside a living cell. This usually requires simulating the mathematical models of these genetic circuits and perceive whether or not the circuit behaves appropriately. Furthermore, synthetic biology utilizes the concepts from electronic design automation (EDA) of abstraction and automated construction to generate genetic circuits with the aim to reduce the in-vitro (wet-lab) experiments. To address this, several automated tools have been developed to improve the process of genetic design automation (GDA) with different capabilities. This thesis attempts to contribute to the advancement of GDA tools by introducing capabilities which we believe that no other existing GDA tools support.

First, we introduce a user-friendly simulation tool, called D-VASim, which allows a user to perform virtual laboratory experimentation by dynamically interacting with the model during runtime. This dynamic interaction with the model gives user a feeling of being in the lab performing wet-lab experiments virtually. This tool allows users to perform both deterministic and stochastic simulations.

Next, this dissertation introduces a methodology to perform timing analyses of genetic logic circuits, which allows a user to analyze the threshold value and propagation delays of genetic logic circuits. In this thesis, it has been demonstrated, through *in-silico* experimentation, that the threshold value and propagation delay plays a vital role in the correct functioning of a genetic circuit. It has also been shown how some circuit parameters effect these two important design characteristics.

This thesis also introduces an automated approach to analyze the behavior of genetic logic circuits from the simulation data. With this capability, the boolean logic of complex genetic circuits can be analyzed and/or verified automatically. It is also shown in this thesis that the proposed approach is effective to determine the variation in the behavior of genetic circuits when the circuit's parameters are changed.

In addition, the thesis also attempts to propose a synthesis and technology mapping tool, called GeneTech, for genetic circuits. It allows users to construct a genetic circuit by only specifying its behavior in the form of a boolean expression. For technology mapping, this tool uses gate library developed by the collective efforts of the researchers at MIT and Boston universities. It is shown experimentally that the tool is able to provide all feasible solutions, containing different genetic components, to achieve the specified boolean behavior.

Finally, it has been shown how D-VASim can be used along with other tools for useful purposes, like model checking. With respect to this, an experimental workflow is proposed for checking genetic circuits using the statistical model checking (SMC) utility of the Uppaal tool and the timing analysis capability of D-VASim. We further demonstrated how the reliability of a simulation can be improved by using real parameter values. In this regard, the relationship between the simulation parameters and real parameters have been derived.

Summary (Danish)

Syntetisk biologi er en ny og vigtig disciplin i hvilken ingeniører og biologer arbejder sammen om at designe nye og nyttige biologiske systemer, bestående af genetiske kredsløb. Formålet med udvikling af genetiske kredsløb er at udføre ønskede logiske funktioner i en levende celle. For at teste om det genetiske kredsløb opfører sig korrekt, udføres der normalt simuleringer af en model af det genetiske kredsløb. Syntetisk biologi udnytter begreber fra elektronisk designautomatisering (EDA), så som abstraktion og automatiseret konstruktion, til at genere genetiske kredsløb med det formål at reducere antallet af in-vitro forsøg (vådrums laboratorie eksperimenter), der ellers kræves for at finde en korrekt løsning. Forskere har i de senere år udviklet en række forskellige genetiske designautomatiserings (GDA) værktøjer til at forbedre denne proces. Denne afhandling bidrager til udviklingen af GDA værktøjer ved at introducere egenskaber, som vi mener at ingen eksisterende GDA-værktøj understøtter.

Først introducerer vi et brugervenligt simuleringsværktøj, kaldet D-VASim, som tillader brugeren at udføre virtuelle laboratorieforsøg gennem dynamisk interaktion med modellen under simuleringen. Denne dynamiske interaktion med modellen giver brugeren en følelse af at være i laboratoriet. D-VASim giver brugerne mulighed for at udføre både deterministiske og stokastiske simuleringer. Oven på simuleringsmodellen er der udviklet en metode til at udføre tidsanalyser af genetiske logiske kredsløb. Denne giver brugeren mulighed for at analysere logiske tærskelværdier og kredsløbsforsinkelser af genetiske logiske kredsløb. I denne afhandling er det, gennem in-silico eksperimenter, blevet påvist at tærskelværdien og kredsløbsforsinkelsen spiller en afgørende rolle for den korrekte funktion af et genetisk kredsløb, samt visse genetiske kredsløbsparametre påvirker disse to vigtige designegenskaber.

Denne afhandling introducerer også en automatiseret analyse af opførslen af genetiske logiske kredsløb ud fra simuleringsdata. Herved kan boolesk logik af komplekse genetiske kredsløb analyseres og / eller verificeres automatisk. Det fremgår også af denne afhandling, at analysen er effektiv til at bestemme variationerne i opførsel af et givet genetiske kredsløb, når kredsløbets parametre ændres.

Endelig introducerer afhandlingen et syntese- og teknologi-mapings værktøj for genetiske kredsløb, kaldet GeneTech. Dette værktøj giver brugeren mulighed for at konstruere et genetisk kredsløb ved blot at specificere dets ønskede opførsel i form af booleske udtryk. Til teknologi-maping, bruges et genetisk komponent bibliotek udviklet af forskere ved MIT og Boston universitetet. Det vises eksperimentelt at GeneTech er i stand til at finde samtlige mulige løsninger, der potentielt kan realisere den specificerede booleske opførsel. Endelig er det vist, hvordan D-VASim kan bruges sammen med andre værktøjer, som f.eks. model checking. Der foreslås et eksperimentelt workflow til validering af genetiske kredsløb ved hjælp af det statistiske model checking (SMC) -værktøjet i Uppaal værktøjet og tidsanalyse delen af D-VASim. Yderligere demonstrerer vi, hvordan pålideligheden af en simulering kan forbedres ved bruge de reelle parameterværdier. I denne henseende er forholdet mellem simuleringsparametre og reelle parametre er blevet afledt.

Preface

This thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark (DTU) in fulfillment of the requirements for acquiring the degree of Doctor of Philosophy (PhD).

The thesis deals with the methods and tools for the simulation, analysis and synthesis of genetic logic circuits.

The research work presented in this thesis has been supervised by Professor Jan Madsen and co-supervised by Professor Paul Pop.

Lyngby, 14-August-2017

[illegible]

Hasan Baig

Research Dissemination

The research work presented in this thesis has been disseminated in the following:

Journals

- Hasan Baig and Jan Madsen, "Simulation Approach for Timing Analysis of Genetic Logic Circuits", *ACS Synthetic Biology*, 2017, 6 (7), pp 1169–1179. *Published.*
- Hasan Baig and Jan Madsen, "D-VASim – An Interactive Virtual Laboratory Environment for the Simulation and Analysis of Genetic Circuits", *Bioinformatics*, vol. 33, issue 2, pp. 297–299, 2016. *Published.*
- Hasan Baig and Jan Madsen, "An Automated Approach to Verify the Logic of Genetic Circuits from Experimental data", *ACM Journal on Emerging Technologies. Under review.*
- Hasan Baig and Jan Madsen, "*GeneTech*: "A Technology Mapping Tool for Genetic Logic Circuits", *IEEE Transactions on Biomedical Engineering. Under review.*

Conferences

- Hasan Baig and Jan Madsen, "Taming Living Logic using Formal Methods", *Models, Algorithms, Logics and Tools*, LNCS Springer, 10460, pp. 503–515, 2017. *Published.*
- Hasan Baig and Jan Madsen, "Logic Analysis and Verification of n-input Genetic Logic Circuits", *Design Automation and Test in Europe (DATE)*, pp. 654–657, 2017. *Published.*

Workshops

- Hasan Baig and Jan Madsen, "A Top-down Approach to Genetic Circuit Synthesis and Optimized Technology Mapping", *9th International Workshop on Bio-Design Automation (IWBDA)*, pp. 28–29, 2017. *Published.*

- Hasan Baig and Jan Madsen, "Logic and Timing Analysis of Genetic Logic Circuits using D-VASim", *8th International Workshop on Bio Design Automation (IWBDA)*, pp. 77–78, 2016. *Published.*

- Hasan Baig and Jan Madsen, "D-VASim: Dynamic Virtual Analyzer and Simulator for Genetic Circuits", *7th International Workshop on Bio-Design Automation (IWBDA)*, pp. 48–49, 2015. *Published.*

Posters and Practical Demonstrations

- Hasan Baig and Jan Madsen, "Timing Analysis of Genetic Logic Circuits using D-VASim", *At the University Booth in the Design Automation and Test in Europe (DATE)*, 2016. *Presented and Published.*

- Hasan Baig and Jan Madsen, "Analysis and Verification of Genetic Logic Circuits using D-VASim", *At the 2nd Synthetic and Systems Biology Summer School (SSBSS)*, 2015. *Presented.*

Industrial Case Studies

- Hasan Baig and Jan Madsen, "D-VASim: A Software Tool to Simulate and Analyze Genetic Logic Circuits", published by National Instruments (http:\www.ni.com) and can be seen at http://sine.ni.com/cs/app/doc/p/id/cs-17225. *Published.*

Acknowledgements

I would first like to thank Allah Almighty Who empowered me with the devoting nature to carry out this work with full passion, enthusiasm and consistency. It is only because of His blessings that helped me to start over after every failure with a belief that *everything happens for a good reason.*

I would then like to express my sincere gratitude to my supervisor, Prof. Jan Madsen, who gave me an opportunity to work on such an interesting and emerging field of technology. The research carried out in this thesis would not have reached to this level without Jan's invaluable guidance, endless support, constructive feedbacks and time-to-time appreciations. I also thank him for encouraging and supporting me to attend relevant events to get myself updated with the ongoing research. I feel that I can never pay him back for his support and favors, but I will forever be grateful to him. I cannot imagine if there could be a better way of mentoring someone than how Jan supervised me. One day, when I will become a supervisor, I strongly believe that I will follow his example of mentorship - a courteous, humble and encouraging attitude which force students to continue working with their mentors passionately.

I further acknowledge Jan that he introduced me to many outstanding individuals in this field of research and I am thankful to all of them for their help. In particular, I would honestly like to thank Prof. Chris J. Myers for his constant support and help in clarifying my each and every confusion related to the topic. He not only helped me solving issues related to iBioSim tool, but also helped me getting my problems solved related to Cello tool. I also like to thank Prof. Douglas Densmore, for providing me an opportunity to work in CIDAR lab at Boston university as a visiting student. The ideas I gathered during this visit motivated me to work on design/synthesis field as well. I am also thankful to Dr. Nicholas Roehner for reviewing the derived relationship between real and simulation parameters; and Arash Khoshparvar for assisting me in solving issues related to SBOL-SBML conversion and providing me with the correct SBOL models of genetic circuits. Without these genetic circuit models, I could have never experimentally proved the concepts presented in this thesis.

In addition, I am greatly indebted to the wonderful staff and colleagues at DTU Compute, specifically Karin Tunder for her boundless help in administrative matters; Henning Christiansen from the IT department for his help in solving issues related to BDA-Compute web page; and DTU itself for supporting my research studies. Honestly, I have never experienced such a wonderful, well-organized, respectable and a friendly working environment ever in my life before. I also like to thank my friends Qasim Rajpoot and Kamran Manzoor for their help in solving compiler-related issues.

I am also grateful to my parents and siblings for their support and love, specially to my father, Prof. Jawed Altaf Baig. I would not be a person I am today without his moral support and guidance in every walk of my life. Last but most important, I would like to thank my ever best friend and wife, Sumaiya Hasan, for her unconditional love and comfort she has given me specially during the whole span of my PhD research. I acknowledge that she had not only been patient with me during my hard times but also console me enormously to start over, after every failure, with new enthusiasm. My family has been a continuous source of my happiness and looking at them always motivated me to work harder for their good future as well. I am also grateful for Sumaiya's sacrifices she only made to take good care of our children (*Saleh and Nourah*), whenever I was away or busy in achieving hard deadlines and project goals. Without any hesitations, I can say that she had struggled equally to help me accomplish this lifetime achievement.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

An advancement in the understanding of cellular processes and DNA synthesis methods suggests that the living cells can be viewed as a *programmable* matter. With this revolutionary finding, logical computations can be performed inside a living cell through a group of biological components, collectively called *genetic circuits.* A genetic circuit represents a gene regulatory network (GRN), which is composed of small genetic components. These components interact with the external signals (like temperature, light, proteins etc.) to control the behavior of a living cell.

Genetic circuits are a key application of *synthetic biology* which is an emerging engineering discipline to program cell behaviors as easy as computers are programmed. Synthetic biology is defined by syntheticbiology.org as;

"***(a)*** *the design and construction of new biological parts, devices and systems and* ***(b)*** *the re-design of existing natural biological systems for useful purposes*".

Biologists are interested in synthetic biology because it provides a viewpoint to analyze, understand, design and ultimately build a biological system. Engineers, on the other hand, are attracted towards synthetic biology because the living world has the abundant mechanisms for controlling life behavior and processing information.

1.1 Why computations in cells?

There are numerous complex computations a living cell performs on the continuous environmental signals they encounter. The natural biological systems can be engineered to perform sophisticated computations in living cells. Biologists and engineers are working together on synthetic biology [1] to design new and useful biological systems. The synthetic biological systems performing digital [2] and analog [3] computations have already been implemented.

The artificial computation in living cells will revolutionize the industry of medicine and biotechnology. The aim of performing synthetic computations in living cell is to develop genetic devices to address real-world problems. These problems include the development of genetic systems to detect and destroy cancer cells [4]; production of liquid biofuels to address the global energy and environmental problems [5]; consuming toxic wastes to avoid environmental pollution; and the production of drugs to treat health problems like Malaria [6], to name a few.

1.2 State-of-the-art

Similar to *electronic design automation* (EDA) processes which dramatically enhanced the design, verification, validation and production of electronic circuits, researchers have started to work on the development of *genetic design automation* (GDA) tools [7] to automate the design, test, verification and synthesis processes of genetic circuits prior to their validation in laboratory. There are several GDA tools (see Section 2.4) which allow synthetic biologists to design genetic circuits at a high level of abstraction with the focus on a desired function, rather than exact genetic components used to achieve this functionality. By encoding standardized data, genetic constraints, and the components library in GDA tools, the process of genetic circuit construction and analysis has been automated. This not only has reduced the lengthy design process and iterative tests for constructing complex genetic circuits, but has also promoted the reuse of experimentally tested genetic components.

The modern trend to analyze genetic circuits is to perform *in-silico* (in computer) analysis either by solving ordinary differential equations (ODEs) or by performing stochastic simulations, with the aim to reduce the number of required in-vitro (in laboratory) experiments. In order to perform these analyses in a computer, models of biological systems must be represented in a standard computerized format. Several different methods have been proposed to represent and analyze genetic systems [8]. Among these methods, the most widely used standards to represent the *behavior* and the *structure* of a genetic model are the Systems Biology Markup Language (SBML) [9], and the Synthetic Biology Open Language (SBOL) [10], respectively. Unfortunately, no single standard exists which can be used to represent both the behavior and the structure of a biological model. SBML allow users to define the behavior of a circuit by specifying the species of a genetic network and how they interact with each other through chemical kinetics. SBOL, however, is used to illustrate genetic designs graphically with the help of standardized vocabulary of schematic glyphs (SBOL Visual) as well as standardized digital data (SBOL Data). More information on *standards* can be found in Section 2.3.

1.3 Motivation

Synthetic biology not only aims to play with natural biological systems but also to construct artificial complex systems from the library of well-characterized biological components, in a similar way as electronic circuits are designed and constructed. While comparison with electronic circuits is useful, there are several important challenges which make the design of genetic circuits more challenging. For instance, genetic components, in contrast to electronic components, are not physically separated from each other. This not only makes the reuse of genetic components in the same system more difficult, but also increases the cross-talk with the neighboring circuit components. Also, the electronic logic gates are composed of transistors which have well-defined and uniform threshold voltage levels that categorizes the logic levels 0 and 1. However, in genetic circuits, each genetic gate is composed of different genetic components which results in the different threshold concentration values. Additionally, in comparison to electronic circuits which have the same physical quantities as input and output signals, the genetic circuits have different species at the input and other at the output, which makes the genetic modules integration more difficult.

As electronic engineers develop circuits using electronic logic gates (such as AND, NAND, and NOT gates), genetic engineers use biological equivalents of these components to control the function of a cell [2, 11]. The field of genetic circuit design is still immature and only small circuits, containing limited number of genes, can be constructed in the laboratory. However, genetic circuits can be designed from a very large number of genetic parts [12] creating a large space of possible solutions even for circuits of limited complexity.

The current practice is to design such circuits directly in the laboratory, through trial and error, which is a time consuming and costly process, as thousands of circuits may have to be tested in order to find a few that works. Due to this, the process of design and implementation of genetic circuits remains very slow. To address these challenges, it is necessary to improve computer aided design (CAD) tools to speed up the design and analyses procedures of genetic circuits. In particular, it is necessary to develop tools which allow genetic design engineers to capture and analyze the stochastic behavior of biological systems dynamically in a way that sounds natural to them.

1.4 Present Challenges

An electronic design engineer would never fabricate a circuit on silicon prior to its functional validation and behavioral analysis. Similarly the most important phase in GDA is the simulation and *in-silico* (in computer) analysis of genetic circuit models to increase the chances that the system would work *in-vivo* (in living organism) correctly. There are plenty of tools developed to simulate the behavior of genetic circuits [13]. These tools, however, lack some important and useful features which can not only increase the designer's productivity but also help them design genetic circuit models more effectively. Out of many challenges in the field of GDA, some of the challenges, listed below, have been addressed in this thesis. We believe that addressing the following challenges will not only increase the productivity of genetic design engineer but will also increase the reliability and robustness of genetic circuit models.

1.4.1 Virtual experimentation

First, it would be very helpful for biologist or design engineers to have a tool which allow them to perform laboratory experiments virtually in-silico. This corresponds to an experimental environment where a user can trigger the concentrations of input species or change the parameter values (for example, increasing temperature) at any instant of time and observe their live effects on the model's behavior. For in-silico analyses, the standard way to capture the instantaneous, discontinuous state change in the model is by defining *events* (see Section 2.3 for more details). For example, events (shown as green-boxes in Figure 2.6(b)) are used to trigger the concentration of input species to a certain level, at a specific point in time, and to observe the effects on the concentration of output species. A single event can be used to represent only one instance of triggering the concentration to a certain level at a specific time. Since events are predefined, they cannot be changed during runtime, which means that the output of a genetic circuit can be observed only for defined events. In order to observe the output, the different set of input conditions, i.e., when to change what input to which level, must be defined in each event. Even for moderate sized genetic circuits, capturing all combinations of inputs and concentration levels may require a very large number of events to be defined and simulated. To the best of our knowledge, there exist no tools that allow users to trigger/change input species on the fly during the simulation, effectively creating a *virtual lab.*

1.4.2 Timing and threshold analysis

In contrast to EDA tools which allows a user to perform timing analyses, to the extent of our knowledge, no GDA tool allows a user to perform timings and threshold value analyses for genetic gates/circuits. Electronic design engineers do not need to estimate the threshold value for each electronic circuit because these values are well defined and holds valid for all electronic logic gates. However, this is not the case for genetic gates where each of them are composed of different components and have different input and output molecular identities. Therefore each genetic gate may have different input threshold values and thus exhibit different timing behaviors. It is therefore necessary to have such a tool which should assists a user in identifying the correct input threshold concentration required to trigger the circuit's output along with the estimation of propagation delays. It may also help a user to perform in-vitro experimentation quickly by applying the estimated threshold concentration values at input (rather than following *trial-and-error* approach) and expect the circuit's output to be triggered approximately within the time estimated as a propagation delay.

Similar to electronic circuits where timing analysis is a vital design characteristic, the timing analysis may also become an essential design characteristic in genetic circuits. It is therefore very important to have such analyses *in-silico* prior to the circuit's implementation *in-vivo.*

1.4.3 Automatic logic validation

It is also interesting to automatically validate if the behaviour of a genetic circuit complies with the design rules. For example, the behavior of a genetic AND gate can be validated by applying all the possible input combinations and determine if the circuit's response obeys the AND Boolean logic. It might be easier to analyze the logical behavior of small circuits by just looking at response curves, but it may become a cumbersome task if the behavior is to be validated manually for complex genetic circuits. Therefore, an automated approach for analyzing the logic in genetic circuits will be helpful.

1.4.4 Effortless circuit designing

One of the several challenges in making the design process of genetic gates easier and user-friendly is to let the designers construct genetic circuits at a very high level of abstraction. Recently a tool, named Cello, is developed [14] which allows users to program genetic circuits as easy as electronic circuits are designed through *hardware description language* (HDL). Cello provides user, specially *computer scientist*, a fairly high-level of abstraction to develop genetic circuits without worrying about the underlying physics of genetic interactions. However, this still requires a *biologist* to learn programming principles and the syntax in which the design module should be written. To let the *biologists* design genetic circuits effortlessly without additional prerequisites, a further simple and straightforward mechanism should be developed.

1.5 Thesis Contributions

The main aim of this research is to enhance the advancement of GDA tools for analysis, verification, and synthesis of genetic logic circuits. The contributions of this dissertation are the development of the following methods and tools to address the challenges mentioned in Section 1.4:

- **Virtual laboratory simulation environment (D-VASim)**

  A simulation tool, named D-VASim (Dynamic Virtual Analyzer and Simulator) is developed, which allows a user to carry out *virtual lab experiments* as an interactive process during runtime, rather than a batch process which is a current practice. It is a user-friendly software with an intuitive graphical user interface, and allows a user to perform both deterministic as well as stochastic simulations. This software tool is available to download freely for public use from http://bda.compute.dtu.dk/downloads/d-vasim/.

- **Timing Analyzer (*A plugin to D-VASim*)**

  A methodology is introduced to perform the timing and threshold value analysis of genetic logic circuits. This methodology is integrated in D-VASim as a plugin tool.

- **Logic Analyzer (*A plugin to D-VASim*)**

  A method is introduced to validate the boolean logic of a genetic circuit from the stochastic simulation data. This boolean logic analysis algorithm is scalable and able to analyse n-input genetic logic circuits through an automated process. This tool is also integrated to D-VASim as a plugin tool.

- **GeneTech (*Standalone tool*)**

  An optimization, synthesis and technology mapping tool for genetic logic circuits. This tool is able to construct a genetic circuit using the library of genetic gates developed at MIT and Boston universities. It is currently a standalone tool but can be integrated to D-VASim as a plugin. GeneTech provides a user an ability to develop genetic circuits only by specifying its behavior in the form of a Boolean expression. With this ability, users, specially biologists, do not need to learn any additional programming language for designing genetic circuits. This tool can be downloaded from `http://bda.compute.dtu.dk/downloads/genetech/`.

- **Use of D-VASim with other tools**

  Besides above mentioned contributions, the research is further extended to demonstrate how D-VASim tool can be used along with existing tools for useful purposes. In particular, the experimental flow for model checking of genetic circuits, using Uppaal [15] and D-VASim [16], has been proposed. Also, an effort has been made to use Cello [14] parameters for simulating the models of real genetic circuits.

Figure 1.1 shows a very high level diagram which describes how the major contributions made in this dissertation can be used. Having the SBML model of a genetic circuit in D-VASim, users can perform ODE and stochastic simulations in an environment which gives them a feeling of being in the lab performing live experiments by interacting with the model during run-time. Similar to many EDA tools which allow hardware design engineers to perform timing analysis of electronic circuits, D-VASim is the first tool which provides users an ability to perform the timing analysis of genetic circuits. Furthermore, the experimental data, generated from stochastic simulations, can be used to analyze the logical behavior of a genetic circuit. Another tool, *GeneTech*, takes a raw Boolean expression as an input and generates all the possible circuits (in the form of structure) to achieve a desired logic. The dotted line between GeneTech and the D-VASim logic analyzer shows that the Boolean expression generated from the logic analyzer can also be used to obtain other possible circuits for the model being simulated. The circuits are generated using the genetic gates library [14]. The generated models of genetic circuit can then be synthesized into SBML form using any SBML-synthesis tool (like iBioSim [17]) and then can be analyzed back in D-VASim again.

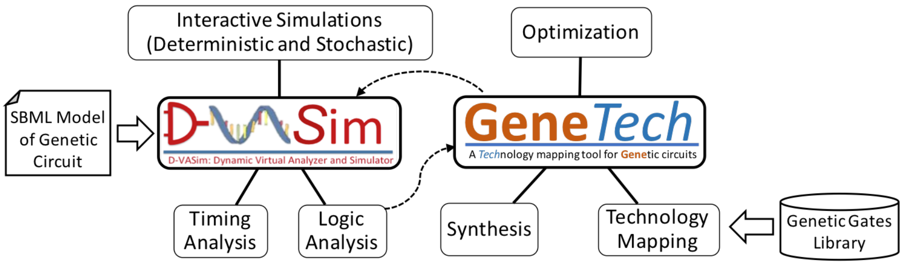

Interactive Simulations
(Deterministic and Stochastic)
Optimization
SBML Model of Genetic Circuit
D-VASim
D-VASim: Dynamic Virtual Analyzer and Simulator
GeneTech
A Technology mapping tool for Genetic circuits
Timing Analysis
Logic Analysis
Synthesis
Technology Mapping
Genetic Gates Library

Figure 1.1: The abstract diagram showing how the contributions of this dissertation can be used.

1.6 Thesis Organization

This dissertation is organized as follows. **Chapter 2** gives the information about genetic circuits. This chapter gives some basic knowledge of genetic terminologies and a brief overview of how genetic systems work. It describes *standards* in more detail and also give information about existing GDA tools.

Chapter 3 gives a brief overview of D-VASim. It briefly describes the whole simulation flow with the help of an example circuit model. The details of each subsequent step in this flow is discussed in separate chapters.

In **chapter 4**, the methodology of timing analysis of genetic logic circuits is presented. This chapter discusses the algorithm developed for analyzing the threshold value and propagation delay of a genetic circuit model. The experimental results are included to support the significance of timing analysis in genetic logic circuits.

Chapter 5 explains the methodology developed to analyze and verify the logical behavior, of a genetic circuit, from the stochastic simulation data. This chapter also contains the experimental results of logic analysis on different genetic circuit models and the performance evaluation of the algorithm.

The approach for synthesis and technology mapping of genetic circuits is provided in **Chapter 6**. This chapter begins with describing the algorithms developed for reducing the Boolean expression of genetic gates into an optimized form, followed by its synthesis into NOR-NOT form. Then the methodology of technology mapping, along with the discussion of how the genetic gates library is constructed from the data disclosed in [14], is presented. In the end, some experimental results on case study have been presented.

In **Chapter 7**, it has been demonstrated how D-VASim can be used in collaboration with other tools to perform useful tasks. First an experimental flow is proposed for the statistical model checking of genetic circuits using Uppaal [15] and D-VASim [16]. The experimentation on genetic circuit models are performed to explore their design parameter sensitivity using Uppaal SMC [18]. Next, an attempt is made to show that how Cello [14] parameters can be used to perform simulation.

Chapter 8 concludes this research work with the discussion of possible future directions.

Appendix A contains the supplementary data of Chapter 3, which includes the sample ODE and stochastic simulation results produced by D-VASim.

Appendix B contains the supplementary data of Chapter 4. It consists of the timing analysis results for all the genetic circuit models being experimented.

The complete experimental data for the logic analysis (Chapter 5) is enclosed in **Appendix C**.

The extended experimental data related to *Gene***Tech** (Chapter 6) is given in **Appendix D**

D-VASim Quick Start Guide (QSG) is included in the **Appendix E**.

CHAPTER 2

Genetic Circuits

A biological system is composed of living organisms which consists of one or more living cells. The behavior of each of these cells is controlled by genetic circuits which perform dedicated tasks to achieve the overall functionality of a biological system. These genetic circuits, which are composed of several biological components (called the genes network) regulate the amount of proteins in a cell. This gene-regulated network is triggered by external signals, for example, light, temperature, presence of specific proteins, etc., to control the behavior of a living cell, effectively exhibiting a Boolean logic function. The aim of this chapter is to briefly introduce genetic circuits to the audience not familiar with synthetic biology. Section 2.1 gives a brief overview of biology and some basic terminologies, frequently used in genetic design, which are necessary to be known specially to the computer scientists or engineers who do not have primary knowledge of synthetic biology. Next, Section 2.2 presents an example of regulated transcription in lac operon and explains its genetic logic. Section 2.3 gives more information on the *standards* and Section 2.4 gives a brief overview of GDA tools.

2.1 Central Dogma of Molecular Biology

The *Living Cell* is the smallest biological unit of any living organism, and is often called the *building block of life*. Each cell is composed of several organelles like mitochondria, ribosomes, nucleus etc. The *nucleus* is the largest cellular component which contains part or all of the cell's genetic information. This genetic information is stored in the *deoxyribonucleic acid* (DNA) molecule, which is packaged into a thread-like structure called *chromosomes*. DNA is further divided into a group of nucleotide sequences called *genes*. Figure 2.1 shows the relationship between the eukaroytic cell's nucleus, chromosomes in the nucleus, and genes.

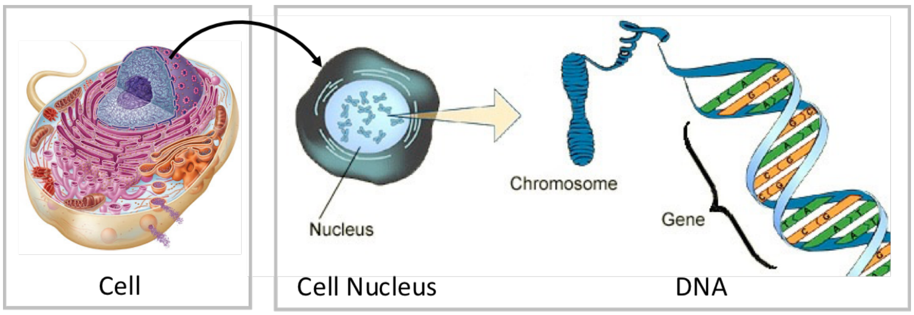


Figure 2.1: The hierarchical relationship of living cell and gene. (Image courtesy of BBC Science[1])

DNA is composed of two nucleotides strands coiled around each other to form a double-helix structure. Each of these strands contain a sequence of four nucleobases - *cytosine (C)*, *guanine (G)*, *adenine (A)* and *thymine (T)*. These bases on both of the strands bind to each other in pairs such that A only binds with T and G only binds with C. The sequence of these base-pairs codes for various genetic components including, promoters, operators, genes, etc.

Each gene is a region of DNA which generates a specific protein through the processes called *transcription* followed by *translation*. During transcription, the particular region of DNA (gene) is converted to *ribonucleic acids* (RNAs) by another RNA molecule called *RNA polymerase* (RNAP), which binds to a specific region of that gene called the *promoter*. RNA polymerase then moves along the gene's coding sequence and temporarily breaks the bond between the two DNA strands causing it to unwind or unzip. During this unwinding process, the RNA transcript is generated as shown in Figure 2.2. This process of RNAP

[1]`http://www.bbc.co.uk/schools/gcsebitesize/science/edexcel/classification_inheritance/genesandinheritancerev1.shtml`

generation continues until the moving RNAP reaches a region of DNA called *terminator.* At this instant, RNAP leaves DNA and the newly formed RNA is released. Many of these RNAs holds the instructions for constructing protein, and are commonly termed as *messenger RNAs* (mRNAs).

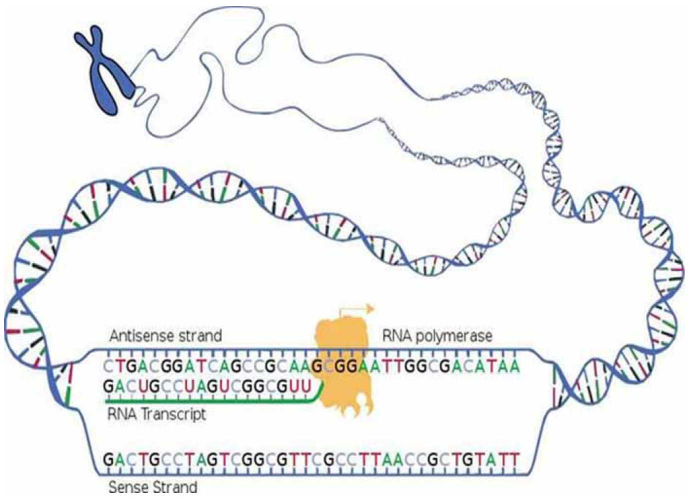


Figure 2.2: The process of transcription. (Image courtesy of the National Human Genome Research Institute)

Now, during the process of translation, another protein, *the ribosome*, binds to mRNA at its specific region, called the *ribosome binding site.* The ribosome moves along mRNA and generate the specific proteins. This process of converting DNA into mRNA through transcription and then the conversion of mRNA into protein through translation is known as the *central dogma of molecular biology* [19].

There are two types of gene expressions, *constitutive* and *regulated.* A Gene is expressed constantly in constitutive type of gene expression, whereas it is controlled and dependent on the environmental changes in regulated gene expression. The genetic circuits are based on the genes which are transcribed through regulation. This transcriptional regulation is carried out by regulatory proteins, called *transcriptional factors*, which binds to an *operator site*, a region of DNA near promoter. The transcription factor either block (referred to as *repressor*) or help (referred to as *activator*) RNAP to bind to the promoter region to initiate a process of transcription.

2.2 Example Genetic Circuit: Lac Operon

One of the classical systems used to investigate the transcriptional regulation of *Lac Operon*, was presented by Jacob *et al.* in [20]. *Operon* is referred to as a region of DNA which consists of a group of genes controlled by a single promoter. Lac operon (or lactose operon) is required for the transport and metabolism of lactose in the bacterium *Escherichia coli*, and it was the first gene-regulatory network to be explored clearly.

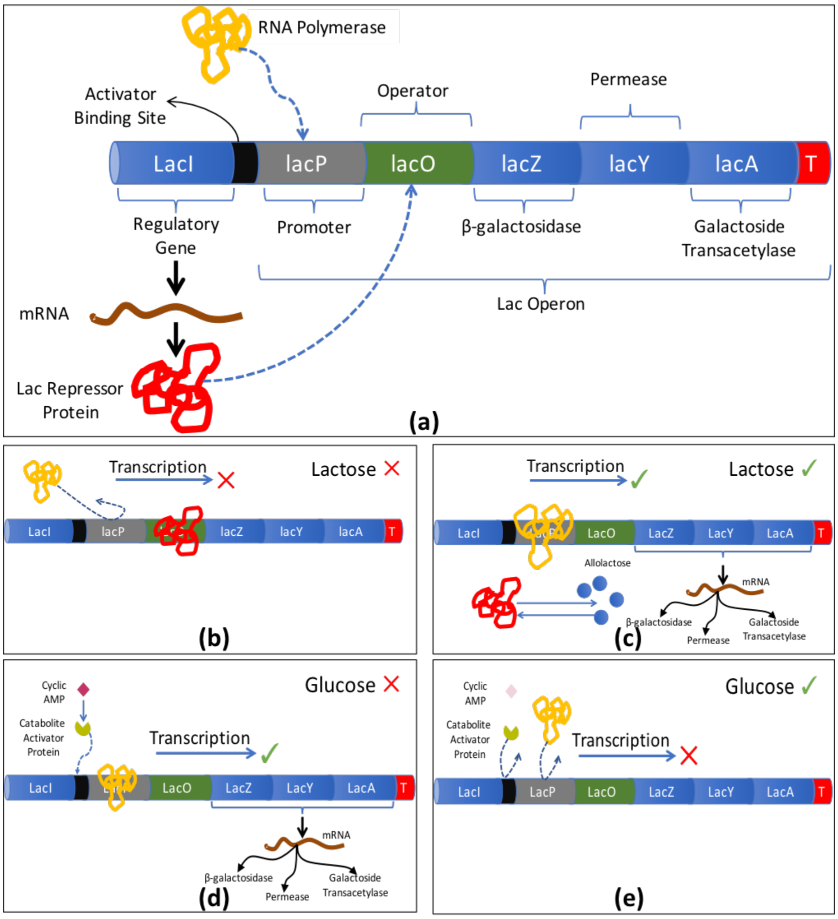


Figure 2.3: Transcriptional regulation of lac operon. (a) Structure of Lac operon. No transcription when (b) lactose is absent and (e) glucose is present. Transcription begins when (c) lactose is present and (d) glucose is absent.

Figure 2.3(a) shows the structure of the lac operon. Three genes, *lacZ, lacY,* and *lacA* are required, as a cluster, to utilize lactose by the bacterium. The lacZ, lacY and lacA genes code for the enzymes *beta-galactosidase*, *lactose permease* and *galactoside transacetylase*, respectively. The *lacP* is the promoter region which transcribes the lacZ, lacY and lacA genes as a single *polycistronic* mRNA. The *lacO* region is an operator site to which a transcription factor binds to regulate gene expression. The complete unit consisting of the lac promoter (lacP), lac operator (lacO), and the three genes (lacZ, lacY and lacA) is known as the *lac operon.* The *lacI* is the regulatory gene of lac operon that codes for an mRNA that is translated to produce a protein known as *lac repressor.* The "T" (shown as red region) corresponds to the terminator region where the RNAP stops transcription. The black region between lacI gene and the lac promoter is the *activator binding site* (ABS), which helps RNA polymerase (shown as yellow structure) to bind to the promoter site.

As shown in Figure 2.3(b), when the lactose is not available inside the cell, the lac repressor recognizes the operator site and binds to it tightly. This prohibits the RNA polymerase to recognize the promoter region, and thus prevents the operon to be transcribed. When lactose enters the cell, a small amount of it is converted to *allolactose*, which binds to the lac repressor. This causes a structural change in the lac repressor protein that prevents it from binding to the lac operator site. When lac repressor is not bound to an operator site, RNAP easily binds to a promoter region and transcribe the polycistronic mRNA. This mRNA is then translated to produce beta-galactosidase, lactose permease, and galactoside transacetylase proteins, as shown in Figure 2.3(c).

The discussion of Figure 2.3(b) and (c) indicates that the lac operon is transcribed when lactose is present inside the cell. However, the binding of RNAP, to a promoter site, weakly depends on the presence of lactose, and strongly on the presence of the *catabolite activator protein* (CAP) inside the cell. CAP attaches to an *ABS* and helps RNAP to bind to the promoter region to drive high levels of transcription. The CAP cannot directly bind to an ABS, rather it is regulated by a small molecule known as *cyclic AMP* (cAMP), which acts as a "hunger signal" when the glucose levels inside the cell are low. Therefore, when the glucose level is low, the cAMP bind to CAP and make it able to attach to the ABS. When CAP attaches to the activation binding site, it helps RNAP to bind to the promoter region strongly, and begin transcription. This process is shown in Figure 2.3(d).

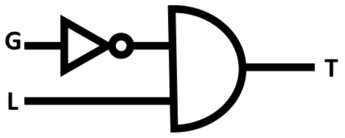
On the contrary, when the glucose level rises, it reduces the concentration of cAMP, which in turn makes the CAP unable to attach to the ABS. Without CAP being attached to the ABS, RNAP cannot attach to the promoter region and thus the transcription process is stopped, as shown in Figure 2.3(e).

2.2.1 Genetic logic in lac operon

From the discussion above, the natural genetic logic exist in the transcriptional behavior of lac operon can be extracted. Figure 2.4(a) summarizes the logical behavior of lac operon in the form of truth table, with inputs being *Glucose* (G) and *Lactose* (L), and the output being *Transcription* (T) of lacZ, lacY and lacA genes.

Inputs		Output
G	L	T
0	0	0
0	1	1
1	0	0
1	1	0

(a)



(b)

Figure 2.4: Genetic logic in lac operon. (a) Truth Table. (b) Circuit schematic.

When glucose is absent (logic 0), CAP binds to ABS and RNAP should perform transcription. However, when the lactose is also absent (logic 0), the *lacI repressor* protein binds to the operator region and blocks RNAP to move along the DNA strand to perform transcription. Therefore, transcription is always blocked whenever lactose is absent in the cell. Similarly, when glucose is present in the cell, it prohibits CAP to bind to ABS and thus reduce the affinity of RNAP to bind to the promoter region to begin transcription. However, when glucose is absent and lactose is present, the lac operon is transcribed, resulting in the Boolean logic shown as a circuit diagram in Figure 2.4(b).

2.2.2 The standard SBOL representation of lac operon

Figure 2.4(b) gives a fairly low-level details of how lac operon works, which is not the standard way of representing any genetic system. As mentioned earlier in *Chapter 1 - Introduction*, the *SBOL* is the standard way to represent the high-level diagrams of genetic systems. The equivalent SBOL visual (or SBOLv) diagram of the genetic system for lac operon would be something similar to the diagram shown in Figure 2.5. This figure indicates that a single promoter, P_{lac}, is responsible for the transcription of three genes, lacZ, lacY and lacA. The presence of glucose represses and the presence of lactose activates the promoter, reflecting that a transcription is initiated when glucose is absent and lactose is present in the cell. The symbol "T" represents the terminator region of DNA where the transcription is stopped. As a result of transcription, an mRNA is produced and the *ribosomes* binds to this mRNA at the *ribosome binding site* (shown as semi-sphere in Figure 2.5), to carry out the production of output proteins *beta-galactosidase, permease* and *galactoside transacetylase*, from the genes lacZ, lacY and lacA, respectively.

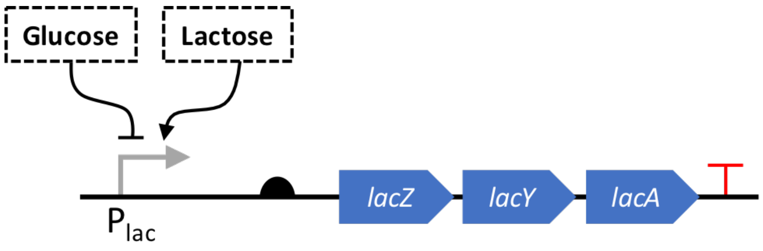


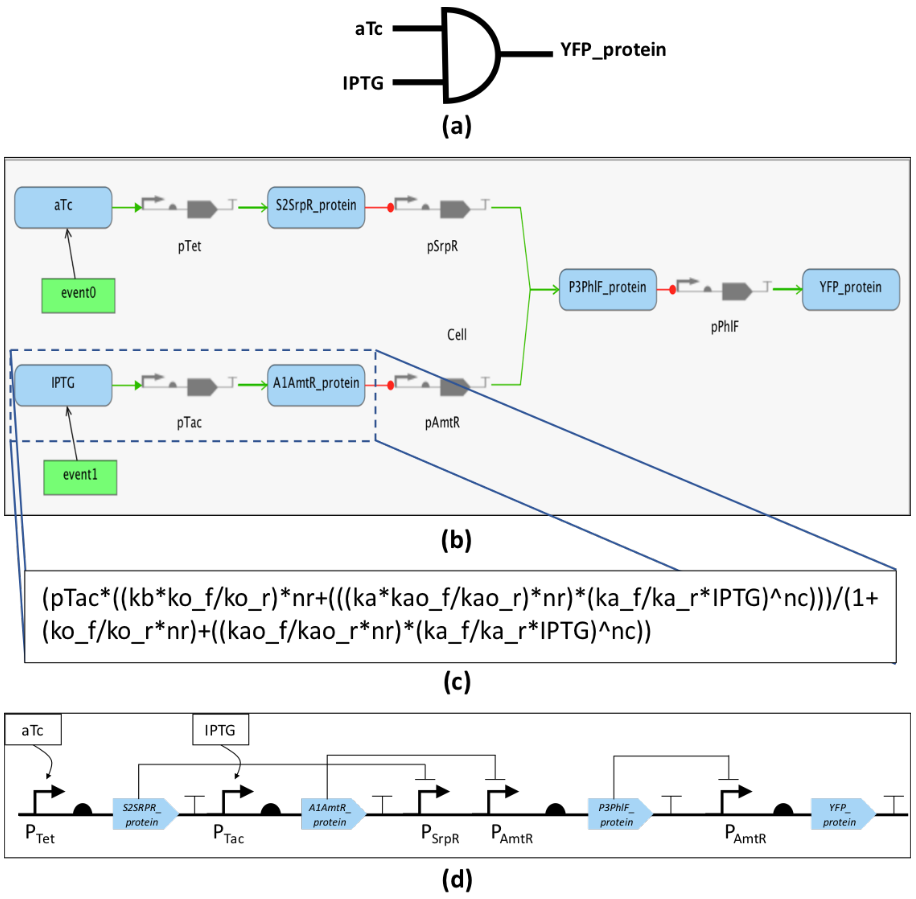
Figure 2.5: SBOL visual representation (or SBOLv) of lac operon genetic system.

The example discussed above is a natural genetic logic circuit which exists in the lac operon. Some researchers have already started to engineer custom genetic logic components to achieve a desired logical behavior in a living cell [14, 21]. We have used the SBML models of these genetic logic circuits [14, 21] to test the tools and method presented in this thesis.

2.3 Standards

It is mentioned in *Chapter 1 - Introduction* that SBML and SBOL are two major standards to represent genetic model's behavior and structure respectively. Both of these standards are developed by the members of systems and synthetic biology community.

The purpose of SBML standard is to exchange essential aspects of biological model among different softwares and is supported by over 290 GDA tools. It is a machine-readable *eXtensible Markup Language* (XML) that is independent of any specific software language. For example, the SBML model of a genetic AND gate circuit, designed in iBioSim [17], is shown in Figure 2.6(b). Tools which support SBML-synthesis allow users to define the model parameters, species and their biochemical interactions using mathematical expressions. As an example, the SBML model of a genetic AND gate circuit is shown in Figure 2.6(b) in which the reaction for *pTac*, along with an external influence of IPTG inducer, to produce *A1AmtR_protein* is shown in Figure 2.6(c). The values of the parameters in this figure, for example *kb*, *ko_r*, *nc*, etc, are defined separately. The different SBML-synthesis tools may have their own icons to represent standard biological processes like *repression*, *activation* etc. For example, in Figure 2.6(b), *A1AmtR_protein* repressing the following promoter *pAmtR* is shown with the red line having round-headed circle. The same processes are represented differently in different tools.

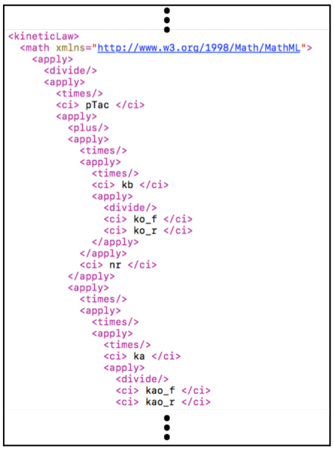


(pTac*((kb*ko_f/ko_r)*nr+(((ka*kao_f/kao_r)*nr)*(ka_f/ka_r*IPTG)^nc)))/(1+(ko_f/ko_r*nr)+((kao_f/kao_r*nr)*(ka_f/ka_r*IPTG)^nc))

Figure 2.6: SBML and SBOLv diagram of genetic AND gate. (a) Circuit schematic. (b) SBML model design in iBioSim [17]. (c) Example kinetic reaction. (d) SBOLv diagram.

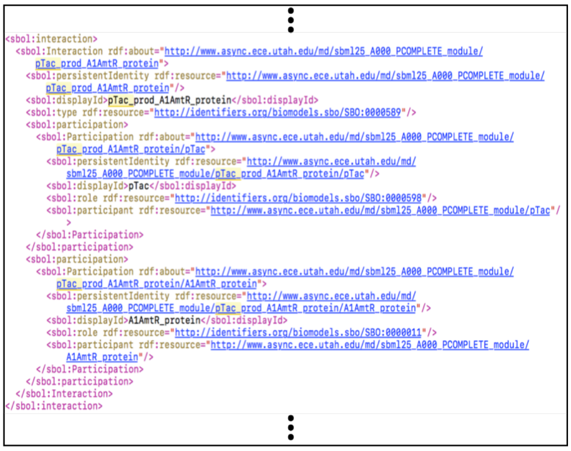
In order to keep the uniformity in representing these models, the SBOL is developed to document all biological models in a standardized manner. It is an emerging data standard for synthetic biology with growing support among several GDA software tools, including biochemical modeling tools [22–24], design composition tools [22,23,25–27], and sequence editing tools [28,29]. It is also an *eXtensible* standard so it can easily adapt the evolving needs of the synthetic biology community. Figure 2.6(d) shows the SBOLv diagram of the genetic AND gate model shown in Figure 2.6(b).

As said before that different tools may have their own icons to represent the biological processes and species, but all of them are supposed to generate the same *XML* document in order to be used by other software tools. Figure 2.7(a) and (b) shows the cropped images of SBML and SBOL *xml* files, respectively, of the same genetic AND gate model shown in Figure 2.6. Both of the images shown in this figure depicts some portion of these SBML and SBOL xml files, showing how the reaction (between *pTac* and *A1AmtR_protein*) shown in Figure 2.6(c) are represented.



```
<kineticLaw>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
      <divide/>
      <apply>
        <times/>
        <ci> pTac </ci>
        <apply>
          <plus/>
          <apply>
            <times/>
            <apply>
              <times/>
              <ci> kb </ci>
              <apply>
                <divide/>
                <ci> ko_f </ci>
                <ci> ko_r </ci>
              </apply>
            </apply>
            <ci> nr </ci>
          </apply>
          <apply>
            <times/>
            <apply>
              <times/>
              <apply>
                <times/>
                <ci> ka </ci>
                <apply>
                  <divide/>
                  <ci> kao_f </ci>
                  <ci> kao_r </ci>
```

(a)



```
<sbol:interaction>
  <sbol:Interaction rdf:about="http://www.async.ece.utah.edu/md/sbml25_A000_PCOMPLETE_module/
      pTac_prod_A1AmtR_protein">
    <sbol:persistentIdentity rdf:resource="http://www.async.ece.utah.edu/md/sbml25_A000_PCOMPLETE_module/
        pTac_prod_A1AmtR_protein"/>
    <sbol:displayId>pTac_prod_A1AmtR_protein</sbol:displayId>
    <sbol:type rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000589"/>
    <sbol:participation>
      <sbol:Participation rdf:about="http://www.async.ece.utah.edu/md/sbml25_A000_PCOMPLETE_module/
          pTac_prod_A1AmtR_protein/pTac">
        <sbol:persistentIdentity rdf:resource="http://www.async.ece.utah.edu/md/
            sbml25_A000_PCOMPLETE_module/pTac_prod_A1AmtR_protein/pTac"/>
        <sbol:displayId>pTac</sbol:displayId>
        <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000598"/>
        <sbol:participant rdf:resource="http://www.async.ece.utah.edu/md/sbml25_A000_PCOMPLETE_module/pTac"/
            >
      </sbol:Participation>
    </sbol:participation>
    <sbol:participation>
      <sbol:Participation rdf:about="http://www.async.ece.utah.edu/md/sbml25_A000_PCOMPLETE_module/
          pTac_prod_A1AmtR_protein/A1AmtR_protein">
        <sbol:persistentIdentity rdf:resource="http://www.async.ece.utah.edu/md/
            sbml25_A000_PCOMPLETE_module/pTac_prod_A1AmtR_protein/A1AmtR_protein"/>
        <sbol:displayId>A1AmtR_protein</sbol:displayId>
        <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000011"/>
        <sbol:participant rdf:resource="http://www.async.ece.utah.edu/md/sbml25_A000_PCOMPLETE_module/
            A1AmtR_protein"/>
      </sbol:Participation>
    </sbol:participation>
  </sbol:Interaction>
</sbol:interaction>
```

(b)

Figure 2.7: Cropped images of SBML and SBOL files of genetic AND gate circuit shown in Figure 2.6. (a) SBML file. (b) SBOL file.

2.4 Genetic Design Automation (GDA) Tools

Numerous computational tools [13,23,30,31] have been developed to assist users in designing genetic circuits through an automated processes. These tools includes DNA sequence editing, biochemical modeling, design composition, and technology mapping tools. According to [13], there are more than 290 tools which support SBML model construction and simulation; and about 30 of them are GDA tools which support sequence editing, design composition, optimization and technology mapping. Some of these tools serve as a toolbox for commercial platforms including MATLAB, Mathematica, and Oracle; some are developed as APIs or plugins to specific software systems, while others are independent tools for design and simulation.

2.4.1 Sequence editing tools

Sequence editing tools are typically considered low-level tools which enable user to construct, edit or annotate the base-pair sequences of genetic components. These tools include SBOLDesigner [32], Synthetic GeneDesigner [28], GeneDesign [33], VectorEditor [29], and Kera [34]. Only few of the sequence editing tools support the SBOL standard. However, nearly all of these tools support reading and writing plain text DNA sequences that comply with the *International Union of Pure and Applied Chemistry* (IUPAC) codes for *nucleotides* [35] and *amino acids.*

2.4.2 Biochemical modeling and design composition tools

These tools allow users to develop mathematical models and usually also provide users with the ability to analyze these models. They require users to manually compose designs and generate standard files automatically. Some of these tools are CellDesigner [36], BioUML [37], iBioSim [17], COPASI [38], D-VASim [16, 39], Uppaal [15], Asmparts [40], GEC [41], GenoCAD [26], Kera [34], ProMoT [42], Antimony [43], Proto [23], SynBioSS [44], TinkerCell [22], and Cello [14]. Few of these tools also support model checking of genetic circuits including [15,17,18,45,46].

A vast majority of these tools provides different level of support of reading and/or writing SBML files to capture the mathematical behaviour of biological models. For example, CellDesigner, iBioSim, COPASI, and ProMoT support both import and export of SBML files; Asmparts, GenoCAD, GEC and SynBioSS, on the other hand, are only able to export SBML; while D-VASim, being a simulation tool, is able to import SBML files only. Each of these tools have their own advantages. For example, iBioSim, Antimony and BioUML tools support hierarchical model composition in a standardized SBML format. iBioSim also supports designing and modifying the DNA sequences of genetic constructs using SBOL data model [32]. Most of these tools support both deterministic and stochastic simulations, including iBioSim, COPASI and D-VASim. A noticeable feature of SynBioSS is that it supports running the complex models on a *supercomputer* to speedup the process of finding genetic parts to construct a desired biological system. An apparent feature of TinkerCell tool is that it provides a platform to integrate third-party algorithms for testing different methods relevant to synthetic biology. The distinctive feature of D-VASim is its virtual simulation environment to let the users perform experimentation by dynamically interacting with the biological models. It also supports *automated* timing and logic analysis of genetic circuits in its virtual simulation platform.

Some of the modeling tools support programming languages and allows the user to design biological circuits using different programming formats. For example, *Proto* converts a high-level program specifications into gene-regulatory networks, optimizes it, and then validate its behavior through simulation. *Kera* is a *C*-like object-oriented programming language which supports a biopart rule library called *Samhita* which is a database containing part IDs, their types and sequences. *GenoCAD*, a web based tool, also supports some features of programming language and a built-in database of BioBricks. It allows user to perform model simulation using *mass action kinetics*. *Antimony* is a module-based programming language which provides a special syntax to create modular genetic networks. The additional library, *libAntimony*, in this tool allows other software packages to import models and convert them to SBML. Unlike GenoCAD, which uses rates of mass action for simulation, Antimony uses the rates of gene expression, for example, *polymerase per second* (PoPS) and *ribosomes per second* (RiPS) for simulations. Since it is difficult to connect genetic gates because of different input and output proteins, the *rate of gene expression* is often used as the integration signal to solve this problem [21]. The rate of gene expression, which is similar to the rate of flow of electron (electric current), is the rate at which the RNA polymerases move across a DNA strand.

GEC, developed by Microsoft corporation, is a rule-based programming language which also allows user to express the logical interaction of biological components in a modular manner. The GEC program is translated into multiple possible genetic devices (RBS, promoters etc). The GEC model is refined by simulating these possible devices iteratively and ruling out those with undesired simulation results. A recently published web-based tool, named *Cello*, allows user to develop genetic circuits using *verilog* (`http://www.verilog.com/`) - a hardware description language used for designing electronic circuits. The circuits are developed using the genetic gates which have already been tested in the laboratory.

2.4.3 Genetic mapping tools

Genetic technology mapping tools automatically select genetic components from library and integrate them together to achieve the desired functionality of a genetic circuit. The mapping techniques, similar to those adapted in EDA and software engineering, are mostly used to perform technology mapping of genetic circuits. However, technology mapping of genetic circuit is much more computationally intensive as compared to the technology mapping of electronic circuits. It is because the genetic components have different input and output signals as opposed to electronic components which have the same physical quantity both at input and output i.e., the voltage. Due to this, it is challenging to search and connect the right components together which should not only be compatible with each other, but also avoid unwanted *cross-talk* to achieve the desired functionality.

Some of the tools which supports genetic technology mapping are BioJADE [47], GEC [41], MatchMaker [48], SBROME [49], iBioSim [50], GeneTech [51] etc. BioJADE and GEC use *exact methods* to find the optimal solutions. On other hand, both MatchMaker and SBROME, use *heuristic methods* to find all possible solutions quickly and then rank them by quality. iBioSim uses a similar *Directed Acyclic Graph* (DAG) based approach for technology mapping which is used in EDA [52]. It generates the DAG first and then perform matching and covering to obtain the optimal solution. This tool is based on two assumptions; first, the circuit generated from the library do not have feedback; and second, the circuit components can be connected together if their molecular signals at input and output are the same. However, GeneTech generates all possible solutions using *depth-first search* approach. It starts off mapping the components at input stage first and then search for the right components (from the gates library) that can be connected in succession as a second stage, and so on. Unlike the first assumption of iBioSim, GeneTech avoid using such components which makes unintended feedback loops.

Chapter 3

Dynamic Virtual Analyzer and Simulator (*D-VASim*)

Simulation and behavioral analysis of genetic circuits is a standard approach of functional verification prior to their physical implementation. Many software tools have been developed to perform in-silico analysis for this purpose, but none of them allow users to interact with the model during runtime. The runtime interaction gives the user a feeling of being in the lab performing a real-world experiment. In this chapter, a *virtual laboratory* software tool named D-VASim (Dynamic Virtual Analyzer and Simulator) is presented, which is developed as a part of this thesis work. This tool provides a user-friendly environment to simulate and analyze the behavior of genetic logic circuit models represented in an SBML format. Hence, SBML models developed in other software environments can be analyzed and simulated in D-VASim. D-VASim offers deterministic as well as stochastic simulation; and differs from other software tools by being able to extract and validate the Boolean logic from the SBML model. D-VASim is also capable of analyzing the threshold value and propagation delay of a genetic circuit model.

The work presented in this chapter has been published in the following peer-reviewed workshop and journal

[16] Hasan Baig and Jan Madsen, "D-VASim – An Interactive Virtual Labora-

tory Environment for the Simulation and Analysis of Genetic Circuits", *Bioinformatics*, vol. 33, issue 2, pp. 297–299, 2016.

[53] Hasan Baig and Jan Madsen, "D-VASim: Dynamic Virtual Analyzer and Simulator for Genetic Circuits", *7th International Workshop on Bio-Design Automation (IWBDA)*, pp. 48–49, 2015.

3.1 Motivation

In the wet lab, biologists are either provided with the ready-made biological model available in a test tube or are given a specification/recipe from which to prepare the model in the lab. Their duty is to analyse the model and verify its functional behavior. This analysis is done interactively, among other things, by increasing the molecular concentration of input species at any instant of time and observing the effects. This process motivated us to develop D-VASim, a virtual laboratory environment where a user can perform interactive experiments by varying the molar concentration of external signals during runtime.

The SBML and Cell Mark-up Language (CellML) [54] are the two standard methods of representing biological models in a machine-readable form, which enable models to be shared and published in a way that can be used by different software tools. SBML is supported by most relevant tools for synthetic biology. An SBML file holds the model information including species, reaction parameters, kinetic laws, initial concentrations etc. Beside these modular descriptions of a bio model, SBML also allows a user to model a sequence of input patterns in order to capture more behavioral details. This is done through events, which describe the instantaneous, discontinuous state changes in the model [9]. For example, in genetic circuits, events are used to trigger the concentration of any input species to a certain level, at a specific point in time, and to observe the effects on the concentration of output species. Since events are predefined and cannot be changed during runtime, the output of a genetic logic circuit will be analyzed only for defined events. In order to observe the complete behavior of an output of a genetic logic circuit, a different set of input conditions, i.e., when to change what input to which level, must be defined in each event. Even for moderate sized genetic logic circuits, capturing all the combinations of inputs and concentration levels may require a very large number of events to be defined and simulated.

On the other hand, a runtime interaction capability with the model is more suitable to make direct changes in the concentration of input species at any instant of time to observe the model's behavior. This not only helps the user to analyze the model easily by triggering the concentration of input species to any level and at any instant of time, but also makes a user free of defining a long list of events for all the possible combinations of inputs in the SBML description. Furthermore, the interactive simulation enables a user to receive feedback in parallel with their experimental intervention, which enables certain types of learning and optimization that would not be possible otherwise.

Besides giving a biologist the feeling of being in the lab, D-VASim may also be useful to help early-stage researchers and students, with less experience of biology, to get an intuitive feeling of the underlying biological processes and their interactions. D-VASim can play a vital role for educating inexperienced users to observe the live biological phenomenon in a virtual laboratory environment without being afraid of overreaction and mishandling of the apparatus.

3.2 Methodology and Experimental Approach

D-VASim is developed on the LabVIEW (Laboratory Virtual Instrument Engineering Workbench) programming platform, which is a graphical programming language commonly used for rapid development of instrumentation systems for data acquisition, instrument control, and industrial automation (`www.ni.com`). The basic flow of the proposed virtual simulation and analysis environment is shown in Figure 3.1.

Figure 3.1 shows that the SBML model of a genetic circuit is first loaded in D-VASim, and the components of an SBML model can be optionally analyzed in a user-friendly manner. Then a user can generate a separate virtual instrument (VI) to perform ODE and stochastic simulations. These VIs help a user to observe the reactions graphically and interact with the model at run-time. This process is equivalent to setting up an apparatus for testing and experimentation of a model in the wet-lab.

For stochastic simulations, the timing analysis of a genetic circuit can also be performed if they are not known. The timing analysis helps a user to analyse the threshold value and the propagation delay of a genetic circuit model. If a user already know the threshold value and propagation delay of a circuit, the genetic circuit model can be analyzed by interacting with the model during run-time. This run-time interaction allows a user to change the concentration of input species and the parameter values at any instant of time and observe the change in circuit's behavior alongside.

Once all the possible input combinations are applied carefully, a user can ana-

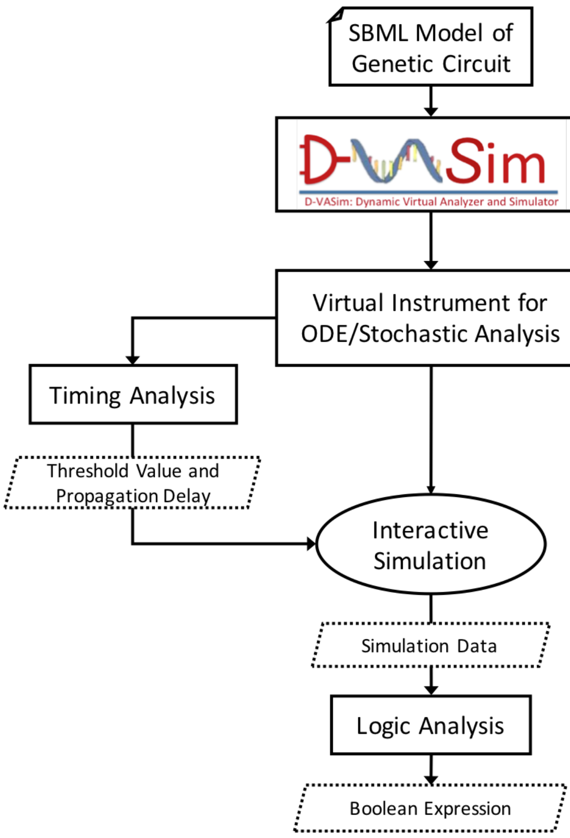
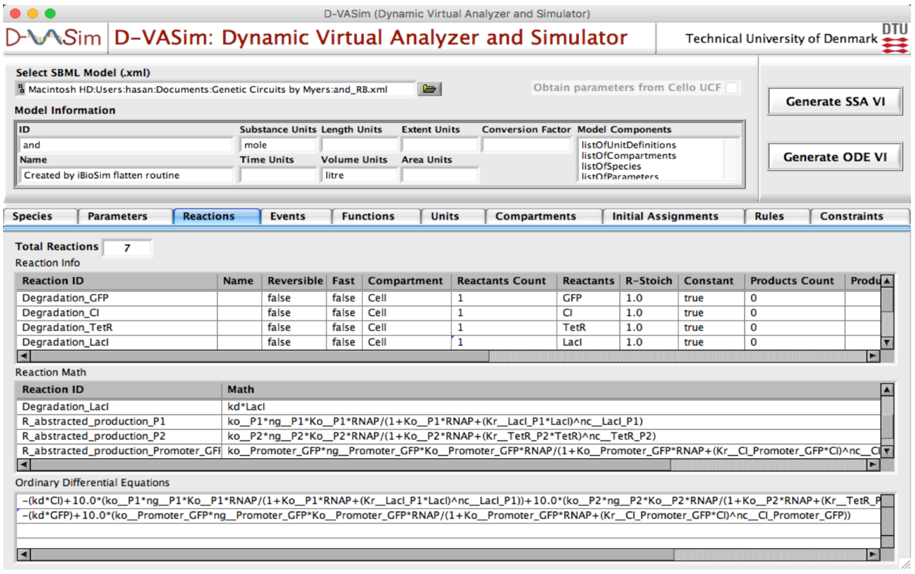

SBML Model of Genetic Circuit
D-VASim
D-VASim: Dynamic Virtual Analyzer and Simulator
Analyze SBML model
Virtual Instrument for ODE/Stochastic Analysis
Timing Analysis
Threshold Value and Propagation Delay
Interactive Simulation
Simulation Data
Logic Analysis
Boolean Expression

Figure 3.1: A work flow diagram of D-VASim showing how it can be used for simulation, analysis and verification of genetic logic circuit models. Elliptical nodes represent the steps to be performed by a user; rectangular nodes represent the automated processes, and the dotted-parallelogram shows the output from the previous stage.

lyze/verify the logical behavior of either a complete circuit, or any intermediate components of a circuit. The results of this analysis comes out in the form of a *boolean expression*. The virtual instrument generated by D-VASim, for stochastic and ODE both, also logs the simulation data for analysis and retrieval of the user-session at a later stage.

3.2.1 SBML support

D-VASim supports the latest SBML format level 3, version 1 (l3v1) core[1]. It processes the SBML file using the JSBML library [55], and extracts and presents the information of all components in a tabular format, as shown in Figure 3.2. This figure shows the first interface which a user see after initializing and loading the SBML model in D-VASim. Each *tab* describes the corresponding SBML component in a user-friendly manner. For example, the *selected tab* shown in Figure 3.2 depicts the information of *Reactions* defined in the SBML file of the genetic AND gate model [21]. It not only shows the kinetic reactions, but also the ordinary differential equations, generated by D-VASim, to simulate the deterministic behavior of this model.


D-VASim (Dynamic Virtual Analyzer and Simulator)
D-VASim
D–VASim: Dynamic Virtual Analyzer and Simulator
Technical University of Denmark
DTU
Generate SSA VI
Generate ODE VI
Obtain parameters from Cello UCF
Species
Parameters
Reactions
Events
Functions
Units
Compartments
Initial Assignments
Rules
Constraints

Select SBML Model (.xml)

Macintosh HD:Users:hasan:Documents:Genetic Circuits by Myers:and_RB.xml

Model Information

ID	Substance Units	Length Units	Extent Units	Conversion Factor	Model Components
and	mole				listOfUnitDefinitions listOfCompartments listOfSpecies listOfParameters
Name	**Time Units**	**Volume Units**	**Area Units**		
Created by iBioSim flatten routine		litre			

Total Reactions 7

Reaction Info

Reaction ID	Name	Reversible	Fast	Compartment	Reactants Count	Reactants	R-Stoich	Constant	Products Count	Produ
Degradation_GFP		false	false	Cell	1	GFP	1.0	true	0	
Degradation_CI		false	false	Cell	1	CI	1.0	true	0	
Degradation_TetR		false	false	Cell	1	TetR	1.0	true	0	
Degradation_LacI		false	false	Cell	1	LacI	1.0	true	0	

Reaction Math

Reaction ID	Math
Degradation_LacI	kd*LacI
R_abstracted_production_P1	ko__P1*ng__P1*Ko__P1*RNAP/(1+Ko__P1*RNAP+(Kr__LacI_P1*LacI)^nc__LacI_P1)
R_abstracted_production_P2	ko__P2*ng__P2*Ko__P2*RNAP/(1+Ko__P2*RNAP+(Kr__TetR_P2*TetR)^nc__TetR_P2)
R_abstracted_production_Promoter_GFI	ko__Promoter_GFP*ng__Promoter_GFP*Ko__Promoter_GFP*RNAP/(1+Ko__Promoter_GFP*RNAP+(Kr__CI_Promoter_GFP*CI)^nc__CI

Ordinary Differential Equations

```
-(kd*CI)+10.0*(ko__P1*ng__P1*Ko__P1*RNAP/(1+Ko__P1*RNAP+(Kr__LacI_P1*LacI)^nc__LacI_P1))+10.0*(ko__P2*ng__P2*Ko__P2*RNAP/(1+Ko__P2*RNAP+(Kr__TetR_P
-(kd*GFP)+10.0*(ko__Promoter_GFP*ng__Promoter_GFP*Ko__Promoter_GFP*RNAP/(1+Ko__Promoter_GFP*RNAP+(Kr__CI_Promoter_GFP*CI)^nc__CI_Promoter_GFP))
```

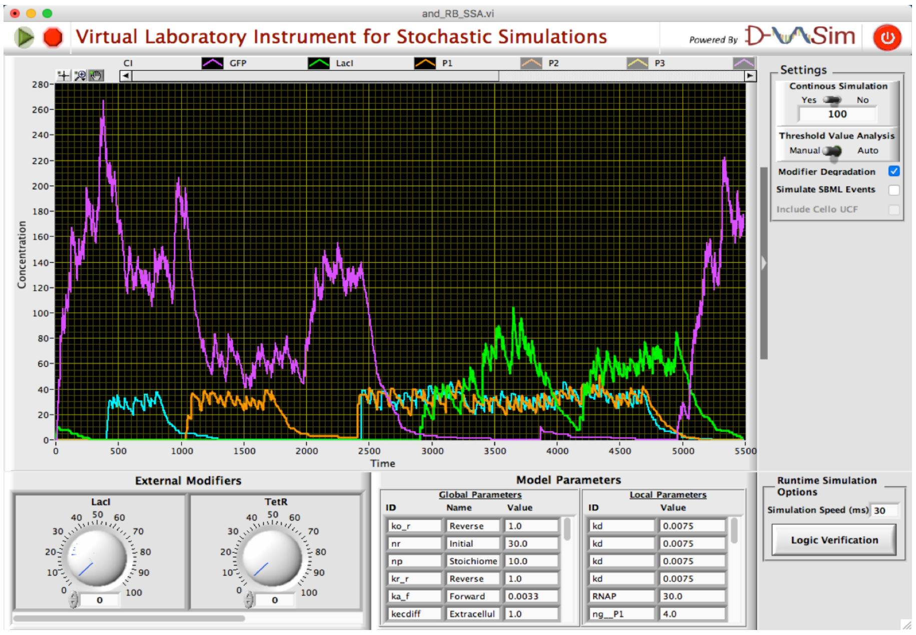
Figure 3.2: The main interface of D-VASim showing how the components of SBML file can be analyzed through a user-friendly interface.

3.2.2 Virtual instrumentation

Depending on the option selected, D-VASim generates a virtual instrument for the deterministic or stochastic analysis separately. Once the instrument is generated, the user can analyze the model by varying those species concentrations,

[1]D-VASim does not currently support SBML *packages*. The complete list of supported SBML components are declared in the *Read Me* file in the D-VASim download package.

which act as external modifiers or external inputs. For example, selecting the *Generate SSA VI option* (shown in Figure 3.2) generates the virtual instrument, shown in Figure 3.3, to perform the stochastic analysis of the genetic logic circuit described in the SBML file.



Settings

Continous Simulation: Yes / No — 100

Threshold Value Analysis: Manual / Auto

Modifier Degradation ✓

Simulate SBML Events ☐

Include Cello UCF ☐

External Modifiers: LacI 0; TetR 0

Model Parameters

Global Parameters

ID	Name	Value
ko_r	Reverse	1.0
nr	Initial	30.0
np	Stoichiome	10.0
kr_r	Reverse	1.0
ka_f	Forward	0.0033
kecdiff	Extracellul	1.0

Local Parameters

ID	Value
kd	0.0075
kd	0.0075
kd	0.0075
kd	0.0075
RNAP	30.0
ng__P1	4.0

Runtime Simulation Options

Simulation Speed (ms) 30

Logic Verification

Figure 3.3: Virtual instrument showing the stochastic simulation traces of the genetic AND gate (shown in Figure 3.5) obtained from [21].

The SBML file contains the complete list of species involved in the circuit model as well as the species acting as the modifiers to the specific kinetic reaction. It does not, however, specify explicitly which species acts as external inputs to the entire circuit model. D-VASim identifies the external species first by arranging the names of all modifiers species and the products species of each reaction in two separate lists, and then search for the species common to both of them. The modifier species, which is present in the list of products, indicates that it is a product of an underlying reaction and, hence, is not applied externally. The species, which are present in the list of modifiers but are not in the list of any reaction products, are those acting as external inputs (or external modifiers) to the circuit.

The above process is explained in the example list of reactions shown in Figure 3.4. There are three species in this figure – M1, P2 and M2, which act as modifiers to the reactions 2, 3 and 4, respectively. The species P2 is modifying

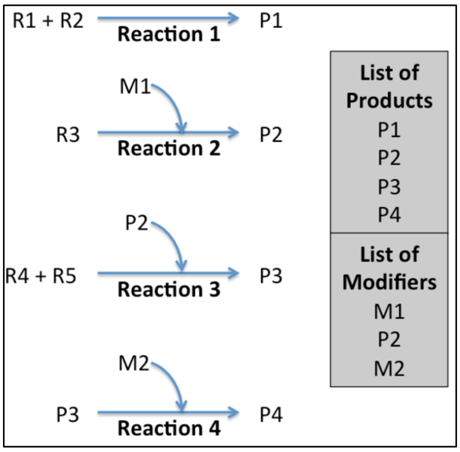


Figure 3.4: Example reactions explaining the identification process of the external modifiers.

the reaction 3, but it is the product of the underlying reaction 2; therefore, it cannot be considered as an external modifier. The rest of the species in the list of modifiers i.e. M1 and M2, are those that are acting as external modifiers because none of them are the products of any of the reactions, and thus considered as external inputs. When the external modifiers or inputs are identified, D-VASim creates the control knobs for them to let a user vary their concentration levels during the run-time simulation. It is also possible in D-VASim to create the control knobs for the specific species (see Appendix E for details).

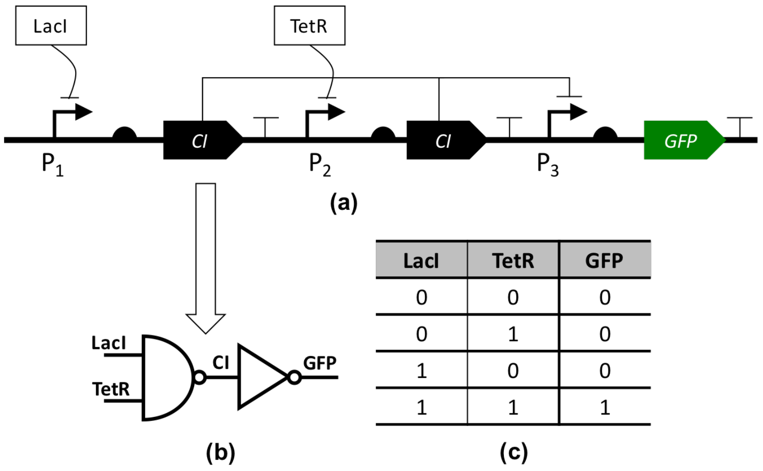
The virtual instrument for each model generated by D-VASim looks similar to a physical instrument, which serves as a standalone simulation tool for that specific model and can be used later without having its SBML file. It can be used to interact with the model by tuning the input concentration levels with the help of control knobs and observing the effects graphically. This run-time interaction with the model also helps users to trigger the concentration of input species to any level at any instant of time without defining the long list of events in an SBML file. In a similar way that input concentrations can be changed using control knobs, the parameters editor can be used to vary parameters, like degradation rate, temperature, etc., during run-time and observe their effects graphically. Furthermore, D-VASim also allows a user to simulate events defined in the SBML file.

Unlike wet-lab experimentation, a user may speed-up or slow-down the reactions (for stochastic simulation) with the help of numeric speed control displayed on each virtual instrument (see Figure 3.3). Moreover, when the simulation is stopped by a user, the simulation data and the screen shot of a graphical window are stored automatically in the default application folder.

3.2.3 Virtual experimentation

The Gillespie's direct stochastic simulation algorithm (SSA) [56] is implemented to capture the stochastic nature of the biological models described in the SBML file. For deterministic simulations[2], D-VASim supports ten different types of continuous solvers (see Appendix E).

Figure 3.5 shows the genetic AND gate circuit, constructed with the genetic NAND and NOT gates [21]. Figure 3.5(a) depicts the SBOLv diagram of genetic AND gate in which P_1, P_2 and P_3 corresponds to the promoter regions of DNA. Figure 3.5(b) and (c) shows the circuit schematic and the truth table of genetic AND gate respectively. In this example, when two proteins, *LacI* and *TetR*, are present in the significant amount within the cell, they inhibit promoters P_1 and P_2 to produce the output protein CI. When the concentration of CI falls below a certain level, promoter P_3 is activated and produces an output, i.e. green fluorescent protein (GFP).



LacI	TetR	GFP
0	0	0
0	1	0
1	0	0
1	1	1

Figure 3.5: Genetic AND gate circuit [21]. (a) SBOLv diagram. (b) Circuit schematic. (c) Truth table.

[2]Deterministic simulation is tested with the first 400 cases of SBML benchmark suite, and it produced the correct results for all of the supported components in D-VASim.

The interactive stochastic simulation traces of the genetic AND gate model, in D-VASim, is shown in Figure 3.3. D-VASim identified the external modifiers (or external inputs), *LacI* and *TetR*, and created control knobs to let a user control their concentration levels during run-time simulation. The simulation traces shown in Figure 3.3 indicates that the user varied the concentration levels of input species at different instants of time and in different combinations. The concentration of *CI* is suppressed, when the concentrations of both LacI and TetR are present in a significant amount in the cell at the same time (between 2500-4700 time units). When the concentration of *CI* falls below a certain level, the promoter P_3 is activated and the GFP is produced, thus exhibiting the AND logic in the cell, as shown in Figure 3.3.

3.2.4 Logic verification and timing analysis

Besides interactive simulation of any SBML-based model, D-VASim is also capable of verifying the logical behavior of a genetic circuit model by extracting the observed Boolean logic function from the simulation data. This functionality is useful in two ways – first, it allows a user to verify circuits, built by cascading several genetic logic circuits; secondly, it helps a user to extract the Boolean logic of a model even when a user does not have any prior knowledge about the model's expected behavior. The details of this functionality is presented in Chapter 5.

In order to obtain the correct Boolean logic, all possible input combinations must be applied in a significant amount to trigger the circuit's output. To determine which concentration level should be considered logic 0 and 1, D-VASim is able to obtain the threshold value and propagation delay of a circuit, which is described in details in Chapter 4.

3.3 Discussion

D-VASim is in the process of continuous development and has been continuously upgraded to higher versions. The latest version of D-VASim can be downloaded from http://bda.compute.dtu.dk/downloads/d-vasim/ and the video demonstration of the complete simulation flow can be seen at http://bda.compute.dtu.dk/user-manuals/d-vasim/.

Besides testing the D-VASim's ODE simulation environment with the SBML benchmark suite, the ODE and the stochastic simulations of D-VASim was also tested with the genetic circuit models presented in [21]. Appendix A contains the screen-shots of ODE and stochastic simulations results of the five genetic circuit models [21], taken in D-VASim.

The next Chapter 4 describes how the threshold value and the propagation delay of these genetic circuit models can be analysed. Such analyses can then be used to obtain the Boolean behavior (Chapter 5) of a genetic circuit model.

Chapter 4

Genetic Circuits Timing Analysis

Analogous to microelectronics, where timing analysis is a crucial requirement for ensuring the correct operation of a logic circuit, the timing analysis of genetic logic circuits may become an essential design characteristic as well. The transistors, used in the composition of digital logic gates, have well-defined threshold voltage values [57], which categorize the logic levels 0 and 1. Hence, the timing characteristics, like propagation delay, hold time, setup time etc., are all well characterized.

However, this is not the case in genetic logic gates, where each gate is composed of different proteins and promoters, resulting in different threshold concentration values. Furthermore, digital logic gates have the same physical quantity, i.e., voltage, as their input and output. On the contrary, genetic logic gates use different biological components including proteins, RNA, inducers, etc., to control the regulation of the corresponding output biological components. Additionally, signals in electronic circuits propagate in separate wires that do not directly interfere with each other. However, in genetic circuits, signals are molecules, drifting in the same volume of the cell, and hence easily merge with the concentration of other compounds, resulting in crosstalk with the neighboring circuit components. These facts make the timing analysis of genetic circuits very challenging.

Challenges of crosstalk have also been encountered in microelectronics; however, most of these have been solved through enhanced fabrication processes or through the development of advanced electronic design automation (EDA) tools. Similarly, advances in GDA tools may help to address these challenges, resulting in the reduction of the design complexity of genetic logic circuits.

In this chapter, a methodology is presented to perform the timing and threshold value analysis on genetic logic circuits. This methodology is implemented as a plug-in tool in D-VASim. We demonstrated that it is possible to perform the timing analysis of a genetic circuit and that it can be used to achieve the desired circuit behavior. The timing analysis is performed on some of the genetic circuit models proposed in [21] and [14], and the sensitivity of circuit timings, in relation of varying different circuit parameters, is investigated. In particular, the timing sensitivity due to the *degradation rate (kd)* and the concentration of input proteins is studied.

The work presented in this chapter has been published in the following peer-reviewed workshop and journal.

[39] Hasan Baig and Jan Madsen, "Simulation Approach for Timing Analysis of Genetic Logic Circuits", *ACS Synthetic Biology*, 2017.

[58] Hasan Baig and Jan Madsen, "Logic and Timing Analysis of Genetic Logic Circuits using D-VASim", *8th International Workshop on Bio Design Automation (IWBDA)*, pp. 77–78, 2016.

4.1 Methodology

As mentioned in Chapter 3 that the threshold value and timing analysis can be used to verify the Boolean function of a genetic logic circuit by extracting the observed logic behavior from the simulation's results. To analyse the Boolean logic, the genetic logic circuit model can be considered as a black box. Applying all possible input combinations and observing the output can result in the combinatorial behavior of this black box. For instance, if a circuit contains two inputs, then there are four possible input combinations; 00, 01, 10 and 11.

The key challenge in determining the correct Boolean logic function from the analog simulation data is to categorize the input concentration levels into logic 0 and logic 1. As mentioned earlier, this is similar to digital electronic circuits in which a certain threshold value of input voltage differentiates logic levels 0 and 1 [57]. Digital electronic circuits are also analog in nature, but a logical abstraction has been employed to reduce the complexity of circuits. Similar abstraction has to be employed to categorize the genetic concentration levels into logic 0 and 1. To categorize these concentration levels into logic 0 and 1, the threshold value for the concentration of input proteins, which significantly affects the concentration of output protein of a genetic logic circuit, must be identified.

As different proteins in a genetic circuit may have different threshold concentration values, the proposed approach calculates a single threshold value of the input proteins that trigger the output, instead of estimating the threshold values of each input protein separately. For instance, in the genetic AND gate (Figure 3.5), the algorithm estimates the threshold value of LacI and TetR, which together trigger the production of GFP, rather than evaluating the separate threshold values for each of them. It may be possible that the threshold value of LacI is, for example, 13 molecules, and that of TetR is, say, 9 molecules. In this case, algorithm tells that 13 molecules is the threshold value of an entire circuit, which triggers the circuit output when the concentrations of input proteins reach this level.

Consider another example of an OR gate in which input-1 triggers the output if the molecular count is greater than 5 and input-2 triggers the output if the molecular count is greater than 10. Setting the upper input threshold to 10 would give the correct answer, i.e. the gate remains off, if the input molecular counts are (4,7). Now, if the input molecular counts are (7,4), then input-1 may trigger the output but it may not be considered logic 1 until the output concentration increases above 10 molecules. It is observed, through simulations that the triggered output for such scenarios is highly unstable (frequently oscillating between logic 0 and logic 1), and this region should be considered a transition region. Therefore, instead of estimating the threshold values of each input protein separately, the proposed approach estimates the global upper and lower threshold values for all inputs.

Furthermore, the proposed approach considers the entire circuit as a black-box and obtains the input threshold value that is required to trigger the final output. Therefore, the threshold value and the number of intermediate circuit components do not matter; the algorithm ensures that the estimated input threshold value is sufficient to trigger the intermediate circuit components all the way from input to the final output. However, the separate threshold values of intermediate circuit components can also be analysed in D-VASim (for more details, see Appendix E or a video demo at http://bda.compute.dtu.dk/user-manuals/d-vasim/).

4.1.1 Preliminary analysis of threshold value

In order to understand the algorithm for estimating the threshold value, consider the simulation traces of the genetic AND gate using iBioSim [17] shown in Figure 4.1. It shows the results from running the stochastic simulation, of the genetic AND gate for one (a) and fifty times (b) and (c). The unit of species concentration used in the circuit models of [21] is the "number of molecules". Figure 4.1(a) shows that both of the inputs are triggered to 10 molecules, TetR after 1000 time units and LacI after 2000 time units, and that the output is highly stochastic, which makes it difficult to determine the input threshold value. A smooth output curve is obtained by plotting the average of 50 runs, as shown in Figure 4.1(b) and (c).

In Figure 4.1(b), it is seen that keeping the input concentrations to 10 molecules causes the average output concentration to stay below the level of the input concentration. Upon increasing the input concentrations further to 13 molecules, the average output concentration goes above the level of the input concentrations, as depicted in Figure 4.1(c). The same analyses are also performed with different concentration levels on different logic circuits. These analyses show a relation between the input and output proteins of a genetic circuit. On the basis of these analyses, an input-output relation of a genetic circuit can be defined in terms of its threshold value as follows:

Definition 4.1 Threshold value: The minimum concentration of input protein(s), which causes the average concentration of output protein to cross the concentration of input protein(s).

In the example shown in Figure 4.1, the upper threshold value of input is 13 molecules; that is, the input concentration above 13 molecules is considered logic 1 and that below 10 molecules is considered logic 0. There is a transition region between these two levels (not shown in Figure 4.1), where the average output concentration is not clearly distinguishable with the input concentration level. Hence, when the concentration levels of both inputs are 10 or fewer molecules i.e. logic 0; the average output concentration remains low (logic 0), else it goes high (logic 1) when the concentration of both inputs reaches 13 or more molecules (logic 1). This relation of input and output concentration is justified because, according to this definition, one do not need to care about how many circuit levels are cascaded between input and output. It simply identifies the input concentration required to trigger the final output. The same definition is applicable to determine the threshold values of intermediate circuit components separately.

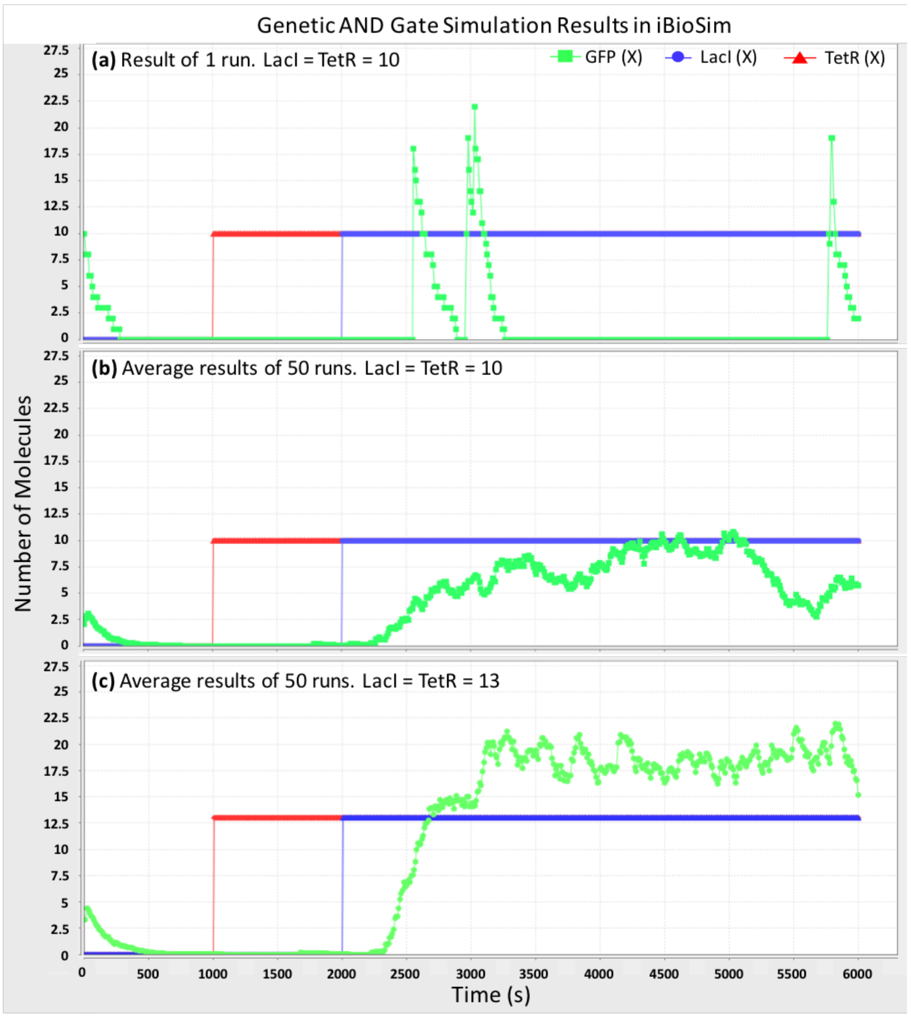


Figure 4.1: Preliminary analysis of a threshold value for the genetic AND gate using iBioSim [17]. (a) Stochastic simulation results when run only once. Average results of 50 runs (b) showing the lower threshold value of inputs LacI = TetR = 10 and (c) the upper threshold value of inputs LacI = TetR = 13.

4.1.2 Preliminary analysis of propagation delay

Another important factor for automatically obtaining the correct Boolean expression from the simulation data is the *propagation delay.* Figure 4.2 shows a zoomed-in version of Figure 4.1(c), which shows that the effect of changes in the input concentration is reflected in the output concentration after a time delay of approximately 700 seconds. That is, the output protein takes about 700 seconds to cross the level of the input concentration when the inputs are triggered to their threshold value. Thus, the propagation delay of a genetic circuit can be defined as follows:

Definition 4.2 Propagation delay: The time from when the input concentration reaches its threshold value until the corresponding output concentration crosses the same threshold value.

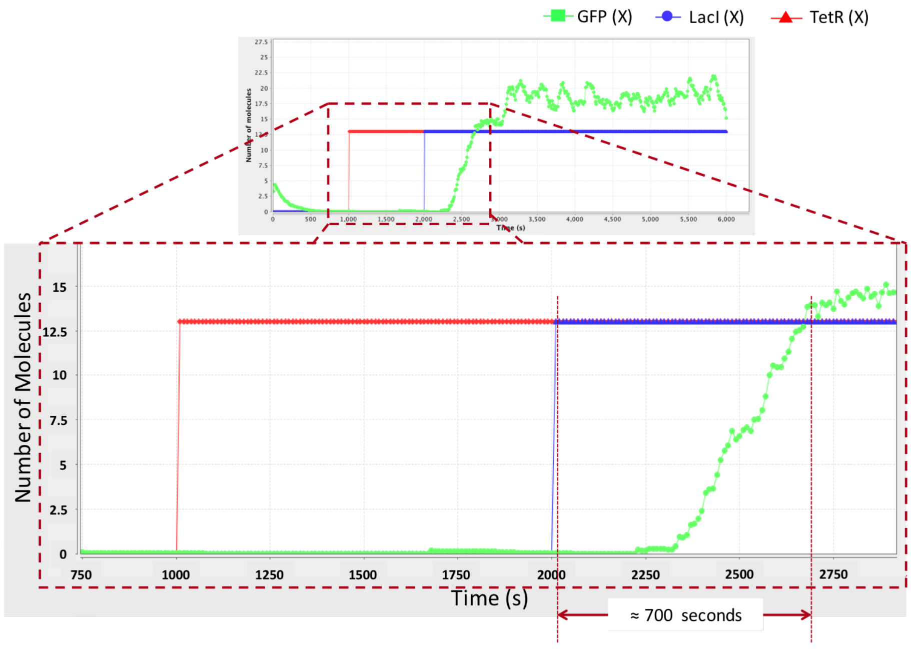


Figure 4.2: Zoomed-in image of Figure 4.1(c) indicating the preliminary propagation delay analysis using iBioSim. In this figure, the propagation delay is approximately 700 seconds.

Figure 4.2 shows that the output goes "high" after approximately 700 seconds from the time when both inputs have reached the significant concentration level (13 molecules). During these 700 seconds, the output remains low and hence does not produce the expected logic output. It also means that, during simulation (or even during experimentation in the laboratory), the user should not change the inputs before this time delay has elapsed.

In order to identify the threshold levels of a circuit in the laboratory, the biologist could perform this analysis by adding the input concentration periodically to see if it significantly affects the concentration of the output. To identify the input concentration, which significantly affects the output, different input combinations must be tried with different concentration levels, which is a very tedious and time consuming task to do in the laboratory. Furthermore, as mentioned above, it must be ensured that each input combination is applied after a certain time delay.

4.1.3 D-VASim plug-in for threshold value and propagation delay analyses

An algorithm is developed to automate the abovementioned iterative processes of identifying the threshold levels and propagation delays. This algorithm is integrated as a plug-in tool in D-VASim. Since the behavior of a genetic circuit is well described by stochastic simulations, therefore the proposed method is applied on the stochastic behavior of a genetic circuit obtained from the Gillespie's stochastic simulation algorithm [56, 59].

The algorithm for the threshold value and propagation delay analysis is shown as a pseudo-code in Algorithm 4.1. The algorithm is initialized by some user-defined parameters as indicated in Algorithm 4.1. C_{in} specifies the value of the input protein(s) concentration, from which the tool should start its threshold analysis. The *Inc* is the value with which the input concentration is increased for each iteration, in order to observe if the resulting concentration level of input affects the concentration of the output. The C_{inE} value specifies the input concentration at which the algorithm should stop the analysis of the threshold value. The algorithm also requires an initial assumption of the input-output propagation delay value, T_D. It is already mentioned earlier that the input-output propagation delay value is critical for extracting the correct logic behavior of a circuit model. Thus, it is necessary to wait until this time value has elapsed before applying the next combination of inputs. Since the time delay value is unknown for the automatic analysis, the tool begins the analysis with an assumed value and later estimates the approximate one. Assuming a higher value increases the estimation time but gives a better estimation of the threshold value.

Algorithm 4.1: Threshold value and propagation delay analysis.

```
input : $C_{in}$, $Inc$, $C_{inE}$, $T_D$, $S_T$, $i$, $O_S$, $V_T$, $OC_{DUTh}$, $OC_{DLTh}$
begin
    for all input combinations do
        if (Current input concentration level ($C_{inC}$) == 0) then
            Determine the initial output concentration ($C_{Oinit}$);
        else
            while ($C_{inC} \leq C_{inE}$) do
                while (Current Time 1 ($T_{C1}$) $\leq T_D$) do
                    Execute simulation
                    if ($C_{Os} > C_{inC}$)** then
                        // $C_{Os}$ = concentration of selected output specie
                        $PT = C_{inC}$ // PT = possible threshold value
                        // Verification process
                        for (number of iterations i) do
                            while (Current Time 2 ($T_{C2}$) $\leq V_T$) do
                                Execute simulation
                                if ($T_{C2} > S_T$) then
                                    Trigger the input to the value of PT
                                end
                                Store the output concentration data in array
                            end
                            Take the running average of all output i arrays
                        end
                        Estimate time delay $T_E$ and consistency $OC_E$
                        Terminate current while loop
                    end
                end
                if ($C_{Os} > C_{inC}$)** then
                    if ($OC_E > OC_{DUTh}$) then
                        Consider lower threshold value = 0 if not already
                        found Return the results and terminate all loops
                    else if ($OC_E < OC_{DLTh}$) then
                        Save lower threshold level and resume analysis
                    else
                        Resume Analysis
                    end
                end
                $C_{inC} = C_{inC} + Inc$
                $T_{C1} = T_{C2} = 0$
            end
        end
    end
    // **Valid when $C_{Oinit}$ is low, For high $C_{Oinit}$, it will become
    // ($C_{Os}$ < $C_{inC}$)
end
```

For a simulation, if every node of a genetic circuit model is not initialized to a stable value, then the output of some of the genetic circuits are initially unstable and exhibit unexpected behavior for a certain amount of time. For example, in the simulation traces (see Figure 4.1) of genetic AND circuit (shown in Figure 3.5), the initial values of LacI and TetR are zero, but when the simulation starts, the output, CI, of a first circuit's component (i.e. NAND gate, see Figure 3.5) is also zero, which enables the inverter and produces GFP until the input value 0 propagate through the NAND gate.

In order to perform correct timing analysis, it is therefore required to initialize all the circuit nodes to a stable value. If the values are not initialized, it is important that the algorithm should wait for the circuit's output to become stable first. The parameter, S_T *(Settling Time)*, helps the user to specify a rough value for the initial time during which the circuit's output is expected to become stable. When the algorithm performs the automatic analysis, it waits for the value defined for S_T to allow the circuit's output to become stable first and then triggers the input combinations to determine the appropriate threshold and propagation-delay values of a circuit. For small genetic circuits, containing a single gate only (for example, NOT, NAND and NOR), and having a low degradation rate (kd $\approx$0.0015), it is observed through simulations that these circuits usually take at least 1000 time units to become stable. This implies that, for these circuits and kd, the S_T parameter should not be less than 1000 time units. If a value less than this is chosen, then the algorithm will not be able to produce the correct estimation.

The algorithm further verifies the obtained threshold value by iterating the model for a predefined number of iterations, i. During this iterative verification process, the algorithm obtains the average propagation delay by running the model for the length of time defined by V_T for each iteration i. It also identifies the extent to which the average output for the estimated threshold value is consistent. In order to understand this procedure, lets assume the parameters values shown in Table 4.1. The unit for concentration here is the "number of molecules".

Now consider the sample time scale plots shown in Figure 4.3. To find the threshold value of the input concentration that significantly affects the output concentration, a specific input combination should be applied. This means that all the possible combinations should be checked one by one until the specific combination of inputs that triggers the output concentration is found. For logic circuits like AND, NAND, OR, NOR and NOT, the output transition can be observed by triggering both the inputs to the same concentration level at the same time. The algorithm, therefore, triggers both the inputs combinations from 00 to 11 first, instead of following the traditional pattern of 00 –> 01 –> 10 –> 11. Because of this, the algorithm estimates the threshold value of some

Table 4.1: Sample values of parameters required for threshold value and timing analyses.

Parameter name	Value
C_{in}	0
Inc	2.75
C_{inE}	15
T_D	800
S_T	200
i	10
V_T	1000
OC_{DUTh}	90
OC_{DLTh}	30

circuits, for example AND gate, relatively faster.

Figure 4.3 shows how the process of automatic threshold value and timing analysis takes place by the proposed algorithm. If more than 90 percent of the average output data, between instants t2 and t3, remains above the input level, the input concentration level is considered to be the upper threshold level. Similarly, if less than 30 percent of the average output data remains above the input level between instants t2 and t3, the input concentration level is considered to be the lower threshold level. The propagation delay is measured from the instant when the input is triggered, from its lower threshold level to its expected upper threshold level, to the instant when the average output crosses the same input level.

Figure 4.3(a) shows the case of input logic combination “11”, i.e., when both inputs are triggered high. According to the settings shown in Table 4.1, the algorithm runs the model first by keeping the input concentration zero until the assumed time delay of 800 time units has elapsed. In order to determine if the output concentration crosses the level of input concentration as defined in Definition 4.1, or in other words, to determine whether the output concentration goes above the input concentration level or falls below it, the initial concentration of output protein at input logic level combination “00” must be known. Therefore, during the first 800 time units (T_D), the average of the initial output concentration is obtained by keeping the concentration of both inputs zero i.e. logic 0. On the basis of this average initial output concentration, the estimation of output concentration crossing the input concentration level is performed.

Once the assumed time delay has elapsed, the input concentration level is incremented to the next level, indicated by line 33 in pseudo-code 4.1. The example case shown in Figure 4.3(a) portrays the scenario of an AND gate where the

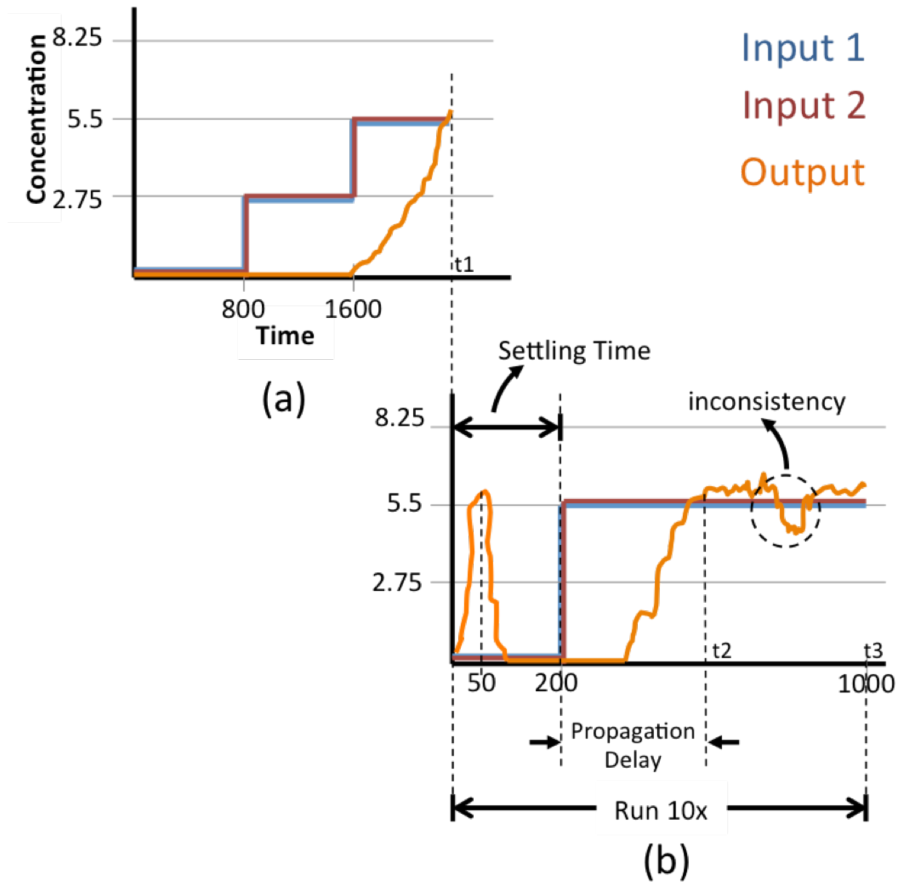


Figure 4.3: Sample time scale plots of the genetic AND gate. (a) First loop to detect the threshold value. (b) Separate loop to verify the estimated threshold value repeatedly for predefined number of iterations, i (10 times in this case).

initial average output of a circuit (with both input concentrations at zero) is zero. The algorithm also works for the case where the average initial output concentration is high, for instance, a NOT gate, by iteratively increasing the input concentration and checking if the output concentration falls below the concentration level of input. Note that this still satisfies Definition 4.1.

Point *t1*, shown in Figure 4.3(a), implies that the algorithm halts the current loop execution when the value of the output protein crosses the input concentration level. This anticipates the possible threshold value (5.5 molecules in this example) as it makes the output concentration cross the input concentration level. To verify this threshold value, the algorithm executes a separate loop to run the simulation of the circuit model for the defined number of iterations, 10 times in this example, as shown in Figure 4.3(b). This process executes in lines 11-20 in pseudo-code 4.1.

In order to measure the correct propagation delay, it is necessary to trigger the input protein, particularly from zero to the expected threshold level, only when the model's initial output is settled. As mentioned before, the outputs of some circuits are unexpectedly high which gradually settles to zero. This scenario is depicted in Figure 4.3(b). Therefore, the initial concentrations of input proteins must not be triggered to their expected threshold level until the output becomes stable. As mentioned above, the parameter Settling Time, S_T, lets the user provide a period of time by which the initial output is expected to become stable. This is the time at (or after) which the algorithm triggers the inputs, to their expected threshold level, to determine the time it takes to trigger the output concentration. If a low value is assumed for S_T, the algorithm may produce incorrect propagation delay. For example, in Figure 4.3(b), if a value 50 would be chosen as a settling time, the inputs would be triggered at 50 time units. At this instant, when the inputs are triggered to their expected threshold level, the concentration of output is already above the threshold level and thus the algorithm would end up estimating the propagation delay to zero. Therefore, depending on the complexity of a circuit and the degradation rate (kd), this value should be chosen carefully.

The simulation output data from all 10 iterations are averaged to obtain the average estimated propagation delay and the inconsistency present in the output plot for the estimated threshold values. The inconsistency, illustrated in Figure 4.3(b), is calculated by determining the size of the average output data, which is less than the input concentration level immediately after the output crosses the input level for the first time, i.e. the inconsistency is estimated between points *t2* and *t3* as shown in Figure 4.3(b). In other words, for examining the upper threshold level, the idea is to determine how consistently the average output data remains above the input concentration level between points *t2* and *t3*. The algorithm accepts the estimated threshold value based on the user-defined parameter, *% acceptance of consistency*, shown as OC_{DUTh} (for upper threshold level) and OC_{DLTh} (for lower threshold level) in pseudo-code 4.1. The results are accepted if the estimated consistency is greater (for upper threshold) and less (for lower threshold) than the user-defined values, OC_{DUTh} and OC_{DLTh}, respectively. This is shown in the lines 25-30 in pseudo-code 4.1. The results are otherwise discarded and the algorithm resumes the analysis from point *t1*, shown in Figure 4.3(a). The percentage output consistency is calculated according to equation 4.1.

$$\%\ output\ consistency = \frac{O_{t2-t3} - D}{O_{t2-t3}} \tag{4.1}$$

where:

O_{t2-t3}	= Size of the average output data between instants *t2* and *t3* (Figure 4.3(b)).
D	= Deviation, which defines the number of times the output data is found to be deviated from the expected (greater or less than the) threshold value.

The quantity D in equation 4.1 is considered to be different in two different cases, i.e., when the initial input concentration is found low, then D in eq. 4.1 indicates the number of times the output data is found “less” than the threshold value, as in the case shown in Figure 4.3. Else, if the initial input concentration is high, then D signifies the number of times the output data is found “greater” than the threshold value. For the sample parameters (Table 4.1) used for the sample plots shown in Figure 4.3, the algorithm estimates the input concentration as the upper threshold level if the output consistency is 90 percent or above. Likewise, the input level is assessed as the lower threshold level if the estimated output consistency is less than 30 percent.

4.2 Experimentation by Simulation

The timing analysis is performed on the nine genetic logic circuit models (Figure 4.4) proposed in [21] and on one of the SBML models of real genetic circuits [14]. The genetic implementation and the description of these circuits can be found in [21] and [14], respectively. These circuits are considered fairly complex in the context of genetic circuits, because each gate is composed of several genetic components. Their kinetic interactions are described by a number of mathematical equations in the SBML model. The SBML models of these genetic logic circuits are run on D-VASim and their threshold value and propagation delay analyses are performed.

In microelectronic devices, the behavior of a circuit depends on many different parameters. For example, in MOS transistors, the drain current depends on the width and length of gate, oxide capacitance, gate-to-source voltage etc. [60].

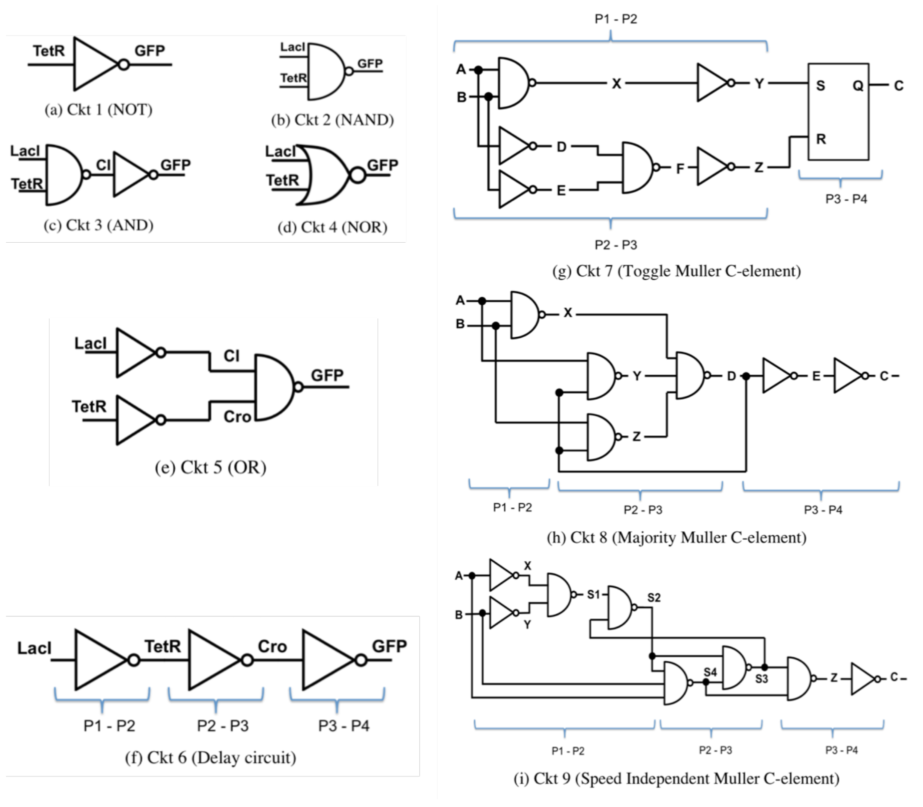
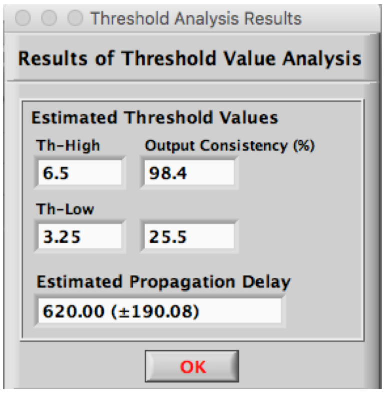


Figure 4.4: Experimental genetic circuits, obtained from [21], for timing analysis. More complex circuits, (f)-(i), are further categorized into three intermediate levels P1-P2, P2-P3 and P3-P4. The timing analyses are performed on these three levels separately, which are mentioned in Table 4.2. The SR latch shown in Ckt 7 is asynchronous and do not require a clock input.

Similarly, the behavior of a genetic circuit also depends on different parameters, including degradation rate, forward repression binding rate, forward activation binding rate etc. These parameters of a genetic circuit model are described in the SBML file. We carried out simulations on these ten genetic circuit models by observing the effects of varying the degradation rate (kd) on the propagation delay and the threshold value of a circuit.

The degradation rate is the rate at which a chemical compound (e.g. a protein) is decomposed into intermediate products, i.e., a produced protein will be effective only for a certain period of time determined by the degradation rate. A zero degradation rate means that the protein does not degrade and hence will be effective forever. This is an often-used assumption, which clearly is not realistic, which is why understanding the impact of the degradation rate on the timing analysis is an important investigation.

The threshold value and propagation delay of each circuit is obtained for five different values of *kd (0.0015, 0.0055, 0.0095, 0.0135, 0.0215)*. These set of values are chosen based on a degradation rate value used in [21]. It has been experimentally observed that the variation in degradation rate (kd) greatly effects the settling time, S_T, of an output. Hence, for each circuit, we chose different parameter values (shown in Table 4.1), except the number of iterations, i, and *% acceptance of consistency* for upper and lower threshold values (OC_{DUTh} and OC_{DLTh}), which were set to 5, 70 and 30, respectively for all circuits. We purposely used $OC_{DUTh} = 70$ percent in order to demonstrate that it is affected by threshold values.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

Th-High	Output Consistency (%)
6.5	98.4
Th-Low	
3.25	25.5

Estimated Propagation Delay

620.00 (±190.08)

Figure 4.5: Results of threshold value and propagation delay analysis of Ckt 8 generated by D-VASim for $kd = 0.0135$. The estimated upper and lower threshold values are 6.5 and 3.25 molecules with 98.4 percent and 25.5 percent consistency, respectively. The approximate input-output propagation delay value is 620 time units with ± 190.08 standard deviation.

Figure 4.5 shows how D-VASim reports the outcomes of a threshold value and propagation delay analysis once the algorithm finishes execution. This figure shows the threshold value and timing analysis results obtained for Ckt 8 (Figure 4.4(h)) when degradation rate (kd) was set to *0.0135*. It indicates that the estimated upper and lower threshold values are 6.5 and 3.25 molecules with 98.4 percent and 25.5 percent consistency, respectively. It also calculates the approximate input-output propagation delay value to 620 time units with a

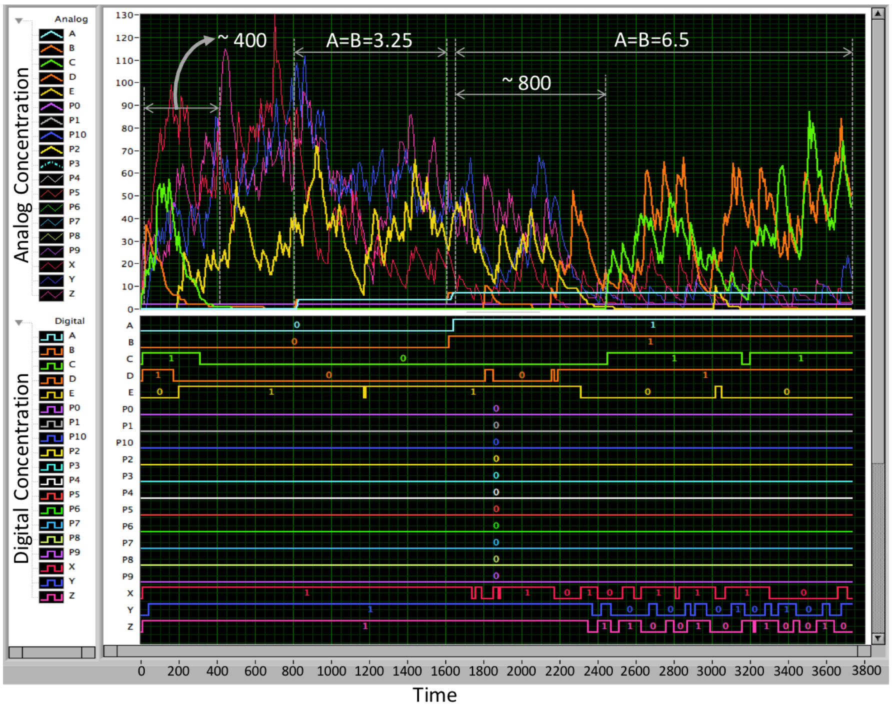


Figure 4.6: Analog simulation traces of Ckt 8 with its corresponding digital waveforms for $kd = 0.0135$.

standard deviation of ±190.08 , in this case based on five iterations. When the results are obtained, the user may interact with the model during run time; apply all the possible input combinations in a significant amount, and match the propagation delay with the one estimated by D-VASim.

Figure 4.6 shows the simulation traces of the same Ckt 8 with $kd = 0.0135$. To keep this chapter concise, only the screen shots of the analysis results and graphical simulation of Ckt 8 (for kd = 0.0135) are included. However, the complete experimental data of all circuits is available in Appendix B. In Figure 4.6, the circuit's inputs are A and B; and the output is C. It can be observed that the initial concentration of output protein, C (shown as green plots), is high above the threshold value and it takes approximately 400 time units to settle down. Furthermore, when the input concentrations are triggered to their lower threshold level, i.e. 3.25 molecules, the output concentration remains zero. When the input concentration levels are triggered sharply to their estimated threshold value i.e. 6.5 molecules, the output of a circuit triggers high approximately after 800 time units.

4.2.1 Effects of varying *kd* on the threshold values and propagation delays

Figure 4.7 shows the graphical plots of timing analysis of all circuits. The values of propagation delays are plotted along the y axis on the left-hand side. The threshold values and percentage of output consistency for each value of kd is plotted along the y axis on the right-hand side. The x axis contains the degradation rate values. The general impression of these experiments is that the propagation delay of genetic circuit decreases with the increase in degradation rate (kd). This is expected because when the degradation rate is high, the protein degrades faster and thus contributes in the reduction of propagation delay. However, the propagation delay does not seem to have an inverse relation with the degradation rate. The propagation delay for all circuits dropped considerably with the first decrement of 40×10^{-4} in kd; and then it decreases slowly for the next higher values of kd.

The standard deviation in propagation delays, calculated for the specified number of iterations (i.e. five in these experiments), is also included for each circuit in the plots shown in Figure 4.7. It can be noticed that the propagation delay of a circuit is more variable for low degradation rates. The variation in the propagation delay decreases with the increase in degradation rate; however, a high degradation rate makes the cascaded circuit's output unstable. This is because the genetic components decay quickly when the degradation rate is high, thus causing the circuit's logic to switch faster even when a small input concentration is applied. This also reduces the transition region between the upper and lower threshold levels, as shown in the data of Ckt 8 in Figure 4.7. However, it can also be noticed for Ckt 8 that a transition region is small for $kd = 0.0135$ (i.e. 3.25) as compared to $kd = 0.0215$ (i.e. 6.5). This is because, at $kd = 0.0215$, Ckt 8 becomes unstable and produces glitches of high output even when the input concentration levels were kept to zero (see simulation traces in the Appendix B.8). This is the reason why the lower threshold value of Ckt 8 at $kd = 0.0215$ is estimated to be zero. It has been observed that the lower threshold value of all circuits approaches zero with the increase in kd.

4.2.2 Effects of varying threshold values on the propagation delays

Beside these analyses, some other interesting facts of varying the upper threshold value on the propagation delay of a circuit has been observed. It is noted that the smaller concentrations of input protein have a weak impact on the output protein, which is analogous to the behavior of microelectronic devices. For

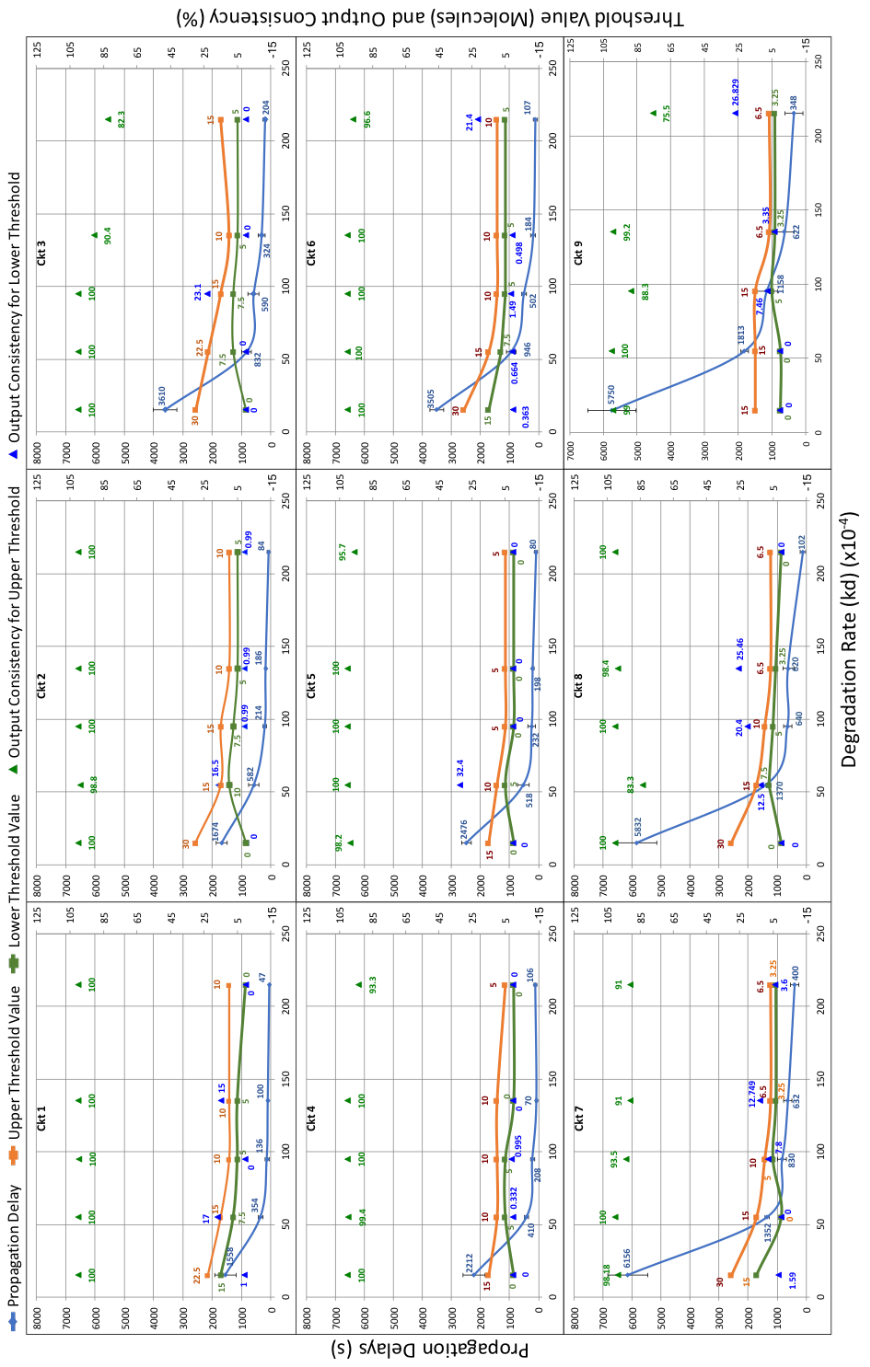


Figure 4.7: Effects of varying degradation rate (kd) on the propagation delay of genetic logic circuits.

instance, in MOS transistors, weak applied V_{GS} (gate-to-source voltage) results in the weak drain current, I_D [60]. This effect can be observed in the graphical plot of Ckt 4 in Figure 4.7. In this plot, for $kd = 0.0215$, the threshold value of a circuit is reduced to 5 molecules as compared to its previous data point, which is 10 molecules at $kd = 0.0135$. Due to the increment in *kd*, the propagation delay at this point is supposed to decrease if the input threshold value remains the same. However, it slightly increases because the input threshold is reduced to 5 molecules. This effect has been observed on other circuits as well during the run-time simulation. For instance, increasing the applied input concentration to 60 molecules for Ckt 7 at $kd = 0.0015$, the propagation delay is decreased from 6156 to 5570 time units (see Appendix B, Figure B.7a). This inverse relation between propagation delay and threshold value holds true to a certain extent, and then further increment or decrement in the applied input concentration does not affect the propagation delay.

4.2.3 Effects of varying threshold values on the *% output consistency* at high *kd*

For higher values of kd, the output consistency of the upper threshold level is increased by reducing the threshold value. This is shown in the plots of Ckt 9 in Figure 4.7. The output consistency of the upper threshold level, at $kd = 0.0135$, was reduced to 49 percent (not shown in Figure 4.7) when the threshold value was set to 30 molecules. Then the output consistency of Ckt 9 was analyzed, at kd = 0.0215, by keeping the threshold value to the same level, i.e. 30 molecules (not shown in Figure 4.7), and noticed that the output consistency was decreased to 2 percent. Then the threshold value was decreased to 6.5 molecules and the output consistency was increased to 75.5 percent, as shown in Figure 4.7.

4.2.4 Other parameters effecting the threshold values

The values of upper and lower threshold levels also depends on the parameter, *Inc* (see Table 4.1), which specifies the input concentration to be added to the previous input concentration level during each iteration, *i*. For example, in the case of Ckt 2 and Ckt3, the value of *Inc* was set to 30 at $kd = 0.0015$. The algorithm thus triggers the input concentration from 0 to 30 directly during the analysis. Because of this, the average output was found to be 100 percent consistent for upper threshold level, which results in estimations of the upper and lower threshold levels of 30 and 0 molecules, respectively. If a lower value of *Inc* would be chosen, the results would be different but more precise.

4.2.5 Intermediate propagation delays

The intermediate delays of larger genetic circuit models are also analyzed by splitting them into three points of measurements, as shown in Figure 4.4. The propagation delays for each of these points are mentioned in Table 4.2. The propagation delay, indicated by a point of measurement *P1-P4* in Table 4.2, is the entire circuit propagation delay. The reader should not confuse these estimations with those depicted in Figure 4.7. The results mentioned in Figure 4.7 are estimated by D-VASim using the proposed algorithm; and the results mentioned in Table 4.2 are those that are obtained by a user through a run-time stochastic simulation.

Table 4.2: Intermediate propagation delays of genetic logic circuits.

kd	**Points of**	**Propagation delays (s)**			
($x10^{-4}$)	**measurement**	Ckt 6	Ckt 7	Ckt 8	Ckt 9
15	P1 - P2	1763.12	3243	1863	4379
	P2 - P3	1067.91	4764	1237	3854
	P3 - P4	554.53	1140	2500	1292
	P1 - P4	3350	5904	5500	6463
55	P1 - P2	382	603	394	936
	P2 - P3	193.7	973	425	395
	P3 - P4	246.4	410	405	171
	P1 - P4	805	1400	1224	1646
95	P1 - P2	80	340	122	441
	P2 - P3	100	380	235	53.045
	P3 - P4	180	280	235	64
	P1 - P4	360	664	631	1003
135	P1 - P2	83	230	39	311
	P2 - P3	81	265	253	112
	P3 - P4	54	240	279	330
	P1 - P4	215	506	573	878
215	P1 - P2	43	229	70	9
	P2 - P3	70	173	103	120
	P3 - P4	58	47	37	81.63
	P1 - P4	56.5	267.75	208	400

The argument that a circuit's output becomes unstable for larger values of *kd*, can also be supported by observing the intermediate delays of Ckt 6 for $kd = 0.0215$ in Table 4.2. As shown in Figure 4.4, Ckt 6 is composed of three inverters connected back-to-back in series. When input protein LacI is triggered to its threshold value, it suppresses the production of TetR. When the concentration of TetR drops below its threshold level, it produces Cro, which in turn suppresses the production of output protein, GFP. However, the intermediate propagation delays of Ckt 6 for $kd = 0.0215$ shows that when the input protein, LacI, is triggered to the estimated threshold value, the overall output of a circuit, GFP, is produced in 56.5 time units. However, one of the intermediate outputs, Cro, is produced in a significant amount after ≈70 time units, which is greater than the propagation delay of the entire circuit. This invalidates the desired circuit's behavior and makes the output unstable, which indicates that the circuit does not behave as designed.

4.2.6 Experimentation on the SBML model of real genetic circuit

The possibility of analyzing the SBML models of real genetic circuits is also explored. The genetic AND gate circuit (composed of inverters and NOR gates) from [14] is chosen for the experimentation. The genetic circuits presented in [14] were first developed on a tool named *Cello*, which generates the SBOL file. Unlike SBML, the SBOL representation does not describe the behavior of a biological model. Therefore, the SBOL-SBML converter [61] is used to generate the behavioral model of the above-mentioned real genetic AND circuit. This SBOL-SBML converter is available as a plug-in in iBioSim [17], which uses the default parameters while defining the reaction kinetics during the conversion process.

Since, the actual parameters, like degradation rate, forward repression binding rate etc., are not disclosed in [14], therefore the default iBioSim parameters are used to perform the timing analysis of real genetic AND gate circuit. However, the parameter values can always be changed, and new parameters can also be added to observe more realistic results. Furthermore, the SBOL file generated by Cello does not include the input sensor block of a circuit (which includes the input inducers); thus these inducers are also not included during the SBOL-SBML conversion process. Hence, we added the input inducers manually in the SBML model using iBioSim, as shown in Figure 4.8. The components inside the yellow-dashed box are manually added, and rest of the model is a result of SBOL-SBML conversion process. In this figure, it is shown that when both of the input inducers, aTc and IPTG, are present, they form a complex with their corresponding regulators, *TetR* and *LacI*, respectively. These regulators then gradually stop inhibiting their respective promoters, which eventually leads to the production of the output, yellow fluorescent protein (*YFP*).

Figure 4.9 shows the timing analysis results of the SBML model of the genetic AND gate circuit [14]. All these analyses for different degradation rates were obtained within 30 minutes, and the simulations with all possible input combi-

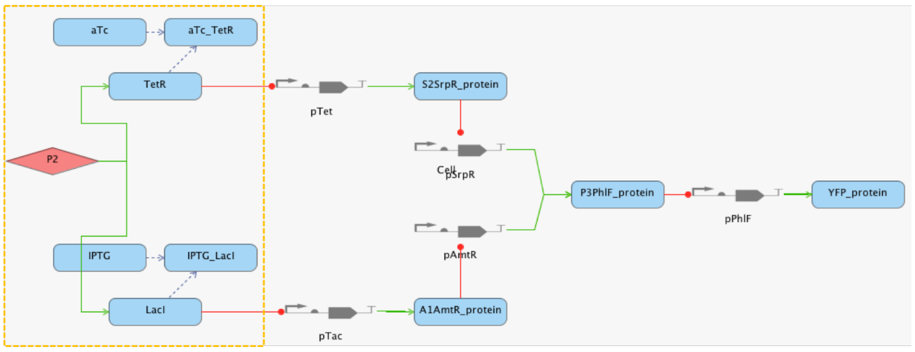


Figure 4.8: SBML design of the genetic AND gate circuit obtained from [14].

nations were performed within 10 minutes. This is obviously faster compared to testing the model in a lab, where the models were first placed in the logic-0 state for 3 hours and then switched to other possible states, one by one, each for another 5 hours [14].

Figure 4.9 indicates that the results of the genetic AND gate [14] are similar to those obtained for the other 9 genetic circuit models [21]. In general, it is observed that the propagation delay, threshold value, and the degradation rate are all interlinked. The output of a circuit is stable for small values of kd but it increases the propagation delay. The variation in the propagation delay is also greater for small values of kd. On the other hand, the output of a circuit becomes unstable for large values of kd but decreases the propagation delay. Large values of kd also contributes to a reduction of threshold value to a certain point. This is because the circuit becomes faster for large kd; therefore, a small input concentration is sufficient to trigger the output protein. The degradation rate cannot be increased beyond a certain point, because it makes the output highly oscillating. This implies that the threshold value of a circuit cannot be decreased beyond a certain point. This corresponds to the scaling trends for the MOSFET device, where the gate width cannot be reduced beyond a certain number of nanometers.

4.3 Discussion

In this chapter, a methodology to perform the timing analysis of genetic logic circuits is proposed, which is then implemented and tested in D-VASim. The threshold value and timing analysis are performed primarily on entire circuits instead of on each individual circuit component. However, D-VASim is also

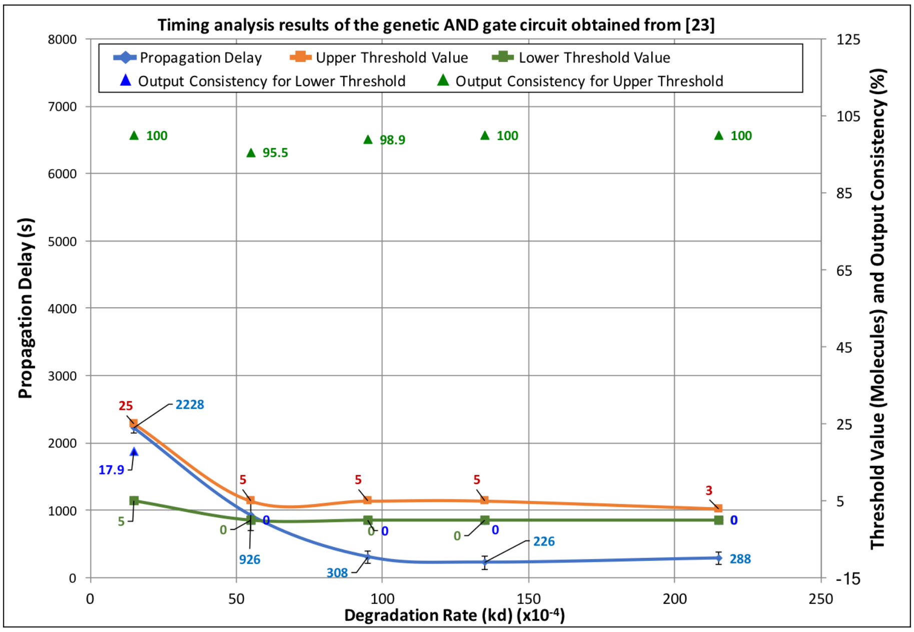


Figure 4.9: The effects of varying degradation rate (kd) on the propagation delay and threshold value of the genetic AND gate circuit [14]. The propagation delay of a circuit decreases with the increase in kd. Also, low input concentrations are required to trigger the output of a circuit at higher values of kd. Note: *These results may differ when the actual parameter values are used.*

able to analyze the threshold value and timing analysis of individual circuit components. In this work, D-VASim is shown to estimate the overall threshold value of an entire circuit, which gives user a minimum value of input species required to trigger the output of a genetic circuit.

The effects of circuits' timings upon varying certain parameters are also explored. This may assist genetic circuit designers in finding an appropriate set of parameters to achieve the desired timings of a genetic circuit. D-VASim could actually help reduce the time consuming in-vitro experiments (laboratory experiments) to analyze and design genetic circuits with desired behavior and timings. We anticipate that the ability of analyzing the timings of genetic circuit may open up a new research area, which may help biologists and scientists to design and characterize the timing properties of genetic circuits. Depending on the complexity of a genetic circuit and the user-defined settings for these analyses, D-VASim may take up to an hour to estimate the threshold value and propagation delays. This estimation time is still reasonable as compared to the number of days of laboratory experimentation, which are required only for a single combination of inputs with a specific set of parameters.

This methodology of threshold value and propagation delay analyses is used in the next chapter to perform the experimentation and logic analysis on some of the real genetic circuit models [14].

Chapter 5

Genetic Circuits Logic Analysis

After obtaining the *threshold value* and *propagation delay* of a genetic circuit through the procedure defined in Chapter 4, a user can begin experimenting on the genetic circuit model and apply all the possible input combinations. Once all the possible input combinations of a genetic circuit are applied, the experimental data (stochastic simulation data) can then be used to obtain (or validate) the logical behavior of a genetic circuit model.

In this chapter, the logic analysis and validation algorithm is presented, which extracts the logic behavior from the simulations and provide a fitness value that can be used to infer how likely it is that the circuit will actually work after implementation in the laboratory. The presented algorithm is scalable and able to analyze n-input genetic logic circuits. The logic analysis of genetic circuits is useful in two ways – first, it allows the user to verify more complex genetic logic circuits, build by cascading several genetic logic gates; secondly, it helps the user to extract the Boolean logic of a circuit even when the user does not have any prior knowledge about its expected behavior. Similar to *timing analyzer* plug-in, the proposed methodology for logic analysis has also been implemented as a plug-in tool in D-VASim. This approach has been tested on 15 different genetic circuit models, with different level of complexity, obtained from [21] and [14].

Some part of the work presented in this chapter has been published in the following peer-reviewed conference and workshop:

[62] Hasan Baig and Jan Madsen, "Logic Analysis and Verification of n-input Genetic Logic Circuits", *Design Automation and Test in Europe (DATE)*, pp. 654–657, 2017.

[58] Hasan Baig and Jan Madsen, "Logic and Timing Analysis of Genetic Logic Circuits using D-VASim", *8th International Workshop on Bio Design Automation (IWBDA)*, pp. 77–78, 2016.

5.1 Methodology

As discussed in Chapter 4, *threshold value* and *propagation delay* of I/O species are two important parameters required to obtain a correct Boolean expression of a genetic circuit. Definition 4.1 states that the *threshold value* defines a significant amount of concentration, which categorizes the analog concentrations into digital logics 0 and 1. Also, definition 4.2 specifies that the *propagation delay* is the time required to reflect the changes in input species concentrations on the concentration of output species.

During the experimentation, if the input species concentrations are applied below their threshold levels and each of the input combination is changed before the propagation delay has elapsed, then the circuit never produces a correct output for some of the input combinations. In this work, the timing analysis plug-in of D-VASim [39] is used to obtain the threshold value and the propagation delay of a circuit. These results are then used to perform experiments on genetic circuit models and log all experimental simulation data. The simulation data is then given to the proposed algorithm to extract the logical behavior of a circuit.

5.1.1 Overview

Algorithm 5.1 shows the pseudo code of the main procedure of the logic analysis and verification algorithm, which contains three sub-procedures; *CaseAnalyzer, VariationAnalyzer* and *ConsBoolExpr*, discussed separately in the Algorithms 5.2, 5.3 and 5.4, respectively. Some initial parameters (N, SD_{An}, Th_{VAL}, FOV_{UD}, I_S, and O_S) are required to execute the algorithm; where N corresponds to the total number of input species, SD_{An} refers to the simulation data of all I/O species, Th_{VAL} denotes the upper threshold value of I/O species, FOV_{UD}

Algorithm 5.1: The main procedure of logic analysis and verification algorithm.

```
1 begin
2     INITIALIZE ($N$, $I_S$, $O_S$, $SD_{An}$, $Th_{VAL}$, $FOV_{UD}$);
3     $SD_{size}$ = Calculate the size of analog simulation data, $SD_{An}$
4     $SD_{Dig}$ = ADC ($N$, $SD_{An}$, $SD_{size}$, $Th_{VAL}$)
      // $SD_{Dig}$ = digital simulation data
5     ($nc$, $Case\_O$, $Case\_I$) = CaseAnalyzer ($N$, $SD_{size}$, $SD_{Dig}$)
      // Case_O = Array holds the output values for each input
         combination
      // Case_I = Array holds the number of occurrences of each
         input combination
      // nc = total number of possible input cases (or
         combinations): $2^N$
6     ($O\_Var$, $HIGH\_O$) = VariationAnalyzer ($nc$, $SD_{Dig}$, $Case\_O$)
      // O_Var = Array to monitor variations in the output for
         each case, nc
      // High_O = Array to hold the number of times the output
         is high for each case, nc
7     ($BoolExpres$, $PFoBE$) = ConsBoolExpr ($O\_Var$, $Case\_I$,
        $HIGH\_O$, $nc$, $N$, $FOV_{UD}$)
      // BoolExpres = Contains the estimated Boolean expression
      // PFoBE = Specifies the percentage fitness of estimated
         Boolean expression in the simulation data
8 end
```

is the user-defined percentage of acceptable variation in the output data (described later), and I_S and O_S specify the names of input and output species, respectively. By giving users an ability to select the input and output species, they can perform Boolean logic analysis on the entire circuit as well as on the intermediate circuit components.

In the simulation of electronic circuits, a logical abstraction is typically applied in which it is only considered if the wire is in high or low state, instead of tracking the exact voltage value. In order to utilize a similar abstraction level here, the algorithm first converts the analog simulation data into digital data with the help of upper threshold values extracted from the *timing analyzer* plug-in of D-VASim (see Chapter 4). This step is shown as the sub-procedure **ADC** at line 4 in Algorithm 5.1. The algorithm scans the chosen N input and an output species and converts their analog values in to digital values, based on the upper threshold value provided. Once the analog data is converted to logic high and low, the exact concentration of proteins are no longer needed in order to obtain the Boolean logic of a genetic circuit.

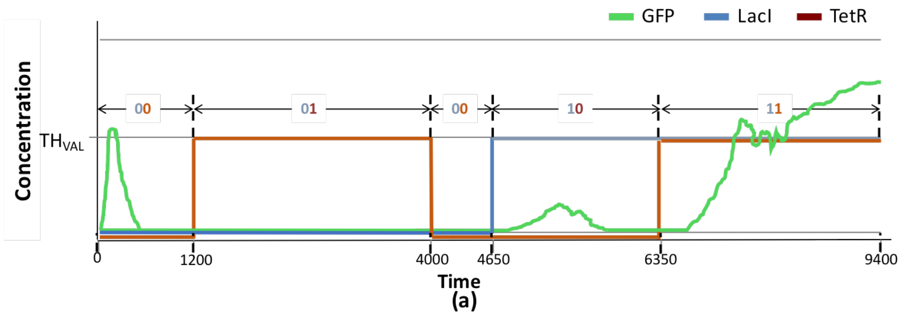
5.1.2 Input combinations analysis

The response time of a genetic circuit is important in order to obtain the correct behavior. Therefore, each input combination has to be applied for sufficient amount of time to observe its correct response on the output species. In electronic circuits, the signals propagate in separate wires and applied voltage remains constant. However, the signals in genetic circuits are molecules drifting in the same volume of a cell and easily merge with the concentrations of other compounds. Due to this, the concentrations of a species in a genetic circuit always varies, and may go up or down below the threshold level at any instant of time. Because of this unstable behavior, for each input combination, it is required to obtain continuous binary streams of output species to extract the correct behavior of a genetic circuit.

Algorithm 5.2: Pseudo code of the procedure **CaseAnalyzer**.

```
   input  : $N$, $SD_{size}$, $SD_{Dig}$
   output: $nc$, $Case\_O$, $Case\_I$
1  begin
2      $nc = 2^N$;
3      Set Array Case_I[$nc$] = 0;
4      Set Array Case_O[$nc$][SD$_{size}$] = 0;
5      Set $icv$; // to read Input Case Values
6      for $all j \in SD_{Dig}$ do
7          $icv$ = At $j^{th}$ value, read the value of corresponding inputs'
             combination;
8          Case_I [$icv$] = Case_I [$icv$] + 1;
9          if ($j^{th}$ value of $SD_{Dig}$ output-specie for input case $icv$ == 1) then
10             Set Case_O [$icv$][$j$]) = 1;
               // Note that the Output is High for specific case
                   icv at simulation instant j
11         else
12             Set Case_O [$icv$][$j$]) = 0;
               // Note that the Output is Low for specific case icv
                   at simulation instant j
13         end
14     end
15 end
```

The next sub-procedure, *CaseAnalyzer*, shown in Algorithm 5.2 analyzes the



Inputs (*i*)	Input occurrences (*Case_I[i]*)	Output stream for each input combination (*Case_O[i]*)	Output occurrences (*HIGH_O[i]*)	Unstability (*O_Var[i]*)
00	1850	00011100...................00000	3	2
01	2800	000.............................00000	0	0
10	1800	000.............................00000	0	0
11	3050	00..011...10011010001...11	1875	7

(b)

Figure 5.1: Logic analysis and verification process. (a) Sample plots of 2-input genetic AND gate. (b) Sample data for illustrating the input case and variation analysis.

number of times each input combination occurs (line 8) and logs their corresponding output binary data streams (line 9-12). In order to understand this procedure, consider the sample simulation plots in Figure 5.1(a), which are produced from the 2-input genetic AND gate of Figure 3.5. *CaseAnalyzer*, processes the data and generates output as depicted in the first three columns in Figure 5.1(b). This data express, for each input combination, the number of simulated data points as well as the output digital data stream of logic-0 and 1 converted according to the upper threshold levels. In this example, the case of input combination 00 appears about 1850 times in total. The small glitch between 4650-6350 time units indicates the stochastic nature of the model. It shows that the logic-0 of GFP may refer to a concentration which is less than its threshold value but may not be sharply zero. Also, the output of some genetic circuit models is initially high which gradually reduces down to zero, as shown in Figure 5.1(a). These unwanted high peaks should be filtered out to obtain the correct Boolean expression.

For each input combination, the corresponding data stream of the output species is also extracted, as shown in the third column of the table shown in Figure 5.1(b). In this example, the output data stream contains binary 1's for two input combinations – 00 and 11 . In this case, it is already known that the output of a circuit is initially high, when both of the inputs are low, and settle down to zero gradually. Furthermore, Figure 5.1(a) depicts a short period of time in which the output oscillates around the upper threshold value (between 6350-9400 time units), before entering into a stable logic-1 state. This happens when both inputs are triggered high (i.e. 11). In order to examine such scenarios, the digital output data streams, corresponding to each input combination, are analyzed for stability through the sub-procedure, *VariationAnalyzer*, (line 6 in Algorithm 5.1).

5.1.3 Variation analysis and Boolean expression construction

The pseudo code of the sub-procedure *VariationAnalzyer* is shown in Algorithm 5.3. *VariationAnalzyer* examines the output data stream (lines 8-27) and counts how many times the output oscillates (or varies) between logic-1 and 0. It first calculates the number of times a logic-1 appears for a specific input combination (line 19). In the example shown in Figure 5.1(b), the logic-1 appears for 3 and 1875 times for the input combinations 00 and 11, respectively. It then analyses for each of these input combinations changing 0-to-1 and 1-to-0 (i.e. how many times the output varies). In Figure 5.1(b) this happens twice for input combination 00 and 7 times for 11. Since the output is high when both the inputs are the same, one may end up estimating the logical behavior of this circuit to be an XNOR gate if the simulation data is not filtered out correctly.

In order to obtain the correct Boolean expression, two filtrations of the data are performed by the sub-procedure, ConstBoolExpr (line 7 in Algorithm 5.1). The first one is the calculation of fraction of variation according to equation 5.1;

$$FOV_{Est_i} = \frac{O\_Var[i]}{Case\_I[i]} \tag{5.1}$$

where:

i	= Input combination at which the output is high at least once.
$O\_Var[i]$	= Number of variations in the output, for i.
$Case\_I[i]$	= Number of times the input combination i occurs in the simulation data.

Note that the value of Case_I[i] is always equivalent to the length of its corresponding output data stream. In other words, if any input combination occurs four times, then the length of its corresponding output stream is four, because the output also appears four times.

Algorithm 5.3: Pseudo code of the procedure **VariationAnalyzer**.

```
   input  : nc, Case_O
   output: O_Var, HIGH_O
1  begin
2      Set Prev_OP_State = 0; // Previous digital state of output specie
3      Set Curr_OP_State = 0; // Current digital state of output specie
4      Set O_Var [nc] = 0;
5      Set HIGH_O [nc] = 0;
       // loop through all input cases nc
6      for all $i \in nc$ do
           // loop through digital data of output species
7          for all $j \in Case\_O$ do
8              if ($j^{th}$ value of Case_O for input case i == 0) then
9                  if (Prev_OP_State == 1) then
10                     Increase O_Var [i] (i.e. for $i^{th}$ input case) by 1;
11                     Set Curr_OP_State = 0;
12                     Set Prev_OP_State = Curr_OP_State;
13                 else
14                     Keep O_Var [i] to its previous value;
15                     Set Curr_OP_State = 0;
16                     Set Prev_OP_State = Curr_OP_State;
17                 end
18             else
                   // Count number of times the output is high for $i^{th}$ input
                       case
19                 HIGH_O [i] = HIGH_O [i] + 1;
20                 if (Prev_OP_State == 0) then
21                     Increase O_Var by 1;
22                     Set Curr_OP_State = 1;
23                     Set Prev_OP_State = Curr_OP_State;
24                 else
25                     Keep O_Var [i] to its previous value;
26                     Set Curr_OP_State = 1;
27                     Set Prev_OP_State = Curr_OP_State;
28                 end
29             end
30         end
31     end
32 end
```

In the example shown in Figure 5.1, the estimated fraction of variations – FOV_{EST}, for input combinations 00 and 11, are 2/1850 and 7/3050, respectively. This indicates that only a small fraction of output, in comparison to its whole size for specific input combination, is varied. This estimated fraction of variation, FOV_{EST}, is compared with the user-defined fraction of variation, FOV_{UD}, and the results are accepted if the estimated value is less than the user-defined one. In the experimentation of this work, up to 25 percent variation ($\text{FOV}_{UD} = 0.25$) is allowed in the output data streams.

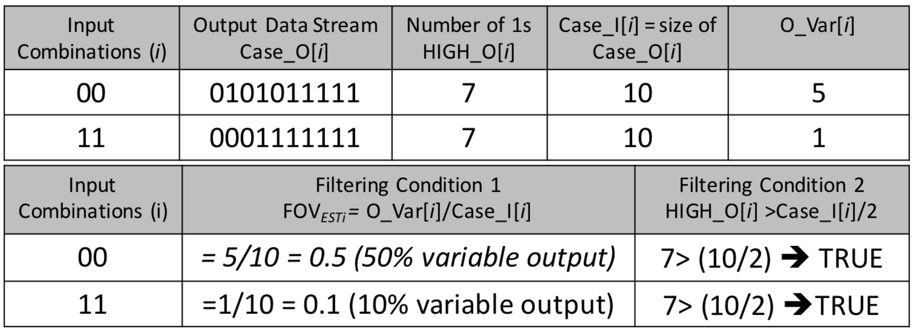
However, this filter alone is not sufficient to obtain the correct Boolean logic of a model. As in the case of the example shown in Figure 5.1, the algorithm considers obtaining the output high for both input combinations 00 and 11, based on the estimated value of FOV_{EST}, and end up obtaining the XNOR logic for this circuit model. Therefore, in order to handle this situation, another filter is applied according to equation 5.2, which checks if the number of 1s' in the output binary data stream, for the specific input combination, are greater than half the size of the whole output data stream.

$$HIGH\_O[i] \; = \frac{Case\_I[i]}{2} \tag{5.2}$$

where:

i	= Input combination at which the output stream is being checked.
$HIGH\_O[i]$	= Number of 1's in the output stream corresponding to the input combination i.
$Case\_I[i]$	= Number of times the input combination i occurs in the simulation data. This is equivalent to the length of corresponding output data stream.

For the example shown in Figure 5.1, this condition holds false for the input combination 00 ($3 \ngtr 1850/2$), but turns true for the input combination 11 (1875 $>3050/2$). This filter also helps in making sure that the output, for a specific input combination, is certain – either high or low. Nevertheless, this filtration technique may also produce wrong results if not applied together with the first technique mentioned above. In order to understand this, consider the example case shown in Figure 5.2, where the output binary data streams of two different input combinations, 00 and 11, are shown. The number of 1s in the output stream, for both the cases, is same; however, the patterns are different. That is, the output, for the input combination 00, remains high for the same number of times it is high for the input combination 11, but the output is highly oscillatory



Input Combinations (*i*)	Output Data Stream Case_O[*i*]	Number of 1s HIGH_O[*i*]	Case_I[*i*] = size of Case_O[*i*]	O_Var[*i*]
00	0101011111	7	10	5
11	0001111111	7	10	1

Input Combinations (i)	Filtering Condition 1 FOV_{ESTi}= O_Var[*i*]/Case_I[*i*]	Filtering Condition 2 HIGH_O[*i*] >Case_I[*i*]/2
00	*= 5/10 = 0.5 (50% variable output)*	7> (10/2) ➔ TRUE
11	=1/10 = 0.1 (10% variable output)	7> (10/2) ➔TRUE

Figure 5.2: Effectiveness of filtration process using both filters. An example showing how both filters are useful, when applied together, in obtaining the correct Boolean expression.

in the former case. The algorithm therefore discards (in this case if FOV$_{UD}$ $\leq$ 0.5) this unstable output and do not consider it while constructing the Boolean expression.

The pseudo code of this sub-procedure is shown in Algorithm 5.4. In order to filter out the results, both of the above mentioned conditions should be true. That is, the filtration is performed, if the estimated fraction of variation, FOV$_{EST}$, is less than the user-defined value (FOV$_{UD}$); and if the number of times the output is high for input case i is greater than half of the occurrence of input case i throughout the simulation, as shown in line 12 of Algorithm 5.4. For each filtered results, the Boolean expression is constructed through the pseudo code shown in the lines 13-26.

In the end, algorithm estimates the percentage fitness of estimated Boolean expression (PFoBE), in the simulation data, according to equation 5.3.

$$PFoBE = 100 \; - \; \frac{\sum_i FOV_{EST_i}}{nc} \; \times \; 100 \tag{5.3}$$

where:

i	= Input combination at which the filtered output stream is high.
FOV_{EST_i}	= Estimated fraction of variation in the output data stream for ith input combination.
nc	= Total number of input combinations.

Algorithm 5.4: Pseudo code of the procedure **ConsBoolExp**.

```
input  : $O\_Var$, $Case\_I$, $HIGH\_O$, $nc$, $N$, $FOV_{UD}$, $Case\_O$
output: $BoolExpres$, $PFoBE$
begin
    Set TV = 0; // Total variation for all input cases, nc
    Set $\text{FOV}_{EST}$ = 0; // $\text{FOV}_{EST}$ = Estimated fraction of variation
    Set Bin[$N$] = 0;
    // holds N-bit binary value of input case, nc
    Set Inter_Expr = null;
    Set Curr_Expr = null;
    Set BoolExpres = null;
    Set PFoBE = 0;
    // loop through all input cases nc
    for all $i \in nc$ do
        if ($HIGH\_O[i] \geq 1$) then
            // Estimating fraction of variation.
            $\text{FOV}_{EST}$ = O_Var[i]/Case_I[i];
            if (($FOV_{EST} < FOV_{UD}$) AND ($High[i] > Case\_I[i]/2$)) then
                Bin[$N$] = $\text{i}_d$;
                // loop through all input bits
                for all $j \in N$ do
                    if ($j^{th}$ bit of $Bin == 0$) then
                        // put a bar (') with the name of $j^{th}$ input
                        Curr_Expr = $\text{In}_j$';
                    else
                        // directly extract the name of $j^{th}$ input
                        Curr_Expr = $\text{In}_j$;
                    end
                    Inter_Expr = Inter_Expr · Curr_Expr;
                end
            else
                Inter_Expr = null;
            end
            BoolExpres = BoolExpres + Inter_Expr;
            Set Inter_Expr = null;
        end
        TV = $\text{FOV}_{EST}$ + TV;
    end
    PFoBE = 100 - (TV/nc × 100)
end
```

5.2 Experimentation by Simulation

The proposed algorithm for logic analysis is tested on the SBML models of 15 genetic circuits. This set of 15 circuits includes 1, 2 and 3-inputs genetic logic circuits, which are composed of 1 to 7 genetic logic gates containing 3-26 genetic components. The five genetic circuit models are obtained from [21] and the remaining 10 are the models of real genetic circuits acquired from [14]. There are a total 60 circuits published in [14], which were first designed on a tool, named *Cello* [14], with the help of a hardware description language for living cells. They were then fabricated and tested in a laboratory. Out of these 60 circuits, 45 of them worked correctly in the lab. Out of 45 working circuits, 10 of them are chosen randomly (with different level of complexity) for the experimentation in this work.

5.2.1 Analysis of the SBOL-SBML converted genetic circuit models

As stated in Chapter 4 (in Section 4.2.6), that *Cello* generates the SBOL files. Therefore, the SBOL files of the 10 circuits are first converted into SBML models using [61]. While analyzing the SBOL/SBML models of the circuits in iBioSim [17], it was noticed that the inputs B and C of all circuits are swapped in comparison to their original circuit diagrams shown in [14]. For example, consider Figure 5.3 in which the original circuit schematic, the SBOLv diagram, truth table, and the converted SBML model of the genetic circuit 0x0B, obtained from [14], are shown in Figure 5.3 (a), (b), (c) and (d), respectively. In Figure 5.3(a) and (b), the inputs A, B, and C, corresponds to P_{Tac}, P_{Tet} and P_{Bad} in Figure 5.3(c) and (d), respectively. In Figure 5.3(b), the solid black distributions are experimental data; and blue and red line distributions are computational predictions from Cello [14], which describes the logic states 1 and 0, respectively. The logic states, for example + / + / −, indicates that the inputs C, B and A (in order) are in logic 1 / 1 / 0 states, respectively.

As mentioned before in the Section 4.2.6 that the SBOL file generated by Cello does not include the input sensor block for the circuit (which includes the input inducers); thus these inducers are also not included during the SBOL-SBML conversion process. Hence, the input inducers are manually added in the SBML model using iBioSim, as shown in Figure 5.3(d) with the red-dotted block. The rest of the model (shown as blue-dotted block) is the result of the SBOL-SBML conversion process [61]. The external inputs, in all these ten circuits, are the inducers IPTG, aTc and Arabinose, which control the activities of the promoters P_{Tac}, P_{Tet} and P_{Bad}, respectively, as shown in Figure 5.3(d).

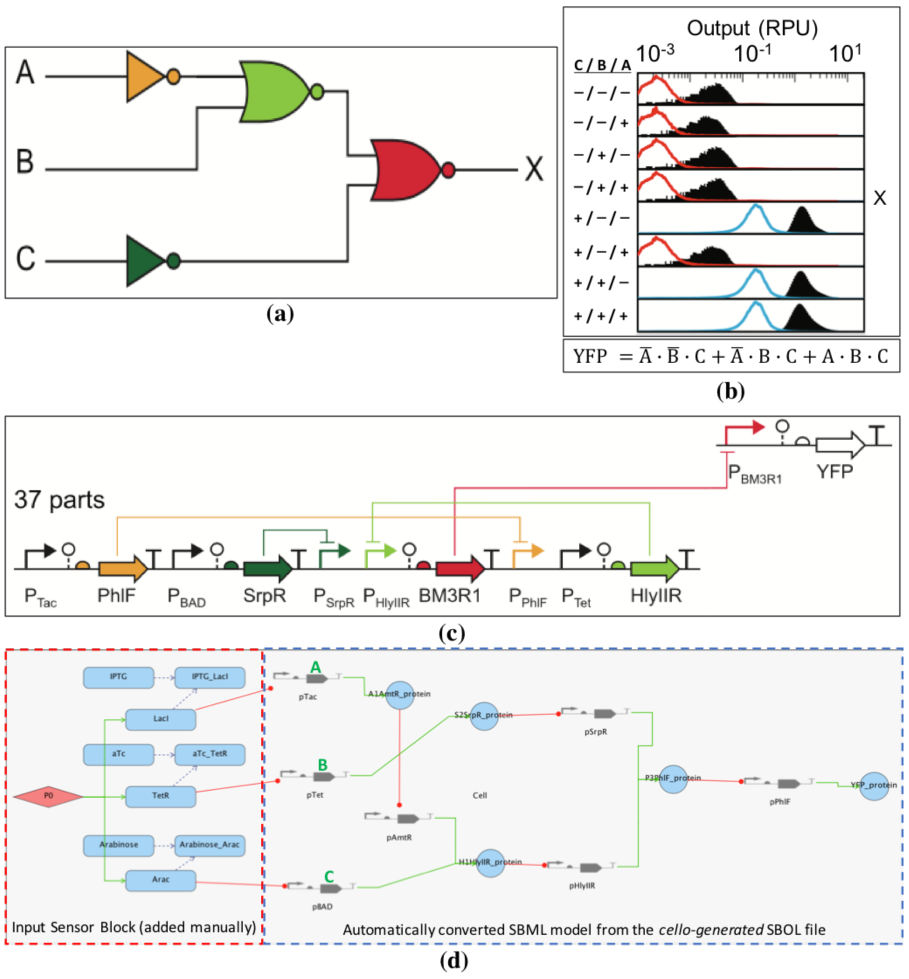


Figure 5.3: Genetic circuit 0x0B [14]. (a) Circuit schematic (b) Truth table and (c) SBOLv diagram [Image courtesy of [14]]. (d) Screen-shot of the auto-generated SBML model in [17] using SBOL-SBML converter [61].

It is also important to mention that, in [14], a circuit is generated by a random search of compatible genetic gates using the simulated annealing algorithm [56]. Since the circuit is created using the non-deterministic search, the solution may be different every time the process is executed. This is why the genetic components shown in Figure 5.3(c) are different from those depicted in Figure 5.3(d). Despite of having different genetic components, the circuit structure should not be changed in order to achieve the same functionality of each circuit. In Figure 5.3(a), the input A (or P_{Tac}) is connected to a NOT gate that produces the *PhlF* protein, which in turn suppresses the output promoter P_{PhlF}, as shown in Figure 5.3(c). The input B (or P_{Tet}) is connected directly to one of the inputs of the NOR gate (producing *HlYllR* protein), as depicted in Figure 5.3(a) and (c). Another input of this NOR gate is the output promoter of the first NOT gate (i.e. P_{PhlF}), which together with the input B (or P_{Tet}) generates the protein *HlYllR* and suppresses the output promoter, P_{HlYllR}, as shown in Figure 5.3(c). Similarly, the input C (or P_{Bad}) in Figure 5.3(a) is connected to a separate NOT gate (producing *SrpR* protein) which suppresses the output promoter P_{SrpR}. The promoter P_{SrpR} together with the output promoter P_{HlYllR} of the first NOR gate (producing *HlYllR* protein) performs the NOR logic to produce the protein *BM3R1*, which in turn produces the final output, yellow fluroescent protein (YFP).

In Figure 5.3(d), the NOT gate of input A is producing the protein *A1_ AmtR*, as opposed to what it is shown to produce (*PhlF* protein) in Figure 5.3(a) and (c). This is because of the non-deterministic gates assignment in Cello, which may have different output proteins, but the functionality (the NOT logic in this case) remains same. The generated protein, *A1_ AmtR*, suppresses the promoter, P_{AmtR}, thus exhibiting the NOT logic. However, it can be noticed, in Figure 5.3(d), that instead of input B (or P_{Tet}), the input C (or P_{Bad}) is directly connected to one of the inputs of the NOR gate (producing *HlYllR* protein) along with the promoter P_{AmtR}. The input B (or P_{Tet}) in Figure 5.3(d), which should be connected directly to a NOR gate, seems to be routed to a NOT gate (producing *SrpR* protein). In other words, the inputs B and C of the original circuit 0x0B are swapped. Due to this problem of swapped B and C inputs, the functionality of any genetic circuit shown in [14] would be changed. For example, the optimized Boolean expression of the original circuit 0x0B is $(\overline{\overline{C} + \overline{\overline{A} + B}})$, in which there is a NOT gate with input C and an intermediate NOR gate with inputs $\overline{A}$ and B. If the inputs B and C are swapped then the expression will become $(\overline{\overline{B} + \overline{\overline{A} + C}})$, having NOT gate with input B and an intermediate NOR gate with inputs $\overline{A}$ and C, which clearly changes the functionality of the circuit.

The same behavior, that is the swapping of inputs B and C, has been observed in all ten circuits obtained from [14]. It has also been observed that the internal structure of some of the circuits are changed in the auto-converted SBML files (See Appendix C.2). We verified that the problem is neither in [61] nor in [17], so we assume that the diagrams shown in [14] are structurally correct and the problem might be in the SBOL file generation in the Cello tool [14].

5.2.2 Logic analysis and verification

The SBML files generated for ten circuits [14] (with inputs B and C swapped) and for five other circuits [21] are used in D-VASim to perform the experimentation followed by the logic verification. The external inputs in the circuits obtained from [14] are IPTG, aTc and Arabinose, which were varied in the experimentation to observe their logical behaviours. In these experiments, all fifteen circuit are run for 10,000 simulation time units. Also, a value of 1000 time units is assumed to be a propagation delay for all circuits, which means that during simulation, each input combination is applied for at least 1000 time units. Furthermore, the upper and lower threshold value equals to 15 and 0 molecules, respectively, are used for all fifteen circuits.

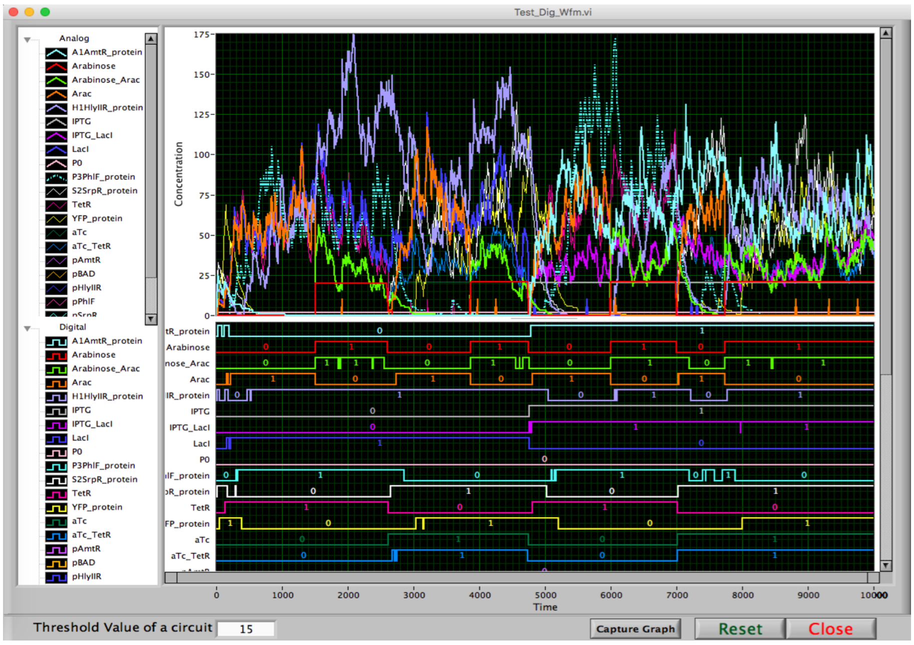
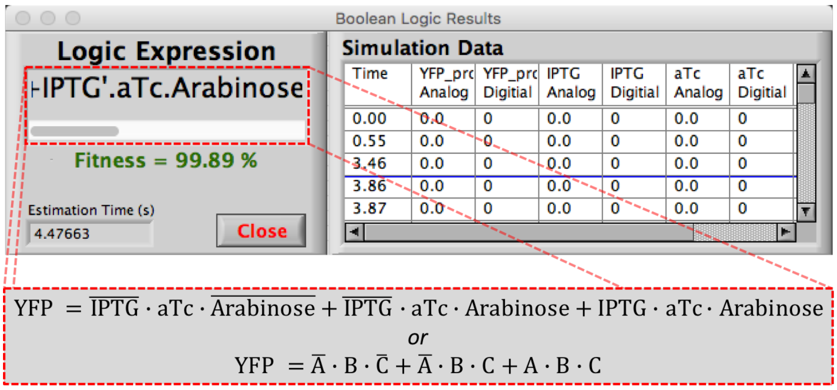


Figure 5.4: Interactive simulation traces of 0x0B, with its corresponding digital waveforms, for logic analysis.

The screen-shot of the interactive experimentation of genetic circuit, 0x0B, is shown in Figure 5.4. The upper half of this figure indicates the interactive analog simulation results in which the concentration of external inputs, *IPTG, aTc* and *Arabinose* is varied to observe the behavior of the output protein, *YFP*. It is evident from this figure that it is not easy to grasp the logic of a genetic circuit model with these messy analog waveforms. On the other hand, the Boolean logic is a bit easier to analyze in the corresponding digital waveforms shown in Figure 5.4. The logic verification algorithm made this analyses further easier by automatically analyzing the whole simulation data (shown in Figure 5.4) and indicating the results in the form of a raw Boolean expression as shown in Figure 5.5.



+IPTG'.aTc.Arabinose

Fitness = 99.89 %

Estimation Time (s)

4.47663

Time	YFP_prc Analog	YFP_prc Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
0.55	0.0	0	0.0	0	0.0	0
3.46	0.0	0	0.0	0	0.0	0
3.86	0.0	0	0.0	0	0.0	0
3.87	0.0	0	0.0	0	0.0	0

$$\text{YFP} = \overline{\text{IPTG}} \cdot \text{aTc} \cdot \overline{\text{Arabinose}} + \overline{\text{IPTG}} \cdot \text{aTc} \cdot \text{Arabinose} + \text{IPTG} \cdot \text{aTc} \cdot \text{Arabinose}$$

or

$$\text{YFP} = \overline{\text{A}} \cdot \text{B} \cdot \overline{\text{C}} + \overline{\text{A}} \cdot \text{B} \cdot \text{C} + \text{A} \cdot \text{B} \cdot \text{C}$$

Figure 5.5: Logic analysis results of the genetic circuit 0x0B in D-VASim.

The percentage fitness of the Boolean expression in the simulation data is also shown in Figure 5.5. The logic verification algorithm expresses the logical behavior in the *Sum of Products (SoP)* form of Boolean expression (See Chapter 6 for more details). The original Boolean expression of the circuit 0x0B, obtained from the truth table shown in Figure 5.3(b), is $\overline{A}.\overline{B}.C + \overline{A}.B.C + A.B.C$. However, the Boolean expression for the same circuit, obtained in these experimentation, is shown in Figure 5.5, which is equivalent to $\overline{A}.B.\overline{C} + \overline{A}.B.C + A.B.C$; where inputs A, B and C corresponds to IPTG, aTc, and Arabinose, respectively. This is because the inputs B and C were swapped in the SBML model (see Section 5.2.1). This indicates that the proposed logic analysis algorithm estimates the correct logic of the SBML model being tested, provided that the experimentation is performed with the correct propagation delay and threshold values.

The simulation data and logic analyses of the circuits, 0x0B, are shown in Figure 5.6. The results shown in Figure 5.6 are used to obtain the logical behavior of the circuit 0x0B. In Figure 5.6, *Case_ I* indicates the number of times each input combination occurs during the total 10,000 time units of simulation. It further includes the number of times the output of a circuit remains high, *High_ O*, for that particular input combination along with the number of variations in the output data, *O_ Var*. Also, the input combinations, at which the circuit's

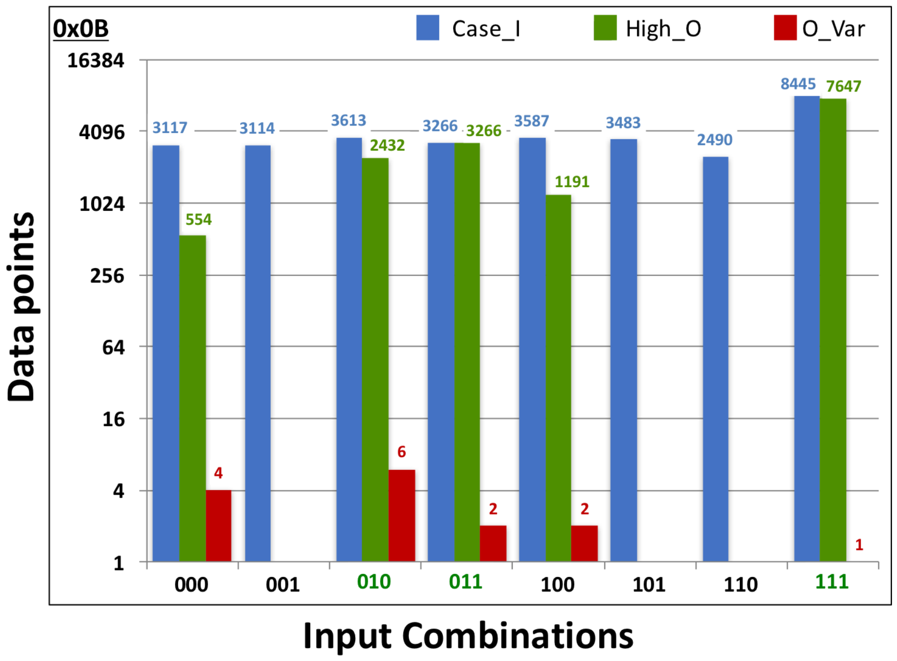


Figure 5.6: Analytical simulation data of the genetic circuit model 0x0B [14].

output is expected to be high, are highlighted in green color along the x-axis.

In Figure 5.6, the output variation of circuit 0x0B is not too high. For example, the output state appears to be logic-1 for the input combination 100 and seems quite stable having very low variation value of 2. The reason why the input combination 100 has so many logic-1 output states is due to the fact that the output is high for the previous input combination 011. When the input combination is changed from 011 to 100, the output starts to decay gradually, and remains high until it passes by the threshold level. This input combination should, therefore, be included in the Boolean expression, but however filtered out using equation 5.2, because for 3587 times of the input combination 100 occurs during the entire simulation, the corresponding output remains high for 1191 times ($< 3587/2$).

It is therefore obvious that similar to electronic circuits, where the output state may be incorrect if the inputs are changed before the propagation delay has elapsed, the correct behavior of a genetic circuit can only be obtained when each possible input combination is applied for sufficient amount of time. The experimental results of the remaining 14 circuits are given in Appendix C, which shows that the algorithm successfully obtained the correct boolean expressions for all these circuits. It has been clearly observed that not only the inputs B and C are swapped in all of the circuits obtained from [14] but also the circuit topology in the converted SBML models of the circuits 0x04, 0x4D, and 0x1C are different than those originally shown in [14]. However, these structural changes in the circuits do not affect the functionality of these circuits except that, similar to other circuits, the inputs B and C are interchanged.

5.2.3 Effects of varying threshold value on the behavior of genetic circuit

The behavior of genetic circuits is also analyzed by varying the threshold value of input concentrations to very low (3 molecules) and very high (40 molecules), and observed that the same circuits behave differently.

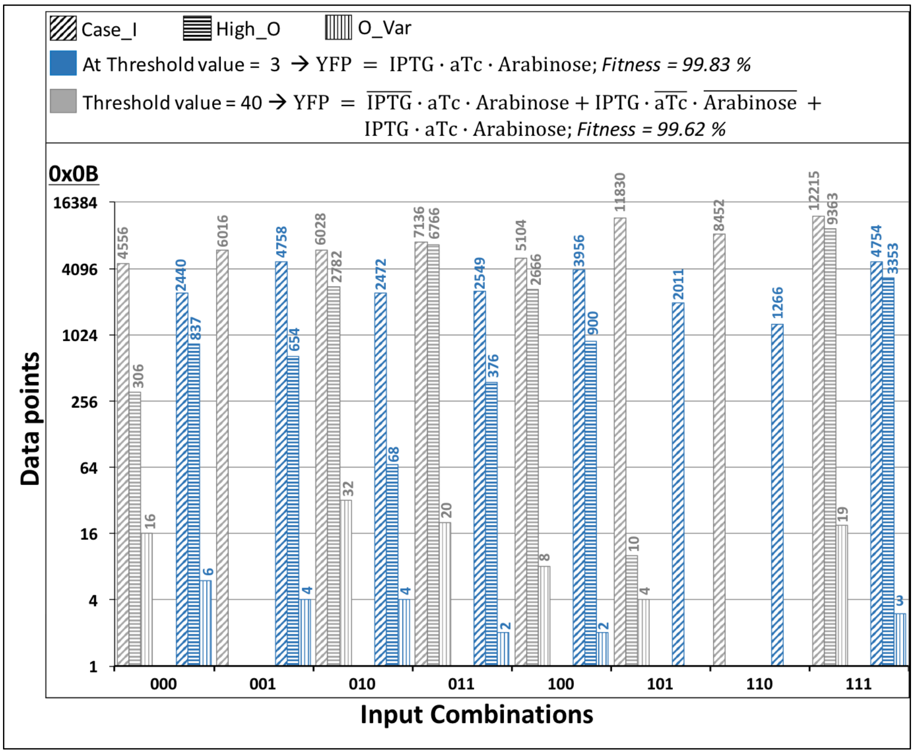


Figure 5.7: Analytical data of circuit 0x0B for threshold values 3 and 40.

Figure 5.7 shows the comparison of simulation data for the circuit 0x0B for the above mentioned two threshold values. In this figure, it can be noticed that the output response of 0x0B circuit, for upper threshold value of 3 molecules, is entirely different and it behaves like a 3-input AND gate. This is because the applied input concentration is too weak to trigger the output concentration; but when applied together i.e., 111, the output is triggered high to satisfy the applied filters. On the other hand, 0x0B circuit has two wrong states (shown in the Boolean expression in Figure 5.7) when 40 molecules are applied as an input concentration. For this case of threshold value, the output response also seems to oscillate between logic-high and low for large number of times (Figure 5.7) as compared to when its threshold value is set to 15 (Figure 5.6). This is because the concentration levels of input and output species are not clearly distinguishable when the applied input concentration is high.

5.2.4 Performance analysis

The performances of the proposed algorithm, on all fifteen circuits, are also analyzed, as shown in Figure 5.8. Depending upon the complexity of a circuit, each circuit may have different amount of data for specified simulation time.

For example, there is only one gate, composed of three genetic components, in genetic NOT gate circuit. For this small circuit, only 3.5×10^3 reactions occurred during 10,000 simulation time units. In contrast, a bigger genetic circuit (x1C) containing 7 cascaded gates, composed of 26 genetic components, had 43.1×10^3 reactions executed during the same simulation time. Therefore, the size of simulation data for NOT gate circuit is much smaller than the size of x1C circuit.

Due to this, the time to estimate the logic of a circuit is much lower for a NOT gate circuit as compared to the time required to estimate the logical behavior of the x1C circuit. This is shown as *Data points* in Figure 5.8, which corresponds to the number of reactions executed during 10,000 simulation time units. Similarly, the difference of analysis time for the circuits having same number of gates reflects the difference in the number of genetic components, and thus in the number of *Data points*.

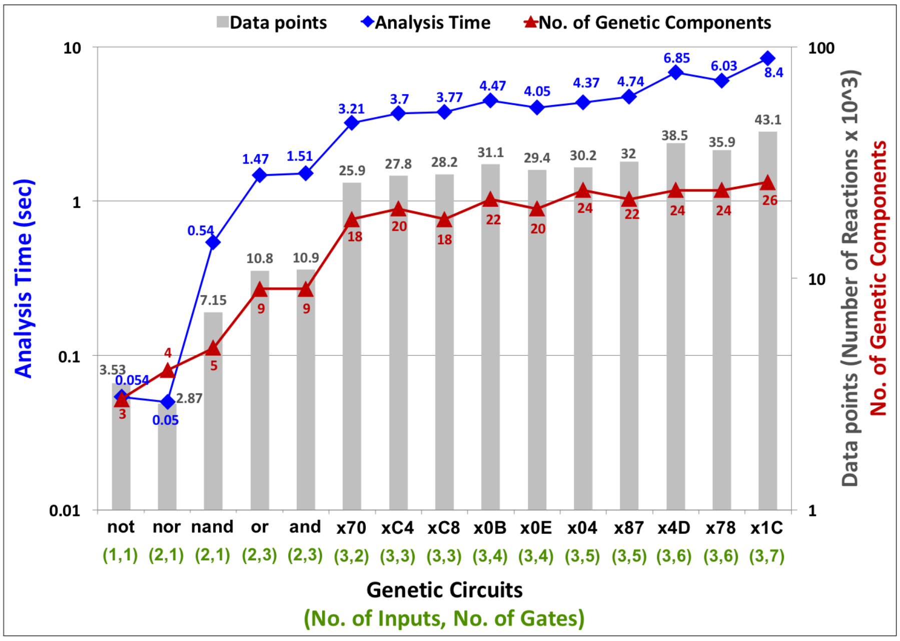


Figure 5.8: Performance evaluation of the proposed algorithm over 15 circuits. First five circuits are from [21] and the remaining ten circuits are from [14].

5.3 Discussion

In this chapter, a methodology to analyze and verify the intended behavior of a genetic logic circuit is presented. It is shown through simulation experiments that the circuit may not behave as intended if the circuit parameter(s), like threshold value, are varied. This may help users to analyze the circuit's behavior and robustness for different parameter sets before creating them in the laboratory. Furthermore, the performance of the proposed algorithm is analyzed over the number of genetic circuit models, and observed that it takes about 8.4 seconds to analyze the logic of a complex genetic circuit with significantly large-sized data. As the experimentation in the laboratory requires a couple of hours to analyze even a single output state [14], the proposed simulation-based approach is likely to be useful for genetic circuit designers to analyze the intended logic of genetic circuits prior to their implementation and testing in the laboratory. The proposed algorithm is scalable and can be used to analyze genetic circuit with any number of inputs. However, when the circuit inputs are increased, the size of simulation data will also be increased, which results in the rise of logic estimation time. However, in comparison to electronic circuits, the genetic circuits are much more complex and difficult to construct due to their stochastic nature. This fact suggests that the size of genetic circuits may not grow with a same pace as the size and complexity of electronic circuits were increased, and thus the proposed method can be used effectively for genetic logic analysis.

Chapter 6

GeneTech – A *Tech*nology Mapping Tool for *Gene*tic Circuits

In contrast to electronic logic gates, which have the same physical quantity i.e., voltage, at the input and output, the genetic logic gates have different quantities acting as an input and output. This makes it very challenging to integrate genetic logic gates to construct complex genetic circuits because the triggering molecules relating input and output, between cascaded gates, has to be compatible and unique.

In this chapter, a new standalone tool, *GeneTech* (extracted from *Gene*tic *Tech*nology mapping) is discussed, which automate the process of generating genetic circuits for dedicated Boolean functions. *GeneTech* performs Boolean optimization, followed by synthesis and technology mapping using a library of genetic logic gates. The genetic logic gates library used in this work has been developed and tested in the laboratory by MIT and Boston University [14]. *GeneTech* takes the Boolean expression of a genetic circuit as input, and then first optimize it. Afterwards, it synthesizes the optimized Boolean expression into NOR-NOT form in order to construct the circuit using the real NOR/NOT gates available in the genetic gates library [14]. In the end, *GeneTech* performs technology mapping to generate all the feasible circuits, with different genetic gates, to achieve the desired logical behavior.

The work presented in this chapter has been disseminated in a workshop and the full manuscript has been submitted in journal, as mentioned below.

[51] Hasan Baig and Jan Madsen, "A Top-down Approach to Genetic Circuit Synthesis and Optimized Technology Mapping", *9th International Workshop on Bio-Design Automation (IWBDA)*, pp. 28–29, 2017. *Published.*

[63] Hasan Baig and Jan Madsen, "*GeneTech*: "A Technology Mapping Tool for Genetic Logic Circuits", *IEEE Transactions on Biomedical Engineering. Under review.*

6.1 Motivation

This work is originally inspired from the processes of optimization and technology mapping of electronic circuits in the EDA industry. In EDA for digital electronics, the combinatorial circuit optimization is always required to implement the circuit with the minimum number of logic gates [64]. This area-efficient implementation of digital circuits not only helps reducing the size of electronic devices but also avoid wasting power and redundant resources.

In order to get the insight of logic optimization, consider the digital circuit for the expression – $ab+b+ac$, in Figure 6.1(a). This figure shows that the original circuit contains four logic gates. After running the optimization algorithm, the number of gates in the circuit reduces down to two while preserving the actual functionality, as illustrated in Figure 6.1(b).

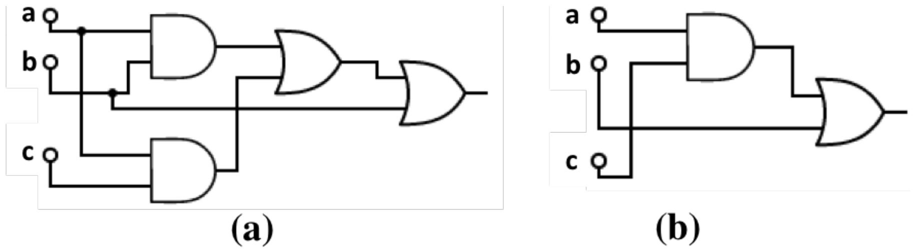


Figure 6.1: Digital circuit of the expression $ab + b + ac$. (a) Original circuit containing four gates. (b) Optimized circuit having two gates.

The 2-input genetic NAND gate is termed universal in [21] because it is possible to construct other combinational gates by cascading the collection of them. For instance, genetic inverter and a 2-input genetic NAND gate can be used to construct a genetic XOR gate. Suppose L and T represent the two genetic inputs, LacI and TetR, then the function of a genetic XOR gate can be described by

$$L \oplus T \ = \overline{L}T \ + \ L\overline{T} \tag{6.1}$$

The standard schematic of a XOR gate is shown in Figure 6.2. It consists of AND, OR and NOT gates. The direct genetic implementation of AND and OR gates are not shown in [21]. However, as mentioned above, these gates can be constructed with the help of universal 2-input genetic NAND gates and genetic inverters [21].

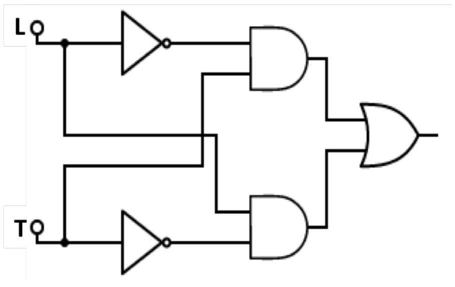


Figure 6.2: Standard schematic of the XOR gate.

The equivalent circuit of a XOR gate constructed with the available genetic components is shown in Figure 6.3(a). This schematic of a genetic XOR gate needs to be optimized with the following two constraints — the functionality of a circuit should remain same, and the components of the optimized circuit must be available in the library of genetic gates. It can be noticed that the back-to-back inverters are present in Figure 6.3(a) resulting in the following equivalent Boolean expression.

$$L \oplus T \ = \overline{\overline{\overline{\overline{\overline{L}T}}} \ + \ \overline{\overline{\overline{L\overline{T}}}}} \tag{6.2}$$

One of the methods to estimate the cost of a circuit is to calculate the number of inputs to each gate – hence the cost is higher if the number of gates is increased. For the circuit shown in Figure 6.3(a), the cost of three NAND gates is 6 and six

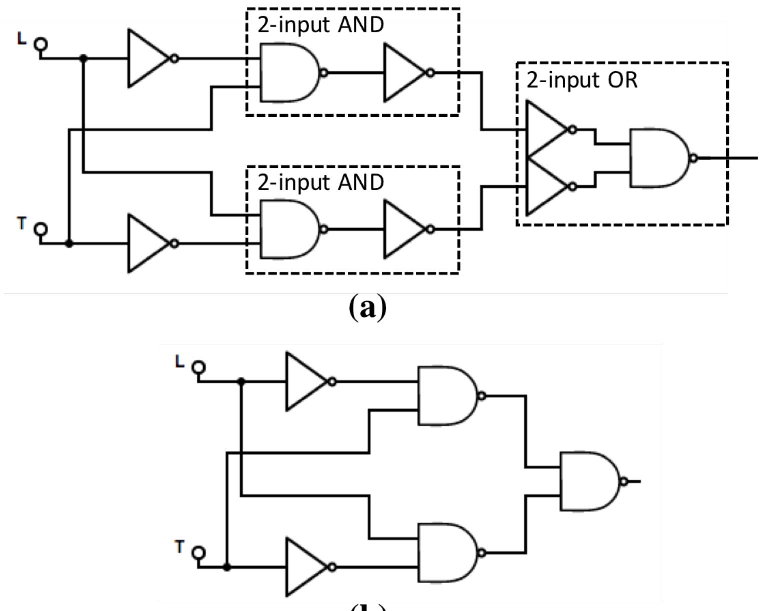


Figure 6.3: Schematic of the genetic XOR gate (a) constructed with 2-input NAND gates. (b) The optimized circuit.

NOT gates is also 6, so the total cost is 12, which can be reduced down to 8 as shown in Figure 6.3(b). This optimization seems straight forward and acceptable as it fulfills the above mentioned constraints and is composed of inverters and 2-input NAND gates, which are available in the genetic-gates library [21]. However, the optimization and technology mapping of genetic circuits is not similar to electronic circuits. This is because the input and output quantities of electronic circuits are the same i.e. voltage, and therefore the electronic gates can easily be cascaded together. On the contrary, the input and output quantities of genetic gates are different, and therefore the signal matching has to be considered while mapping genetic gates on the circuit.

Similar to the digital electronic circuits, we want to avoid having redundant logic in genetic circuits. Therefore, the logic expression, either for digital or genetic circuits, needs to be minimized using any optimization technique.

6.2 Methodology

A digital circuit or Boolean logic can be expressed either in the minterm or maxterm canonical form. Minterms are called products because they are the logical AND of a set of variables/literals and maxterms are termed as sums because they are the logical OR of a set of variables/literals. Therefore, the Boolean expression can either be expressed as sum of minterms/products (SOP) or product of maxterms/sums (POS). In the example shown in equation 6.3, the left-hand side represents the SOP form and the right-hand side represents its equivalent POS form.

$$ab + b + ac = (a + b + c)(a + b + \overline{c})(\overline{a} + b + c) \tag{6.3}$$

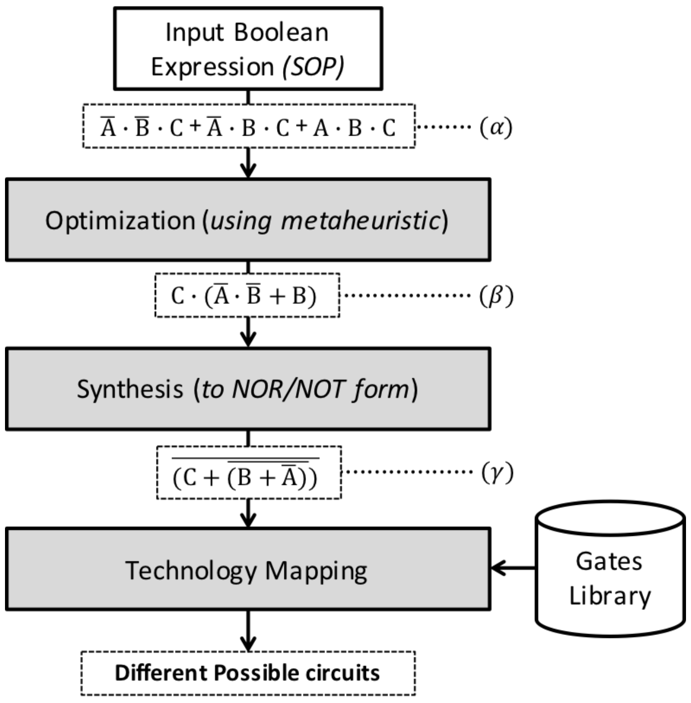

Input Boolean Expression (SOP)
$\overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C + A \cdot B \cdot C$
(α)
Optimization (using metaheuristic)
$C \cdot (\overline{A} \cdot \overline{B} + B)$
(β)
Synthesis (to NOR/NOT form)
$\overline{(C + \overline{(B + \overline{A})})}$
(γ)
Technology Mapping
Gates Library
Different Possible circuits

Figure 6.4: The technology mapping flow of *GeneTech*.

GeneTech is able to process the Boolean expressions available in SOP form. The design flow of *GeneTech* is shown in Figure 6.4. It takes the SOP Boolean expression and first optimizes it followed by the synthesis and technology mapping. Each of the highlighted steps shown in Figure 6.4 are described separately in the following subsections.

6.2.1 Logic optimization

The digital logic minimization using meta-heuristics is a classical optimization problem and has already been addressed before [65]. As demonstrated above, the minimization of genetic logic circuits follows the same procedure as that of digital circuits. Therefore, meta-heuristics can also be employed to optimize genetic logic circuits. The algorithm used in *GeneTech* for Boolean expression optimization is based on the simulated annealing algorithm [66]. In order to apply simulated annealing to any optimization problem, one must define the search space, the neighbor selection method, acceptance probability, and the annealing schedule. The initial temperature, the rate at which the temperature reduces, the number of iterations at each temperature and the stopping criterion is known as the annealing schedule [65].

Suppose, **B**, be a search space, is a set of all possible Boolean expressions implementing a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}^m$. During each iteration, the annealing algorithm considers some neighboring state expression, $E_N \in \mathbf{B}$ of the current state expression $E_C \in \mathbf{B}$, and decides probabilistically either to move to state E_N or remain in the state E_C. The pseudo-code of annealing-based algorithm for Boolean expression minimization is shown in the Algorithm 6.1. The objective function for the minimization of the Boolean expression is

$$Minimize \sum L$$

where, L is the total number of literals in the expression.

That is, the objective is to minimize the cost of a Boolean expression in terms of literals. This minimization is carried out by the help of Boolean replacement rules listed in Table 6.1.

The algorithm starts by taking the inputs - Boolean expression, temperature coefficient, initial temperature, and the time to execute algorithm. The procedure *COST_ CAL* calculates the initial cost of the Boolean expression (line 2). The while loop runs until the end time, t_E, lapsed (line 5). The procedure, *SEARCH-NEIGHBOUR* (line 6), obtains the neighboring state expression, E_N, by applying the Boolean replacement rules listed in Table 6.1.

Algorithm 6.1: Pseudo code of *logic optimization.*

```
input  : $E_0$, $T_{COF}$, $T_0$, $t_E$
output: $E_B$, $C_B$
// $E_0$ = Initial expression, $T_{COF}$ = Temperature coefficient
// $T_0$ = Initial temperature, $t_E$ = End time to finish algorithm,
// $E_B$ = Best expression, $C_B$ = Best cost
begin
    $C_0$ = COST_CAL($E_0$); // $C_0$ = Initial cost
    $E_C = E_0$; // $E_C$ = Current expression
    $T_C = T_0$; // $T_C$ = Current temperature
    while ($t_C < t_e$) do
        // $t_C$ = Current time
        ($E_N$, $C_P$, $C_N$) = SEARCH-NEIGHBOUR ($E_C$);
        // $E_N$ = Neighbouring expression, $C_P$ = Previous cost, $C_N$ = New
            cost
        // * Calculate acceptance probability (AP) *
        if ($C_N > C_P$) then
            AP = CALC($T_C$, $C_P$, $C_N$);
        else
            AP = 1;
        end
        // * Update new expression *
        if (AP > random[0,1)) then
            $E_C = E_N$;
        else
            $E_C = E_C$;
        end
        $T_C = T_{COF} \times T_C$;
    end
    $E_B = E_C$;
    $C_B = C_N$;
end
```

The annealing algorithm calls the procedure, *SEARCH-NEIGHBOUR*, by passing the current state of expression as an input. It then randomly decides whether to expand any of the min-term in the expression or proceeds without expanding it. The re-expansion of already reduced min-terms is sometimes useful to avoid getting stuck in local optimum value. The total number of min-terms in the Boolean expression is then calculated and a combination of any two min-terms is selected randomly for minimization. The number of literals in each min-term is then evaluated. If both of the min-terms contain single literal only, they directly scan through the replacement rules followed by the construction of a new expression. If the number of literals, in any of them, is greater than one, then

Table 6.1: Boolean replacement rules.

Case	Replacement
x + x (x.1) or (1.x) (x + 0) or (0 + x)	x
x' + x' (x'.1) or (1.x') (x' + 0) or (0 + x')	x'
(x + 1) or (1 + x) (x' + 1) or (1 + x') (x + x') or (x' + x)	1
(x.0) or (0.x) (x.x') or (x'.x)	0
x.y + x.z	x.(y + z)

the algorithm performs a check if a common literal(s) is present in both of them. If no match is found, the input expression is reconstructed. If a common literal is found, the algorithm arranges the min-terms in the form

$$\mathrm{M}(R_{L-1} + R_{L-2})$$

where:
M is the matched literal(s).
R_{L-1} corresponds to rest of the literals in the first min-term.
R_{L-2} corresponds to rest of the literals in the second min-term.

The procedure, SEARCH-NEIGHBOUR, then checks the nested elements (elements inside braces) of the reduced expression for further reduction, and passes them through the Boolean replacement rules. The process then searches for the nested min-terms containing braces and keep searching until all the nested min-terms, containing braces, are passed through the Boolean replacement rules. Finally, all the possible combinations, within nested expression, are checked for common literals. Unlike the beginning of the algorithm, where the combination of two min-terms is randomly chosen, all the combinations are checked one by one at this stage. The new expression, E_N, is then constructed and a new cost of a reduced expression, C_N, is calculated.

As an example, consider the same SOP form of the expression 6.3, which is rewritten as expression 6.4 below

$$ab \ + \ b \ + \ ac \tag{6.4}$$

To minimize the expression 6.4, the algorithm randomly chooses the combination of, say, first and third min-terms initially, i.e. *ab* and *ac.* As literal *a* is common in both of them, the expression reduces down to the expression 6.5.

$$a(b\ +\ c)\ +\ b \tag{6.5}$$

Since these two min-terms are selected randomly, it is possible that this combination may not produce the optimized result. Because of this uncertainty, the capability of expanding the reduced min-terms is incorporated in the algorithm. The reduction of expression (6.4) to (6.5) results in the reduction of cost from 5 to 4 literals with no further reduction possible. The circuit generated for this expression, is shown in Figure 6.5., which is not the optimized one. With the capability of expanding the reduced min-terms, the algorithm is able to come out of this local optimized solution. Therefore, in the above example, if the expression (6.5) is expanded again into (6.4) and a combination of first and second elements is chosen, i.e., *ab* and *b*, then the optimized cost of the expression (6.4) would reduce from 5 literals to 3 literals, as shown below; and would result in the circuit pictured in Figure 6.1(b).

$$b(a\ +\ 1)\ +\ acb\ +\ ac \tag{6.6}$$

$$b\ +\ ac \tag{6.7}$$

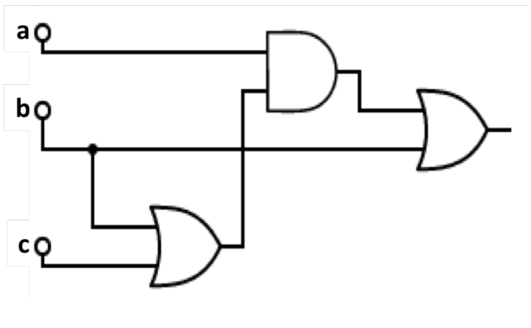


Figure 6.5: The local optimized circuit of an example expression 6.4.

Finally, the procedure *CALC* calculates the acceptance probability based on the cost of Boolean expressions (lines 7-11). If the new cost, C_N, is less than the previous cost, C_P, the new solution is assigned a probability equals to one and accepts the new expression as a current best solution. Otherwise, the acceptance probability of new expression is calculated according to the following expression:

$$AP = e^{\frac{-(C_N - C_P)}{T_C}} \tag{6.8}$$

Then a random number between 0 and 1 (exclusive) is generated. The algorithm accepts the new expression as a best solution if the acceptance probability, AP, is greater than the generated random number. Else it is rejected and the previous state of expression is considered as a current best solution (lines 12-16).

6.2.2 Logic synthesis

GeneTech is developed to be used for constructing real genetic circuits from the genetic gates library tested in the laboratory [14]. The circuits experimented in [14] are composed of genetic NOR and genetic NOT gates. Therefore, in this stage of logic synthesis, the AND/NAND terms present in the optimized Boolean expression are converted to NOT/NOR form by applying the following DeMorgan's Laws.

$$\overline{AB} = \overline{A} + \overline{B} \tag{6.9a}$$

$$AB = \overline{\overline{A} + \overline{B}} \tag{6.9b}$$

$$\overline{A}.\overline{B} = \overline{A + B} \tag{6.10}$$

The pseudo code of the logic synthesis algorithm is shown in the Algorithm 6.2. The algorithm is briefly explained with the help of example shown in Figure 6.6, where the input expression is shown in red and the final output expression is highlighted in green. The algorithm begins by taking the optimized Boolean expression as an input and checking first if it contains any minterms with braces (line 2 in Algorithm 6.2). All minterms inside each braced-minterm are first passed through the procedure, *ProcessANDTerms*, to convert ANDed minterms to NOR terms (lines 4, 5 in Algorithm 6.2). These steps are shown in Figure 6.6 in line 0 and 1. The algorithm then process OR terms, if the number of minterms inside braces are greater than one (lines 6-9 in Algorithm 6.2). This is shown in line 2 in Figure 6.6, where the OR terms are converted to NAND by the procedure ProcessORTerms, using equation (6.9a). The same procedure, also expand braces by multiplying minterms and check if any Boolean replacement rules (shown in Table 6.1) are applicable. This is shown in lines 3 and 4 in Figure 6.6.

Algorithm 6.2: Pseudo code of *logic synthesis*.

```
   input  : $Exp_{in}$
   output: $Exp_{out}$
   // $Exp_{in}$ = Input expression, $Exp_{out}$ = Output Expression
 1 begin
 2     if ($Exp_{in}$ contain braces terms) then
 3         for (all braces terms in $Exp_{in}$) do
 4             for (all minterms inside braces) do
 5                 MTIB = ProcessANDTerms;
                   // MTIB = Array holding minterms inside braces
 6                 if ($MTIB > 1$) then
 7                     OutputString = ProcessORTerms;
 8                     OutputString = ProcessNANDTerms;
 9                     OutputString = MintermsToExpression;
10                 else
11                     OutputString = MintermsToExpression;
12                 end
13             end
14         end
15     else
16         for (all minterms in $Exp_{in}$) do
17             Minterms = ProcessANDTerms;
18             OutputString = MintermsToExpression;
19         end
20     end
21     for (all minterms in OutputString) do
22         ConvertToNOR;
23     end
24     $Exp_{out}$ = OutputString;
25 end
```

After applying equation (6.9a) (in line 2 in Figure 6.6) to convert the expression from OR to NAND, it is converted back again to NAND form only if the expression is manipulated by applying the Boolean replacement rules. This NAND to OR conversion is shown in the lines 4-6 in Figure 6.6, using the procedure, *ProcessNANDTerms*, (line 8 in Algorithm 6.2). Then all of the minterms arranged in an array are converted to string expressions using the procedure, *MintermsToExpression* (line 9 in Algorithm 6.2). If there are no minterms in the expression

0	$\bar{c}\,.\,(ab + \bar{b})$	
1	$\bar{c}\,.\,(\overline{\bar{a} + \bar{b}} + \bar{b})$	$\because$ *(6.9b)*
2	$\bar{c}\,.\,\overline{\left((\bar{a} + \bar{b}).b\right)}$	$\because$ *(6.9a)*
3	$\bar{c}\,.\,\overline{(\bar{a}b + \bar{b}b)}$	$\because$ *braces multiplication*
4	$\bar{c}\,.\,\overline{(\bar{a}b)}$	$\because \bar{b}b = 0$
5	$\bar{c}\,.\,(\bar{\bar{a}} + \bar{b})$	$\because$ *(6.9a)*
6	$\bar{c}\,.\,(a + \bar{b})$	$\because \bar{\bar{a}} = a$
7	$\bar{c}\,.\,\overline{\overline{(a + \bar{b})}}$	$\because \bar{\bar{a}} = a$
8	$\overline{(c + \overline{(a + \bar{b})})}$	$\because$ *(6.10)*

Figure 6.6: Example expression to illustrate how synthesis algorithm works.

which contain braces (e.g. $a + bc$), the algorithm then processes the ANDed minterms in the expression (lines 16-19 in Algorithm 6.2). After processing all minterms with or without braces, the algorithm then converts all minterms in the expression to NOR (lines 21-22 in Algorithm 6.2). In the example shown in Figure 6.6, there is only one minterm in the expression with braces, which contain two sub minterms inside braces. In the last step, equation 6.10 converts the ANDed minterms (with braces) to NOR, as shown in lines 7 and 8 of Figure 6.6.

6.2.3 Genetic technology mapping

When the Boolean expression is synthesized into NOR/NOT form, genetic technology mapping can be performed using the genetic gates library. We have extracted the genetic gates from [14] by analyzing the SBOLv diagrams of all the circuits shown in [14] and arranged them in the separate lists of genetic NOT and NOR gates. The list of genetic NOT and NOR gates is shown in Figure 6.7. The lists shown in this figure consists of 35 NOR and 17 NOT gates, which indicates that there are several different genetic components which can be used to perform the logical NOR or NOT operations inside a living cell.

Figure 6.8 shows the equivalent SBOLv diagrams [10] of the genetic NOT-1 (*"1" refers to the identity of this gate shown in Figure 6.7*) and NOR-1 gates. For example, the input of a genetic NOT-1 gate, shown in Figure 6.8, is P_{Tac} promoter which produces the protein AmtR based on the presence or absence of isopropylethio-glactoside (IPTG) inducer. If IPTG is absent, the input promoter, P_{Tac} is active (logic-1) and produces the AmtR protein, which then suppresses the activity of the output promoter PAmtR (logic-0), and vice versa.

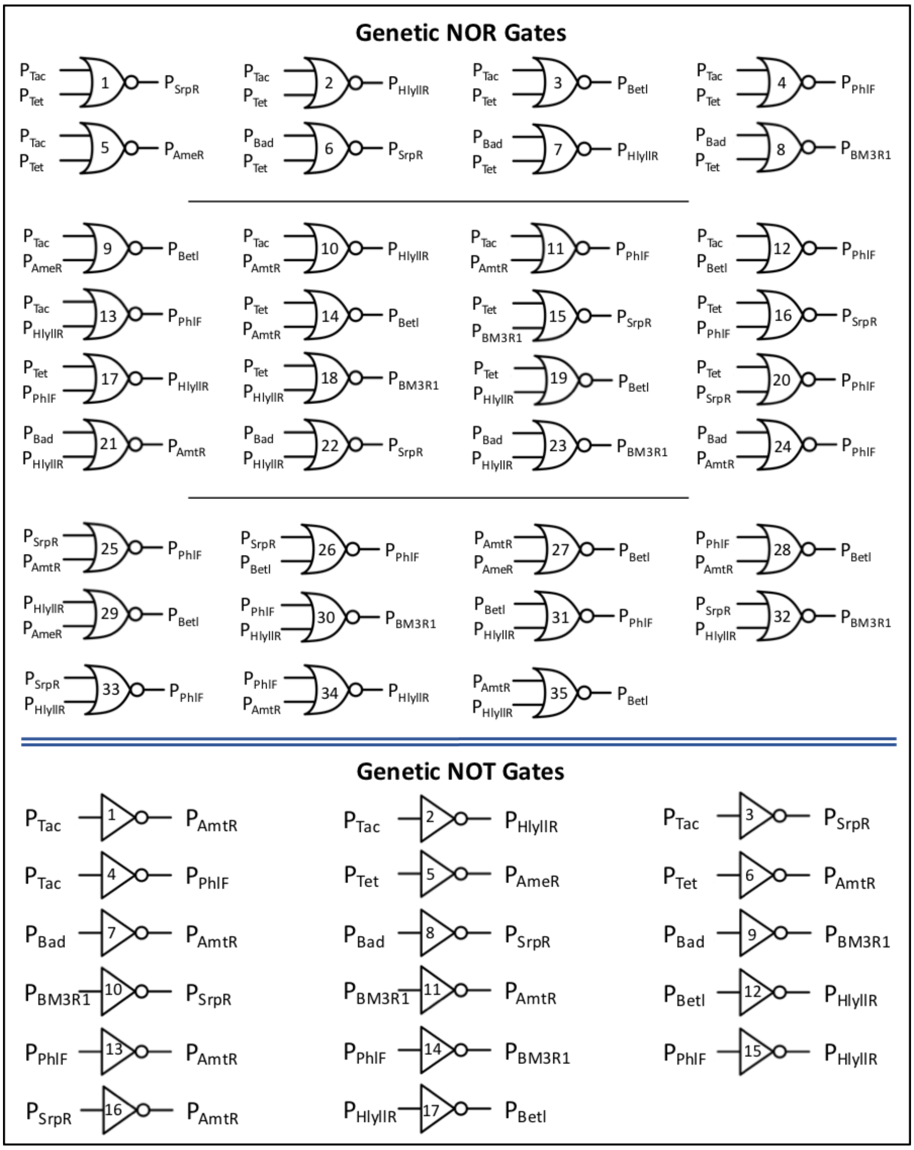


Figure 6.7: Genetic gates library constructed from the circuits shown in [14].

Similarly, for the NOR-1 gate, shown in Figure 6.8, the activities of the input promoters, P_{Tac} and P_{Tet}, are controlled by the inducers IPTG and anhydrotetrascycline (aTc), respectively. When both of them are present, the activities of input promoters, P_{Tac} and P_{Tet}, are suppressed (logic-00), which reduces down the production of protein SrpR. The output promoter, P_{SrpR}, becomes active (logic-1) when the protein SrpR is not produced in a significant amount to suppress it. When either or both of these inducers are absent, the corresponding input promoter(s) becomes active (logic-01, 10, 11) and produces the protein, SrpR, which in turn suppresses (logic-0) the activity of output promoter, P_{SrpR}, thus exhibiting the NOR logic.

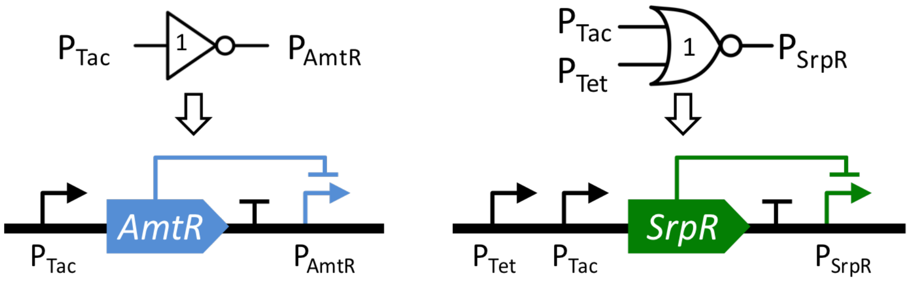


Figure 6.8: SBOLv diagrams of NOT-1 and NOR-1 gates.

As discussed above, the input and output quantities in genetic gates are different. This make it challenging to construct a genetic circuit by making sure that the output of the first gate is compatible with the input of the following one. The *GeneTech* mapping algorithm constructs a genetic circuit by using the genetic gates, from the gates library (Figure 6.7), whose inputs and output proteins are matched with each other.

The mapping algorithm is explained briefly with the help of example shown in Figure 6.9. In this figure, the output expression from the previous stage, *logic synthesis*, is considered to be an input to the algorithm for technology mapping. Inputs a, b and c correspond to the external inputs, P_{Tac}, P_{Tet}, and P_{Bad}, respectively in [14]. The algorithm works on the depth-first search approach and hence begins by mapping genetic gates first on the deepest element(s) in the expression. Therefore, the list of all the possible NOT gates for $\bar{b}$ are extracted from the genetic gates library first.

For understanding, let us name the list of inverters for $\bar{b}$ as list-A, which consists of NOT-5 and NOT-6 gates. The algorithm selects NOT-5 first from list-A, and then checks if its output is compatible with the input a (or P_{Tac}) in the form of NOR gate. At this step, a new list, say list-B, is created which contains three NOR gates with P_{Tac} as one of their inputs. Since the output of NOT-5 is compatible with the input of the NOR-9, the algorithm proceeds further with the

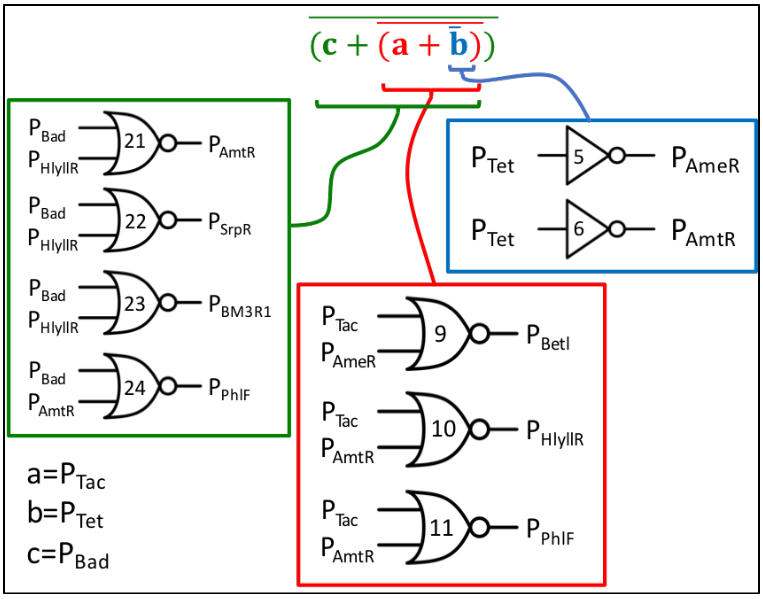


Figure 6.9: Explanatory example of mapping algorithm.

search of NOR gate(s) with one input c (or P_{Bad}) and the other one compatible with the output of NOR-9. Again, a new list, say list-C, is created containing four NOR gates with P_{Bad} as one of their inputs. The second input of any of these 4 gates (in list-C) do not match with the output of NOR-9 i.e. P_{Betl}. The algorithm then steps back and remove NOR-9 from list-B, and search for any other NOR gate compatible with the output of NOT-5, P_{Amer}. Since there are no other NOR gates available in list-B which are compatible with P_{Amer}, the algorithm further steps back and remove NOT-5 from list-A. The algorithm then selects NOT-6 and proceeds further by checking if its output, P_{Amtr}, is compatible with the available NOR gates in list-B. Both of the remaining NOR gates in list-B (2 out of 3), NOR-10 and NOR-11, matches the output of the gate NOT-6. The algorithm selects NOR-10 first and proceeds ahead by checking the compatibility of its output, P_{HlyllR}, with the NOR gates available in list-C.

At this stage, there are three matching NOR gates present in the list which can be used to construct the final stage of circuit. This may result in three possible circuits constructed from the genetic gates in sequence of NOT-6 $\rightarrow$ NOR-10 $\rightarrow$ NOR-21, NOT-6 $\rightarrow$ NOR-10 $\rightarrow$ NOR-22, and NOT-6 $\rightarrow$ NOR-10 $\rightarrow$ NOR-23. There are no matching gates available in list-C with the output of second stage NOR-11 from list-B. Therefore, to implement the desired logic, the total number of circuits generated by the algorithm would be three.

While constructing genetic circuits, the algorithm avoids using the components which makes an unintended feedback loop with the preceding stage components. For example, the circuit diagram of one of the above mentioned solutions, with the components NOT-6 → NOR-10 → NOR-21, shown in Figure 6.10, has the output of the final gate, NOR-21, being the same as one of the input of the previous gate, NOR-10. In genetic circuits, signals are molecules and unlike electronic circuits, they do not propagate in separate wires. Therefore, it is impossible to prevent the AmtR protein generated by the NOR-21 gate from suppressing both the output promoter PAmtR for NOR-21 and the input promoter for NOR-10. Hence, the algorithm discards this circuit for achieving the desired functionality, and hence the final number of possible circuits are two.

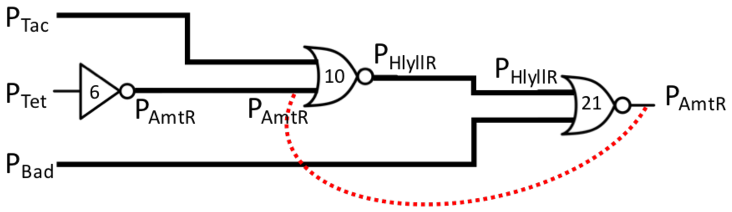


Figure 6.10: Circuit diagram of one possible solution, for the example expression shown in Figure 6.9, which creates an unintended feedback loop.

The pseudo-code of the mapping algorithm is presented in Algorithm 6.3. The procedure, *SolveNestedEl*, in line 3 recursively extracts all the possible gates (from the genetics gates library) for each circuit element in the expression and arrange them in separate lists. Any empty list indicates that the genetic gates are not available in the library for the desired input combination. In the case of empty list, the circuit cannot be generated and the algorithm stops executing. Once the lists are generated for all circuit elements, the procedure *GatesMatching* search for the matching gates which can be cascaded together. This procedure also filters out those components which forms an unintended feedback loop with the preceding gates, as described above in Figure 6.10. Even if the list of circuit elements is not empty, it may happen that the involved gates cannot be cascaded together due to incompatible input and output. This screening is also performed by the same procedure *GatesMatching*. Once all the matching gates are found, the procedure, *GenerateCircuits*, cascade all the possible compatible gates together to construct different possible circuits. The set of generated circuits are meant to exhibit the same Boolean logic, with the combination of different genetic components, and without causing cross-interference with each other.

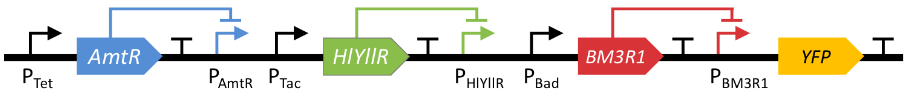
Algorithm 6.3: Pseudo code of *technology mapping.*

```
input  : $Exp_{in}$
output: $Exp_{out}$
// $Exp_{in}$ = Input expression, $Exp_{out}$ = Output Expression
begin
    for (all NOR terms, $Exp_{Nor}$, in $Exp_{in}$) do
        SolveNestedEl($Exp_{Nor}$);
        if (any LIST is not empty) then
            GatesMatching; // perform gates matching
            GenerateCircuits; // generate all possible circuits
        else
            Print"Gates not available in library;
        end
        Procedure SolveNestedEl(Expression)
            if ($Exp_{Nor}$ contains braces) then
                for (all minterms, M, in $Exp_{Nor}$) do
                    if (M contains further braces) then
                        $Exp$ = Extract expression inside braces;
                        SolveNestedEl($Exp$);
                    else
                        Create the list, LIST, of all possible gates of M;
                    end
                end
            else
                for (all minterms, M, in $Exp_{Nor}$) do
                    Create the list, LIST, of all possible gates of M;
                end
            end
    end
end
```

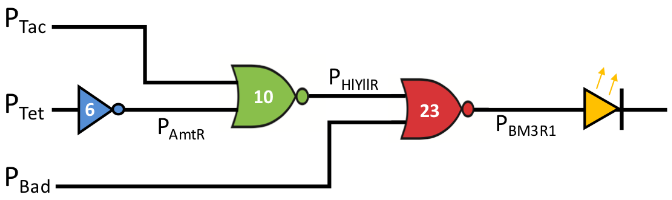
GeneTech generates the circuit in a text string format which resembles the structure of SBOLv. For instance, one of the possible circuits, for the example shown in Figure 6.9, consists of NOT-6 → NOR-10 → NOR-23. *GeneTech* structured this circuit in the form shown in Figure 6.11(a). The corresponding SBOLv diagram and the gate-level circuit schematic of Figure 6.11(a) are shown in Figure 6.11(b) and Figure 6.11(c), respectively. In Figure 6.11(a), promoters are shown with the symbol "->", the generated proteins are represented by round braces "( )", and the repression is indicated by the symbol "−−−|" or "T". Figure 6.11(a) indicates that a P_{Tet} promoter generates a protein AmtR which in turn represses the output promoter P_{AmtR}. The promoter P_{AmtR} together with the promoter P_{Tac} generate the protein HlYllR, which represses the out-

PTet-> (AmtR)----|PAmtR-> PTac-> (HlYIIR)----|PHlYIIR-> PBad-> (BM3R1)----|PBM3R1-> (YFP)

(a)



(b)



(c)

Figure 6.11: One possible solution for the expression shown in Figure 6.9. (a) Circuit diagram developed by *GeneTech*. (b) Equivalent SBOLv diagram. (c) Equivalent gate-level circuit schematic.

put promoter P_{HlYIIR}. The promoter P_{HlYIIR} together with the promoter P_{Bad} generates a protein BM3R1, which represses the activity of the output promoter P_{BM3R1}. The promoter P_{BM3R1} is used to produce the output indicator, the yellow fluorescent protein (YFP), which is shown as a yellow LED in Figure 6.11(c). *GeneTech* use multi-lines text string to represent more complex genetic circuits.

6.3 Experimentation by Simulation

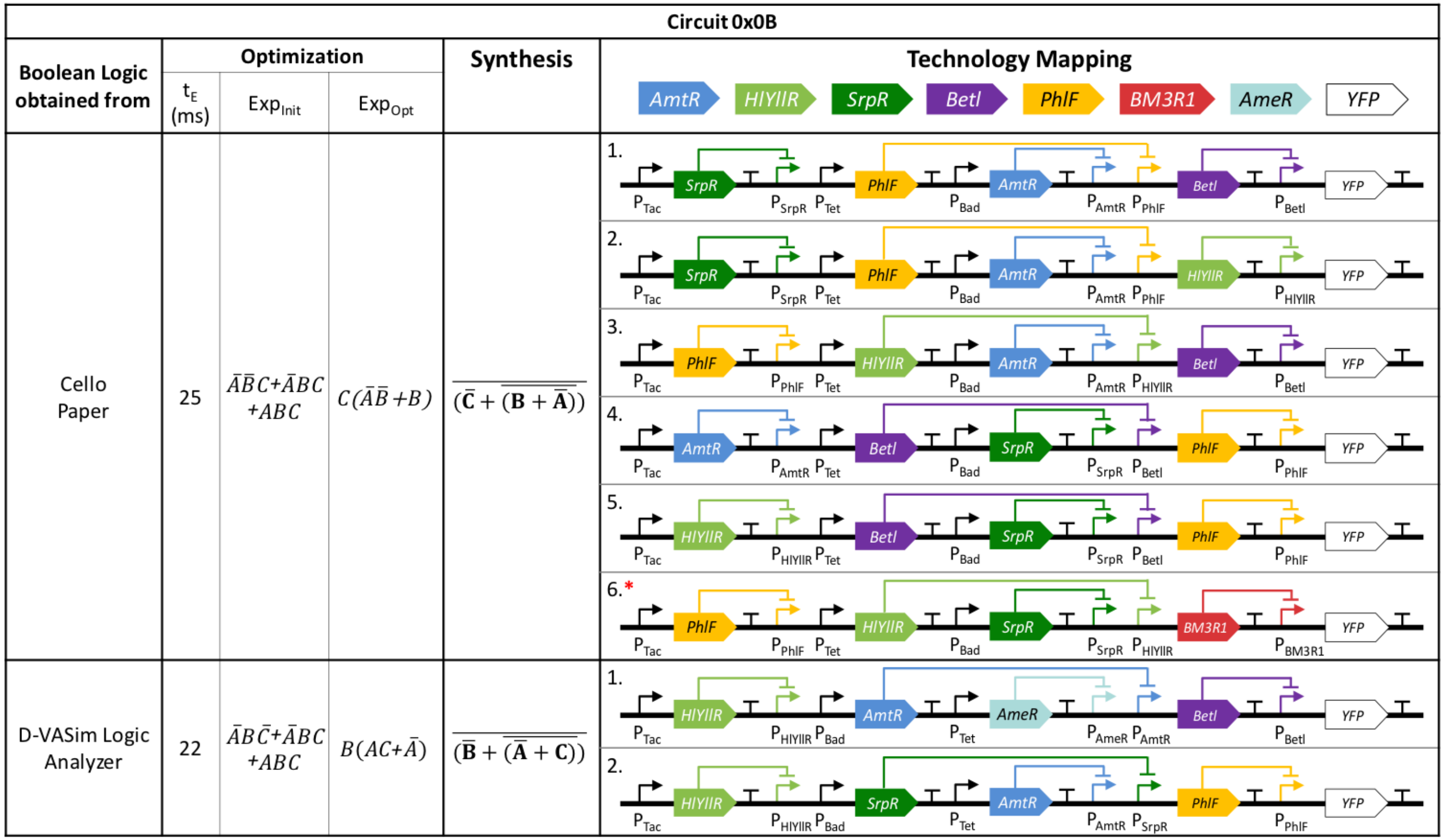
GeneTech is tested on five genetic circuits (0x0B, 0x70, 0xC4, 0xC8 and 0x0E) obtained from [14]. The Boolean expression of these five circuits are obtained from their respective truth tables shown in [14]. The truth table and Boolean expressions of the circuits, 0x0B, 0x70, 0xC4, 0xC8 and 0x0E, are reproduced in this thesis and can be seen in Figures 5.3, C.6a(ii), C.7a(ii), C.8a(ii), and C.9a(ii), respectively.

As mentioned before in Chapter 5 that the inputs B and C, in all SBOL/SBML circuit models obtained from [14], are interchanged. Due to this change in circuit inputs, the corresponding change in the logic behaviors of these circuits are also verified using D-VASim (*See Chapter 5 and Appendix C*). Therefore, each of the above mentioned five circuits are tested on *GeneTech* using two different expressions i.e. the original shown in [14] and the one obtained from D-VASim (*See Chapter 5 and Appendix C*) with the inputs B and C swapped.

The functionality of the mapping algorithm depends on how the Boolean expression is optimized in the first step by logic optimization algorithm (shown in Algorithm 6.1). The optimization via the simulated annealing algorithm mainly depends on the parameters; initial temperature (T_0), temperature coefficient (T_{COF}), and the time to run the algorithm (t_E).

In these experiments, each circuit is run for $T_0 = 10$, 20 and 30 ^0C; $T_{COF} = 0.90$, 0.95, and 0.99; and $t_E = 15$, 20, 25, 30 and 40 milliseconds. For all combinations of these parameters, each circuit is run for 10 times, resulting in a total of 2000 simulation experiments. Based on the mean and the standard deviation values of these experiments, it was observed that the parameter values, $T_0 = 10$ and $T_{COF} = 0.90$, were suitable for generating the optimized expression. Beside these, it is understood that the optimization of a Boolean expression mainly depends on the amount of time the optimization algorithm is run for each circuit, i.e. t_E. It also depends on the combination of minterms selected randomly for optimization (see the explanation of equation 6.4). For example, the circuit may reduce the expression into most optimized form when it runs for, say 25ms, as compared to 20ms. It may also be possible that running the same expression again for 25ms may not produce the optimized expression. That may happen because a wrong combination is selected randomly by an algorithm to reduce the expression. Since the algorithm is not run for sufficiently long amount of time, therefore the algorithm is unable to come out of the local optimum value and does not produce the most optimized expression. We therefore report the time, t_E, which ensures that the algorithm will run sufficiently long to produce the optimized expression even if it stuck in the local optimum value.

Figure 6.12 depicts the experimental results of the genetic circuit 0x0B, obtained from [14]. The results of remaining four circuits are given in Appendix D. For each circuit, separate results are included for the Boolean logic obtained originally from [14], and for the one obtained experimentally by using the logic verification method in D-VASim (*See Chapter 5 and Appendix C*). The optimum time to run the algorithm, t_E, on each of the five circuits are mentioned in milliseconds, which shows that each circuit produce optimized expression when run for specified interval of time. Exp_{Init}, indicates the initial Boolean expression taken as input. Exp_{Opt}, specifies the expressions after being optimized through logic optimization algorithm (Algorithm 6.1). The expressions shown in the column of Synthesis shows the results of logic synthesis algorithm (Algorithm 6.2), which converts the optimized expression into NOR/NOT form. The SBOLv diagrams of all the generated circuits, using mapping algorithm (Algorithm 6.3), are displayed in the right most column in Figure 6.12.



Circuit 0x0B					
Boolean Logic obtained from	**Optimization**			**Synthesis**	**Technology Mapping** (AmtR, HlYIIR, SrpR, BetI, PhlF, BM3R1, AmeR, YFP)
	t_E (ms)	Exp_{Init}	Exp_{Opt}		
Cello Paper	25	$\bar{A}\bar{B}C+\bar{A}BC$ $+ABC$	$C(\bar{A}\bar{B}+B)$	$\overline{(\bar{\mathbf{C}} + \overline{(\mathbf{B} + \bar{\mathbf{A}})})}$	1. P_{Tac} SrpR P_{SrpR} P_{Tet} PhlF P_{Bad} AmtR P_{AmtR} P_{PhlF} BetI P_{BetI} YFP
					2. P_{Tac} SrpR P_{SrpR} P_{Tet} PhlF P_{Bad} AmtR P_{AmtR} P_{PhlF} HlYIIR P_{HlYIIR} YFP
					3. P_{Tac} PhlF P_{PhlF} P_{Tet} HlYIIR P_{Bad} AmtR P_{AmtR} P_{HlYIIR} BetI P_{BetI} YFP
					4. P_{Tac} AmtR P_{AmtR} P_{Tet} BetI P_{Bad} SrpR P_{SrpR} P_{BetI} PhlF P_{PhlF} YFP
					5. P_{Tac} HlYIIR P_{HlYIIR} P_{Tet} BetI P_{Bad} SrpR P_{SrpR} P_{BetI} PhlF P_{PhlF} YFP
					6.* P_{Tac} PhlF P_{PhlF} P_{Tet} HlYIIR P_{Bad} SrpR P_{SrpR} P_{HlYIIR} BM3R1 P_{BM3R1} YFP
D-VASim Logic Analyzer	22	$\bar{A}B\bar{C}+\bar{A}BC$ $+ABC$	$B(AC+\bar{A})$	$\overline{(\bar{\mathbf{B}} + \overline{(\bar{\mathbf{A}} + \mathbf{C})})}$	1. P_{Tac} HlYIIR P_{HlYIIR} P_{Bad} AmtR P_{Tet} AmeR P_{AmeR} P_{AmtR} BetI P_{BetI} YFP
					2. P_{Tac} HlYIIR P_{HlYIIR} P_{Bad} SrpR P_{Tet} AmtR P_{AmtR} P_{SrpR} PhlF P_{PhlF} YFP

Figure 6.12: Experimental results of *GeneTech* for the optimization, synthesis and technology mapping of 0x0B circuit.

For ease of understanding and to be consistent with the conventions used in [14], the gates are shown in colors. The color legends shown at the top signifies that the components shown with these colors (in the SBOLv diagrams of generated circuit) produce the corresponding proteins. The number of genetic gates in these auto-generated circuits vary from 3 to 6, with no repetition of same genetic gates (which generates the same output protein) in any of them. In general, the number of possible solutions are more for the circuits originally obtained from [14], and less for the Boolean behavior of these circuits obtained using D-VASim (*See Chapter 5 and Appendix C*). In other words, the number of possible circuits are reduced when the inputs B and C are swapped (also see Appendix D). In case of circuit x70 (see Appendix D), for the Boolean logic obtained from D-VASim (with inputs B and C swapped), no circuit is generated because there is no NOR gate available in the genetic gates library (shown in Figure 6.7) with P_{Tac} and P_{Bad} promoters, as inputs.

In case of 0xC8 (see Appendix D), for the logic expression obtained from [14], the components in circuit 1 and 2 are same, however the input source of generating SrpR and AmtR proteins are interchanged in both circuits. In circuit 1, AmtR and SrpR proteins are produced by the P_{Tac} and P_{Bad} promoters respectively, but vice versa in circuit 2.

Among several possible solutions which *GeneTech* proposed for 0x0B circuit in Figure 6.12, the solution similar to the one mentioned in [14] is marked with red asterisk (*). Similarly, the solutions proposed in [14] for circuits 0xC4 and 0x70 are also marked with red asterisk (*) (see Appendix D). However, for the remaining two circuits, 0xC8 and 0x0E, *GeneTech* did not generate any circuit similar to their SBOLv diagram shown in [14]. It is because these circuits are shown in [14] as a multi-level output circuits, which means that the production of output protein, YFP, depends upon the activities of multiple output promoters. This is depicted as an OR logic in [14]. For example, the schematic of a circuit 0xC8 (reproduced in Figure C.8a(i)) consists of two NOR gates both of which are generating YFP, and their combined output is shown as an OR gate. It may simply be considered as two different NOR gates which can be independently used to trigger YFP. However, in Figures D.3 and D.4 (Appendix D), the results of *GeneTech* show that it is possible to obtain the same logic behavior, using single-level output circuits.

6.4 Discussion

In this chapter, it has been demonstrated with the help of five case studies that the proposed tool, *GeneTech*, can be used to develop a genetic circuit from a raw Boolean expression. In [14], a circuit is generated by a random search of compatible genetic gates using the simulated annealing algorithm. Since the circuit is created using the non-deterministic search, the solution may be different every time the process is executed. On the contrary, *GeneTech* finds all possible genetic circuits based on a deterministic approach. *GeneTech* is also scalable, meaning that it can process newly added genetic gates without any further modifications.

This tool is typically helpful for both, the biologists and the computer scientists. It is because the computer scientists do not need to learn any biological terminologies nor in which order the genetic components should be connected to construct genetic circuits. Biologists, on the other hand, do not need to learn any programming language or syntax to design genetic circuits *in-silico*. The users only need to specify the boolean behavior, in the form of expression, they want to achieve in a living cell, and *GeneTech* lists down all the possible circuit structures to achieve it.

CHAPTER 7

Parameter Estimation and Sensitivity Analysis

We have demonstrated that the genetic models can be simulated intuitively in D-VASim (Chapter 3) and their threshold values, propagation delays (Chapter 4) and logic (Chapter 5) can be analyzed. We have made assumptions about many of the model parameters such as reaction and degradation rates of chemical reactions. As demonstrated in Chapter 4, the exact values of these parameters may have an effect on the overall circuit behavior. Many of these parameters are very difficult to determine without doing wet-lab experiments. However, as an alternative, we may do a sensitivity analysis, i.e. determining the range of values for a certain parameter for which the model behaves correctly. In this chapter, we show how statistical model checking can be used to make such sensitivity analyses.

In Section 7.1, it is shown that how D-VASim can assist in model checking of genetic circuits along with the statistical model checking (SMC) tool like Uppaal [18]. In Section 7.2 we show an attempt to relate Cello [14] UCF parameters with iBioSim default simulation parameters to perform more realistic simulations in D-VASim. In this regard, we derived a relation between some of the iBioSim's default parameters and Cello parameters. Then we demonstrate how the model's behavior is effected by using real parameter values.

The work presented in Section 7.1 has been disseminated for publication as mentioned below.

[67] Hasan Baig and Jan Madsen, "Taming Living Logic using Formal Methods", *Lecture Notes in Computer Science*, Springer, pp. 489–501, 2017. (Accepted)

7.1 Statistical Model Checking with Uppaal

During the design process of genetic circuits, biologists are often interested in the probability of a system to work under different conditions. Since genetic circuits are noisy and stochastic in nature, the verification process becomes very complicated. The state space of stochastic genetic circuit models is usually too large to be handled by classical model checking techniques. Therefore, the verification of genetic circuit models is usually performed by the statistical approach of model checking.

The dynamics of genetic circuits, and hence their correct functioning, are dependent on a large set of parameters (such as reaction and degradation rates) which in general are very difficult to predict and control. Hence, biologists are usually interested in determining the sensitivity of their circuits for fluctuations in these parameters. For instance, it might be a question of interest to find out, if the circuit behaves as expected when the values of certain parameters are varied within a specified range. Such sensitivity analysis is well suited for explorations using statistical model checking (SMC) and the aim of this work is to show how Uppaal SMC [18] can be used to address the problem, effectively taming living logic.

Model checking of biological systems is getting popular as it is an effective means of analyzing the dynamics of complex biological systems. Besides many interesting work in this domain [68–76], Madsen *et al.* [77] proposed a stochastic model checking approach which greatly speeds up the search process of genetic design space by using the numerical techniques (Markov chain analysis). In this work, multiple threshold levels are used, to test each circuit, which are obtained by graphing reaction rates and selecting few values around the inflection point. However, we used the D-VASim tool to obtain the specific value of threshold level and used it for model checking in Uppaal.

In this section, a workflow for checking genetic circuit models using a statistical model checker (Uppaal SMC) and the stochastic simulator (D-VASim) is presented. We demonstrate with experimentation that the proposed workflow is not only sufficient for the model checking of genetic circuits, but can also be used to design the genetic circuits with desired timings. In particular, we performed experimentation on genetic circuits models and explored their design parameter sensitivity using Uppaal SMC [18]. There are certain number of tasks which cannot be performed in Uppaal [15]. We therefore used D-VASim [16] to address those, which will be detailed in the experimentation section.

7.1.1 Methodology

To determine the range of parameter values for which the genetic circuit would work, it is first important to know the threshold concentration levels of the inputs of those circuits. As stated in Definition 4.1 in Chapter 4, the threshold level of a genetic circuit can be defined as the minimum concentration of input protein(s), which causes the average concentration of output protein to cross the level of input protein(s) concentration.

D-VASim [16] is a simulation tool which supports the capability of analyzing the threshold value and timings of genetic circuits through an automated process. Once the correct threshold levels are found, the inputs are triggered to that level and the circuit parameters can be varied to determine if the circuit still behaves correctly. This analysis could be very time consuming for large genetic circuit models with more inputs. For large-scale circuits, it is difficult to determine or verify the expected logic of a circuit without careful analysis. To determine or verify the logic of a genetic circuit, it is important to know the correct input combination with the correct threshold levels which trigger the output of the circuit. This may apparently become a tedious task to check different input concentration levels for each input combination.

7.1.2 Role of D-VASim in model checking

The search process of threshold value can be automated by the use of statistical model checking in Uppaal [15]. Uppaal is an integrated tool environment for modeling, verification and validation of real-time systems modeled as networks of timed automata. *Uppaal SMC* [18] is an extended plug-in tool to Uppaal which allows the user to check the expected behavior of models in the form of probability distributions.

In Uppaal SMC, it is possible to let the tool arbitrarily select any input concentration value, within a specified range, and see if the chosen value significantly effects the output concentration level. This can, however, only be achieved when

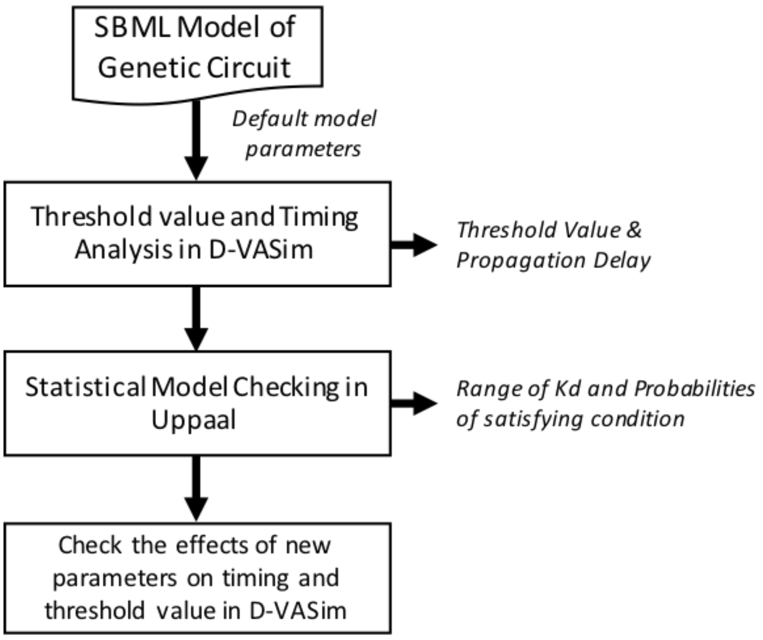
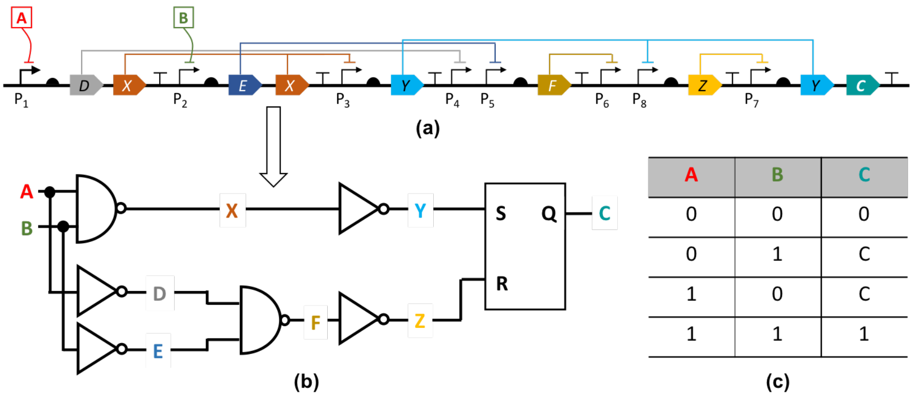

SBML Model of Genetic Circuit
Default model parameters
Threshold value and Timing Analysis in D-VASim
Threshold Value & Propagation Delay
Statistical Model Checking in Uppaal
Range of Kd and Probabilities of satisfying condition
Check the effects of new parameters on timing and threshold value in D-VASim

Figure 7.1: Experimental flow of genetic circuit model checking and verification.

the correct input combinations triggering the output of the circuit are known. As Uppaal does not have the capability to automatically detect the input combination which triggers the output of the circuit, the threshold value analysis of a genetic circuit cannot be performed automatically in Uppaal. We also tried to find the threshold concentration levels for the specific input combination (which triggers the output concentration). However, Uppaal SMC was unable to identify the appropriate threshold level because of the infinite-sized search-space of floating-point concentration values.

As stated before, D-VASim [16] is the only tool which allow users to perform threshold value and propagation delay analysis through an automated process [39]. However, D-VASim is not capable of performing the automated statistical model checking. Thus, we used D-VASim for threshold value analysis and then perform the statistical model checking in Uppaal to determine the range of circuit parameters within which the circuit satisfy the desired behavior.

The proposed experimental flow of checking genetic circuit models is shown in Figure 7.1. The genetic circuit models developed in the SBML [78] are used in this work. The SBML model of a genetic circuit is used as input to D-VASim.



A	B	C
0	0	0
0	1	C
1	0	C
1	1	1

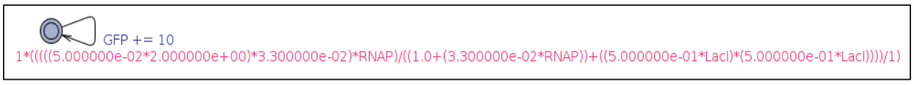
Figure 7.2: Genetic toggle muller-C element [21]. (a) SBOLv diagram. (b) Circuit schematic. (c) Truth table.

D-VASim analyze the threshold and propagation delay (details are given in next section). The threshold value is then used in Uppaal to trigger the input levels to this value and observe the output behavior of the circuit while varying the circuit parameters. The effects of varying parameters on the threshold value and propagation delay of the circuit are then analyzed in D-VASim.

7.1.3 Experimentation by Simulation

In this work, the genetic circuit models from [21] are tested, by varying the degradation rate parameter (k_d) to determine the range within which the circuit exhibits the expected behavior. The aim is to propose an experimental flow for model checking of genetic circuits. To demonstrate that this flow can be applied to a complex genetic circuit as well, the experimental results of a small (NAND gate) and a reasonably large (toggle muller-C element) genetic circuit models are included. The NAND gate contains 5 species and 5 kinetic reactions, whereas the toggle muller-C element contains 20 species and its behavior is defined by 18 kinetic reactions.

The SBOLv diagram, circuit schematic and the truth table of the toggle muller-C element are shown in Figure 7.2 (a), (b) and (c), respectively. In Figure 7.2(a), the input protein A suppresses promoter P1 to produce protein D, which in turn inhibits promoter P4 to reduce the production of protein F, and so on. The concentration of the output species, C, toggles when both of the inputs are changed together, else the output retains its previous state.


GFP += 10
1*(((((5.000000e-02*2.000000e+00)*3.300000e-02)*RNAP)/((1.0+(3.300000e-02*RNAP))+((5.000000e-01*LacI)*(5.000000e-01*LacI))))/1)

(a)

```
koP1*ngP1*KoP1*RNAP/(1+KoP1*RNAP+(KrLacIP1*LacI)^ncLacIP1)
```

(b)

Figure 7.3: The process of a genetic NAND gate to produce 10 molecules of the GFP when the input LacI is not present in a cell. (a) Uppaal interpretation. (b) Kinetic Reaction.

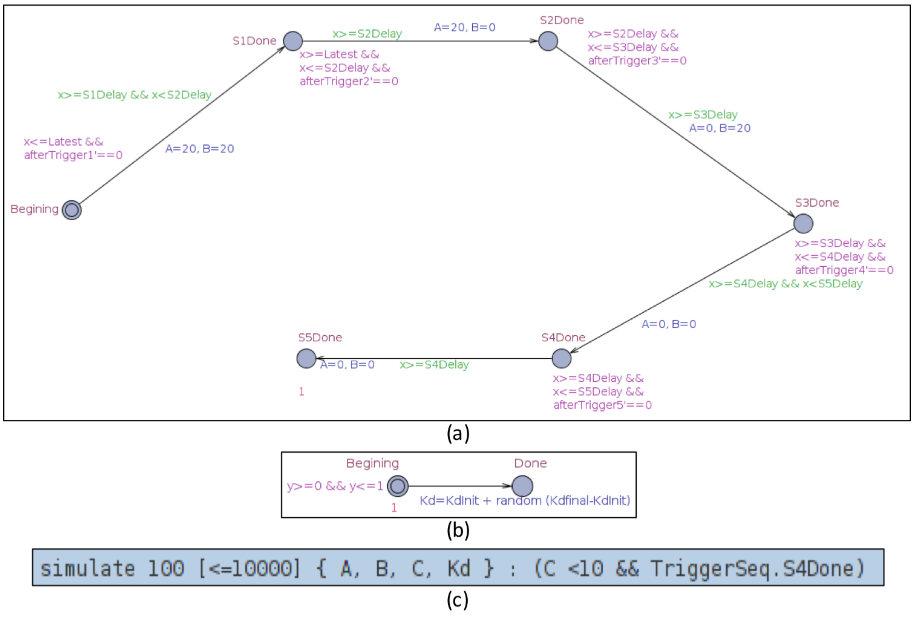
Table 7.1: Threshold and propagation delay values obtained in D-VASim prior to model checking in Uppaal.

Circuit name	Threshold value (High)	Threshold value (low)	Propagation delay value
NAND	15	5	324 (±51.61)
Toggle Switch	10	5	1108 (±272.89)

Table 7.1 shows the threshold and propagation delay values for both of the circuits obtained from D-VASim. The high threshold value specifies the input concentration level above which the logic is considered high, and the low threshold value specify the input concentration level below which the logic is considered low. The propagation delay, as defined in Definition 4.2 in Chapter 4, is the time from when the input concentration reaches its threshold value until the corresponding output concentration crosses the same threshold value.

Uppaal uses a continuous time markov chain model (CTMC) for model checking, therefore the SBML models were first converted into CTMC models using the simple conversion utility in Uppaal. It creates a separate automaton for each of the reaction kinetics defined in the SBML file. For instance, Figure 7.3(a) shows one of the processes, in the genetic NAND gate circuit, which represents the kinetic reaction (Figure 7.3(b)) to produce 10 molecules of GFP when the input protein LacI is not sufficiently present in the cell. In Figure 7.3(b), the value of ncLacIP1 is 2, due to which the factor KrLacIP1*LacI is multiplied twice in Figure 7.3(a).

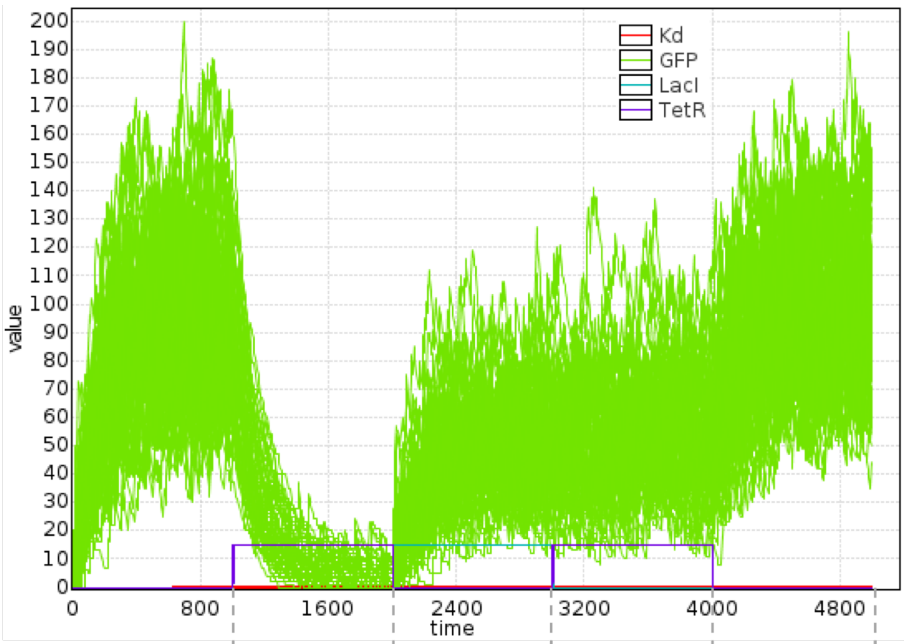
We then created a separate process for triggering all the possible input combinations to their threshold values (obtained from D-VASim), as shown in Figure 7.5(a). In this figure, each node represents the automaton (or state) and each edge represents the action. For example, the automaton "Beginning" starts at zero time units and fires the first edge to trigger the input concentrations, A and B, to 20 molecules. This state is defined by an invariant "$x <= Latest\&\&afterTrigger1' == 0$", where the value of $Latest$ is 2000, x is the main clock and *afterTrigger1* is a local clock for this state. The edge is fired after the delay of 2000 time units (defined by *S1Delay*) and runs for next 2000 time units (defined by *S2Delay*) i.e. until the time equals 4000 time units. The edge is guarded by the assignment expression "$x >= S1Delay\&\&x < S2Delay$". Another automaton process is used to vary the values of k_d as shown in Figure 7.5(b). During each iteration of simulation, the value of k_d is chosen randomly from the range defined by $KdInit$ and $Kdfinal$ values. Then the UPPAAL SMC tool verifies the user-defined queries. An example query is shown in Figure 7.5(c), where the model is required to be simulated 100 times each for 10,000 time units and to check if the concentration of output protein C is less than 10 molecules when the state "S4Done" is triggered.



```
simulate 100 [<=10000] { A, B, C, Kd } : (C <10 && TriggerSeq.S4Done)
```

Figure 7.4: The process automaton for triggering inputs and varying k_d.

Figures 7.5 and 7.6 shows the Uppaal SMC simulation results of the genetic NAND and the toggle muller-C element switch circuits, respectively. These figures show all the simulation traces for 100 iterations. All possible input combinations are applied and the correct operation is verified within a defined range of k_d. Due to the stochastic nature of a model, the probability of an expected behavior cannot be 100 percent satisfied when the value of k_d is randomly chosen

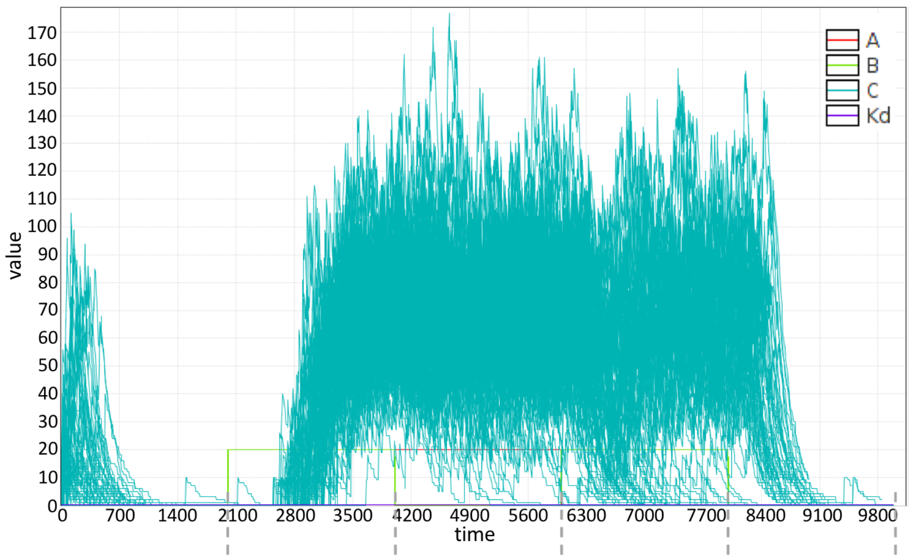


Input combinations:	11	10	01	00
Probability:	0.92-0.99	0.95-1	0.95-1	0.95-1
Satisfying Simulations:	98 %	100 %	100 %	100 %

Figure 7.5: Statistical model checking of the genetic NAND gate in Uppaal.

from a defined range. We, therefore, set the probability of the expected behavior to be greater than at least 90 percent as the acceptance criteria. Inputs correspond to the applied combination of input proteins over the course of simulation time. The logic-1 for the NAND gate corresponds to 15 or more molecules and logic-0 corresponds to 5 or less molecules. For the toggle muller-C element, the logic-1 corresponds to 20 or more molecules and logic-0 corresponds to 10 or less molecules, as obtained from D-VASim (see Table 7.1).

Probability values at the bottom of both Figures 7.5 and 7.6 signifies the probability of the expected behavior of a circuit for all possible input combinations, where each input combination is applied for 1000 time units for the NAND gate and 2000 time units for the toggle muller-C element. These values are chosen sufficiently larger than their respective propagation delay values, estimated from D-VASim, to ensure that the appropriate amount of delay is provided to observe the effects of applied input combinations on the output of the circuit.



Input Combinations:	**11**	**10**	**01**	**00**
Probability:	0.94-0.99	0.96-1	0.86-0.97	0.96-1
Satisfying Simulations:	99%	100 %	93 %	100 %

Figure 7.6: Statistical model checking of the genetic toggle muller-C element in Uppaal.

Satisfying Simulations indicates the percentage of simulations which satisfy the defined condition for specific input combination. These conditions are set according to the truth tables of respective circuits. For example, for the NAND gate, the condition to be checked for when the input combination is 11, is to see if the concentration of output protein, GFP, falls below its lower threshold level i.e. 5 molecules. The NAND gate circuit exhibits the probability of greater than 98 percent to work correctly when the value of k_d varies between 45x10-4 and 85 x10-4. Similarly, the toggle muller-C element is at least 93 percent probable to work correctly when the value of k_d varies between 60x10^{-4} and 85x10^{-4}. Outside these ranges of k_d, the expected behavior do not satisfy the acceptance criteria mentioned above. In a similar manner, other circuit parameters can be varied to check the output response of genetic circuits.

Finally, we used D-VASim to observe how the changes of k_d values impact the threshold value and the output of a circuit. In Table 7.2, the effects of the boundary values of k_d for both circuits are shown. For example, in the case of the NAND gate, the effects of lower and higher-bound values of a k_d, 45x10^{-4} and 85x10^{-4}, respectively, are checked. It is observed that the upper threshold

Table 7.2: Threshold and propagation delay values obtained in D-VASim for upper and lower bounds of k_d values found in Uppaal.

Circuit name	k_d (x10^{-4})	Threshold value (High)	Threshold value (Low)	Propagation delay
NAND	45	20	10	554 ($\pm$ 56.07)
	85	15	0	274 ($\pm$ 91.78)
Toggle Switch	60	20	10	1228 ($\pm$ 135.11)
	85	10	5	833($\pm$ 97.41)

concentration level required to trigger the output of the NAND gate is increased from 15 to 20 molecules when the value of k_d was decreased from 75x10^{-4} (default value) to 45x10^{-4}. An increment in the propagation delay value is also observed. The latter is due to the fact that a decrease in the degradation rate causes the output response of the circuit to be slower, and thus more input concentration may be required to trigger the output. If the threshold value of a circuit is kept to its previous value, i.e., 15 at k_d = 45x10^{-4}, the output may appear after a very long time; in other words, the propagation delay increases further.

Likewise, when the value of k_d is increased to 85x10^{-4}, the threshold values as well as the propagation delays are decreased. Similar observations have been made for the toggle muller-C element as shown in Table 2. These observations indicate the minimum-high and maximum-low threshold values. For example, in order for the toggle muller-C element to work within a range of k_d between 60x10^{-4} and 85x10^{-4}, the minimum-high threshold value would be 20 molecules and a maximum-low threshold value would be 10 molecules.

7.2 Semi-realistic Simulation with Cello UCF parameters

It is mentioned in [14] that out of 60 genetic circuits, 15 did not work as expected when tested in wet-lab. We were interested in determining the cause of their failures through simulation with the hope that the *timing analyses* may help in identifying the cause of failure. In order to capture the correct behaviour, it is important to simulate those circuits with the same parameters and circumstances under which they were actually carried out in the wet-lab. We were interested in finding out the *real* values of the parameters (use by iBioSim [17]) for the genetic circuits developed by Nielsen et al. [14]. The default values of these parameters, used by iBioSim, [21, 79] are reproduced in Table 7.3. After

Table 7.3: Default parameter values use in iBioSim and originally disclosed in [21, 79].

Parameter	Symbol	Value
Forward repression binding rate	k_{r_f}	0.5
Reverse repression binding rate	k_{r_r}	1
Forward activation binding rate	k_{a_f}	0.0033
Reverse activation binding rate	k_{a_r}	1
Forward RNAP binding rate	k_{o_f}	0.033
Reverse RNAP binding rate	k_{o_r}	1
Forward activated RNAP binding rate	k_{ao_f}	1
Reverse activated RNAP binding rate	k_{ao_r}	1
Stoichiometry of binding	n_c	2
Initial RNAP count	n_r	30
Open complex production rate	k_o	0.05
Basal production rate	k_b	1.00E-04
Initial promoter count	n_g	2
Stoichiometry of production	n_p	10
Activated production rate	k_a	0.25
Degradation rate	k_d	0.0075
Forward complex formation rate	k_{c_f}	0.05
Reverse complex formation rate	k_{c_r}	1

discussion with the authors of [14] at BU and MIT, it was concluded that the real values of the parameters (shown in Table 7.3) cannot be determined easily through the available experimental setup.

As demonstrated in Chapter 4, the timings of genetic circuit depends on the value of *degradation rate* (k_d), therefore we first made an attempt to derive a relation between degradation rate (k_d) of the genetic circuit components and the response function parameters (see equation 7.3) disclosed in [14].

7.2.1 Relation between iBioSim default parameters and Cello response parameters

Consider the circuit schematic and the SBOLv diagram of a two-input genetic AND gate (obtained from [14]) shown in Figure 7.7(a) and (b), respectively. The promoters P_{Tac} and P_{Tet} control the production of $BM3R1$ and $SrpR$ proteins respectively. The generated proteins $BM3R1$ and $SrpR$ in turn repress their respective output promoters P_{BM3R1} and P_{SrpR}, respectively. When the activities of promoters P_{BM3R1} and P_{SrpR} are suppressed, the production of protein $PhlF$ will be suppressed and thus the final output promoter P_{PhlF} will generate YFP protein. The simplest form of this circuit model is shown in Figure 7.7(c). This figure indicates that the input promoters P_{Tac} and P_{Tet} suppresses the promoters P_{BM3R1} and P_{SrpR}, respectively. When these two promoters are suppressed, they will not produce the output protein $PhlF$ to suppress the corresponding promoter P_{PhlF}. When the promoter P_{PhlF} is not suppressed, it will produce the output YFP protein.

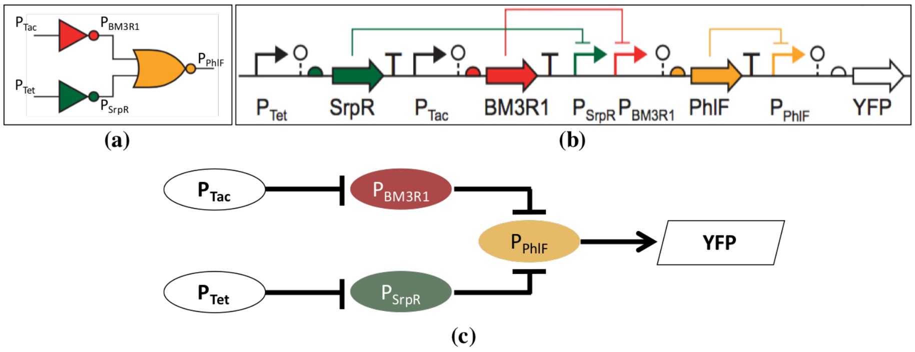


Figure 7.7: The genetic AND gate shown in [14]. (a) Circuit schematic. (b) SBOLv diagram. (c) Simplified model.

Lets assume that the degradation rates (k_d) of all genetic components are the same, and the rate of production is k_o, then the rate of change in P_{BM3R1} can be defined using Hill function [80] as

$$\frac{dP_{BM3R1}}{dt} = \frac{k_o \times K_r^n}{K_r^n + P_{Tac}^n} - k_d \times P_{BM3R1} \tag{7.1}$$

where:

K_r = Repression coefficient i.e. the rate at which P_{Tac} represses P_{BM3R1}
n = Hill coefficient, which controls the steepness of the switch between full to zero repression.
k_d = degradation rate.

At steady state;

$$\frac{dP_{BM3R1}}{dt} = 0$$

Therefore, equation 7.1 $\implies$

$$\frac{k_o \times K_r^n}{K_r^n + P_{Tac}^n} - k_d \times P_{BM3R1} = 0$$

or

$$P_{BM3R1} = \frac{(k_o \times K_r^n)/k_d}{K_r^n + P_{Tac}^n}$$

or

$$P_{BM3R1} = \frac{k_o/k_d}{1 + (P_{Tac}/K_r)^n} \tag{7.2}$$

In [14], the empirical response function with a relation between input and output promoter activities is also defined by a Hill function, as shown in equation 7.3.

$$y = y_{min} + \frac{(y_{max} - y_{min})K^n}{x^n + K^n} \tag{7.3}$$

where:

y	= Output promoter activity
x	= Input promoter activity
y_{max}	= the maximum observed promoter output value.
y_{min}	= the minimum observed promoter output value.
K	= Repression threshold (the input value at which the output is half maximum).
n	= Hill coefficient.

Lets assume that the value of y_{min} is too small as compared to the second term in equation 7.3. Thus, ignoring the first y_{min} term in equation 7.3 $\implies$

$$y = \frac{(y_{max} - y_{min})K^n}{x^n + K^n}$$

or

$$y = \frac{(y_{max} - y_{min})}{1 + (x/K)^n} \tag{7.4}$$

Hence for the output promoter $y = P_{BM3R1}$ and input promoter $x = P_{Tac}$, equation 7.4 $\implies$

$$P_{BM3R1} = \frac{(P_{BM3R1_{max}} - P_{BM3R1_{min}})}{1 + (P_{Tac}/K)^n} \tag{7.5}$$

Equating equations 7.2 and 7.5 $\implies$

$$\frac{k_o/k_d}{1+(P_{Tac}/K_r)^n} = \frac{(P_{BM3R1_{max}} - P_{BM3R1_{min}})}{1+(P_{Tac}/K)^n} \tag{7.6}$$

or

$$k_d = \frac{k_o}{P_{BM3R1_{max}} - P_{BM3R1_{min}}} \tag{7.7}$$

In general,

$$k_d = \frac{k_o}{y_{max} - y_{min}} \tag{7.8}$$

Also from equation 7.6,

$$K_r = K \tag{7.9}$$

$$n = n \tag{7.10}$$

Hence, the degradation rate (k_d) is inverse to the difference of the maximum and minimum activities of the output promoter; and the repression threshold K is equivalent to the rate at which the input promoter represses the output promoter K_r. The Hill coefficients values are also equivalent which means that the values of n for respective promoters, shown in Figure 7.10, can be used in place of the default value of n_c shown in Table 7.3.

7.2.2 Modification of kinetic laws in D-VASim

In Cello [14], the circuit components are chosen from the library using heuristic approach (via simulated annealing algorithm), therefore the generated circuit for same functionality may have different circuit components everytime when the circuit is run. The original model of the genetic AND gate shown in [14] indicates that the input promoter P_{Tac} generates $BM3R1$ protein which suppresses its corresponding output promoter P_{BM3R1}, as shown in Figure 7.7(a) and (b).

However, when we ran the same circuit on Cello again, the heuristic approach chose to map the inverter based on $AmtR$ protein instead of $BM3R1$ protein. This means that input promoter P_{Tac} generates $AmtR$ protein which suppresses its corresponding output promoter P_{AmtR}. The SBOL-SBML converted diagram of this model, using SBOL-SBML conversion plugin in iBioSim [61], is shown in Figure 4.9. This figure also includes the input inducers (see Section

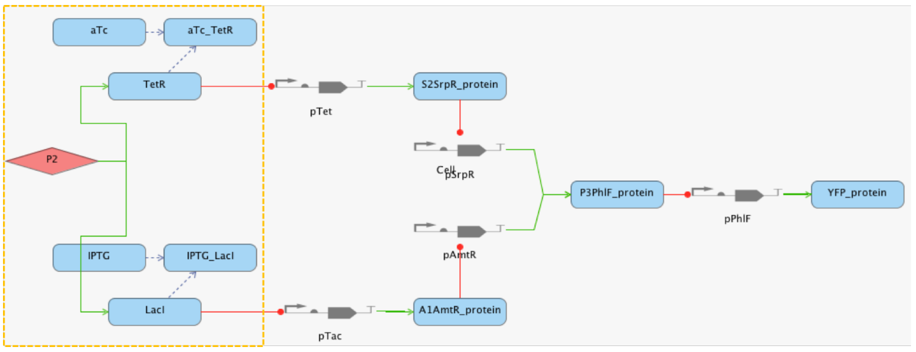
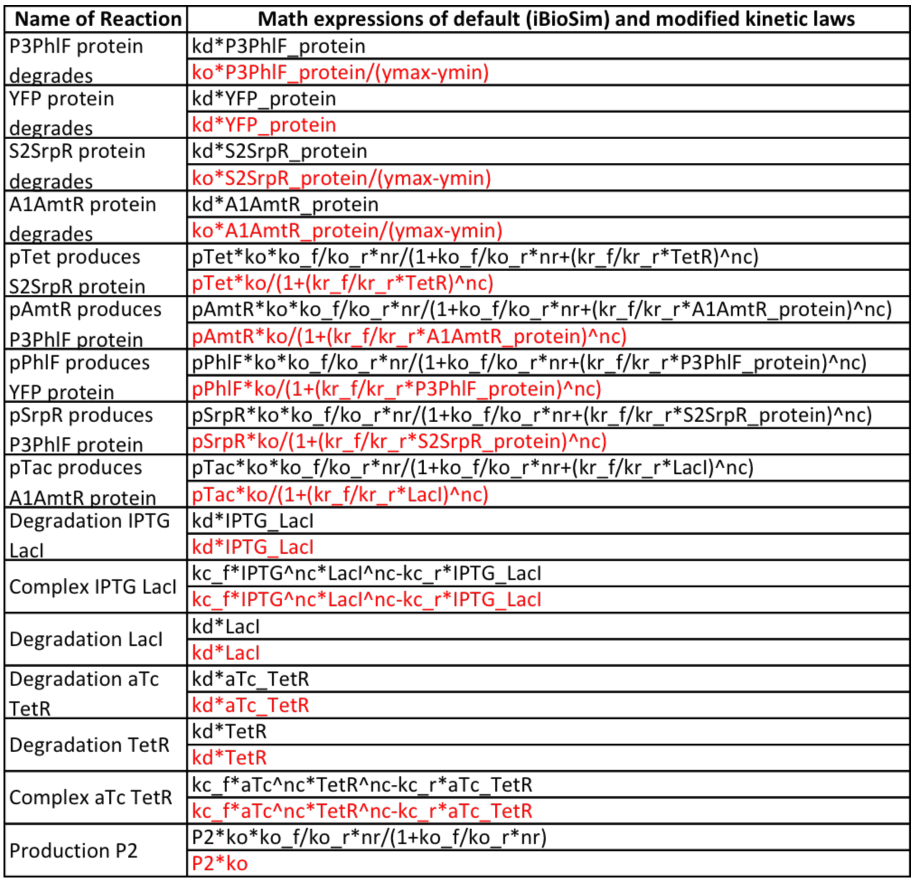


Figure 7.8: SBML design of the genetic AND gate circuit [14] in iBioSim.

4.2.6 for more details). For convenience, Figure 4.9 is reproduced here again as Figure 7.8.

The default and the modified kinetic laws of genetic AND gate model are shown in Figure 7.9. For simplification, the RNAP related parameters (ko_f, ko_r and n_r) shown in default expressions are ignored in the modified kinetic laws (shown in red). The values of $ymax$, $ymin$, kr_r and n_c for respective promoters can be obtained from the Cello *UCF* file. These values, for all available promoters, are shown in Figure 7.10. We used the default iBioSim values for the remaining parameters k_o, k_{r_f} and k_{c_r}. The y_{min} and y_{max} values given in [14] describe the input and output promoter activities in the circuit and not in the input and output sensor blocks. Therefore, the iBioSim default values are used for the parameters involved in the input and output blocks. The output block consists of YFP protein, and the input block consists of IPTG, aTc, LacI, IPTG-LacI complex, TetR, and aTc-TetR complex.

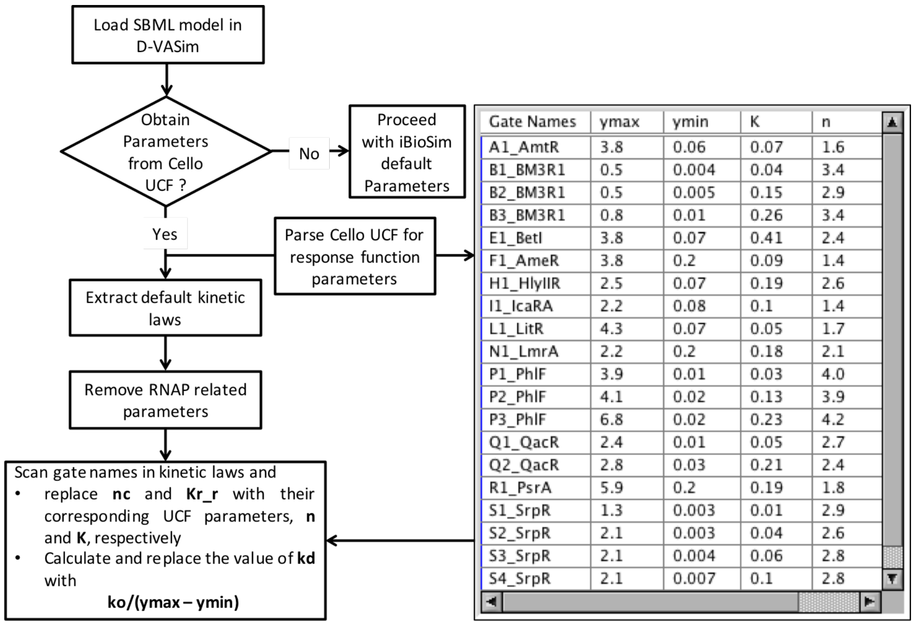
Figure 7.10 shows the high-level integration flow of D-VASim and Cello using Cello UCF and iBioSim simulation parameters. Once the SBOL model of any genetic circuit [14] is converted to SBML model using [61], it can be loaded into D-VASim. A user can decide to proceed with analyzing the model in D-VASim either using iBioSim tool's default parameters or the parameters from Cello UCF.



Name of Reaction	Math expressions of default (iBioSim) and modified kinetic laws
P3PhlF protein degrades	kd*P3PhlF_protein
	ko*P3PhlF_protein/(ymax-ymin)
YFP protein degrades	kd*YFP_protein
	kd*YFP_protein
S2SrpR protein degrades	kd*S2SrpR_protein
	ko*S2SrpR_protein/(ymax-ymin)
A1AmtR protein degrades	kd*A1AmtR_protein
	ko*A1AmtR_protein/(ymax-ymin)
pTet produces S2SrpR protein	pTet*ko*ko_f/ko_r*nr/(1+ko_f/ko_r*nr+(kr_f/kr_r*TetR)^nc)
	pTet*ko/(1+(kr_f/kr_r*TetR)^nc)
pAmtR produces P3PhlF protein	pAmtR*ko*ko_f/ko_r*nr/(1+ko_f/ko_r*nr+(kr_f/kr_r*A1AmtR_protein)^nc)
	pAmtR*ko/(1+(kr_f/kr_r*A1AmtR_protein)^nc)
pPhlF produces YFP protein	pPhlF*ko*ko_f/ko_r*nr/(1+ko_f/ko_r*nr+(kr_f/kr_r*P3PhlF_protein)^nc)
	pPhlF*ko/(1+(kr_f/kr_r*P3PhlF_protein)^nc)
pSrpR produces P3PhlF protein	pSrpR*ko*ko_f/ko_r*nr/(1+ko_f/ko_r*nr+(kr_f/kr_r*S2SrpR_protein)^nc)
	pSrpR*ko/(1+(kr_f/kr_r*S2SrpR_protein)^nc)
pTac produces A1AmtR protein	pTac*ko*ko_f/ko_r*nr/(1+ko_f/ko_r*nr+(kr_f/kr_r*LacI)^nc)
	pTac*ko/(1+(kr_f/kr_r*LacI)^nc)
Degradation IPTG LacI	kd*IPTG_LacI
	kd*IPTG_LacI
Complex IPTG LacI	kc_f*IPTG^nc*LacI^nc-kc_r*IPTG_LacI
	kc_f*IPTG^nc*LacI^nc-kc_r*IPTG_LacI
Degradation LacI	kd*LacI
	kd*LacI
Degradation aTc TetR	kd*aTc_TetR
	kd*aTc_TetR
Degradation TetR	kd*TetR
	kd*TetR
Complex aTc TetR	kc_f*aTc^nc*TetR^nc-kc_r*aTc_TetR
	kc_f*aTc^nc*TetR^nc-kc_r*aTc_TetR
Production P2	P2*ko*ko_f/ko_r*nr/(1+ko_f/ko_r*nr)
	P2*ko

Figure 7.9: The default and modified kinetic laws of genetic AND gate model. The modified kinetic laws are shown in *red* under each of their corresponding default kinetic laws, shown in *black*.

If an option "*Obtain parameters from Cello UCF*", shown in Figure 3.2, is marked checked, D-VASim first parse the Cello UCF file. It then extracts the default kinetic laws and remove RNAP related parameters. Then it scans for the promoter names in kinetic laws and replace the values of n_c and K_{r_r} with the corresponding values of n and K respectively from Cello UCF parameters. In the very end, it calculates and replaces the value of k_d with $k_o/(y_{max} - y_{min})$; where y_{max} and y_{min} are the maximum and minimum activities of the corresponding promoters; and k_o is the rate of production. Since the actual value of k_o is not known, therefore D-VASim uses its default value (see Table 7.3).


Load SBML model in D-VASim
Obtain Parameters from Cello UCF ?
No
Proceed with iBioSim default Parameters
Yes
Parse Cello UCF for response function parameters
Extract default kinetic laws
Remove RNAP related parameters
Scan gate names in kinetic laws and
• replace **nc** and **Kr_r** with their corresponding UCF parameters, **n** and **K**, respectively
• Calculate and replace the value of **kd** with
ko/(ymax – ymin)

Gate Names	ymax	ymin	K	n
A1_AmtR	3.8	0.06	0.07	1.6
B1_BM3R1	0.5	0.004	0.04	3.4
B2_BM3R1	0.5	0.005	0.15	2.9
B3_BM3R1	0.8	0.01	0.26	3.4
E1_BetI	3.8	0.07	0.41	2.4
F1_AmeR	3.8	0.2	0.09	1.4
H1_HlyIIR	2.5	0.07	0.19	2.6
I1_IcaRA	2.2	0.08	0.1	1.4
L1_LitR	4.3	0.07	0.05	1.7
N1_LmrA	2.2	0.2	0.18	2.1
P1_PhlF	3.9	0.01	0.03	4.0
P2_PhlF	4.1	0.02	0.13	3.9
P3_PhlF	6.8	0.02	0.23	4.2
Q1_QacR	2.4	0.01	0.05	2.7
Q2_QacR	2.8	0.03	0.21	2.4
R1_PsrA	5.9	0.2	0.19	1.8
S1_SrpR	1.3	0.003	0.01	2.9
S2_SrpR	2.1	0.003	0.04	2.6
S3_SrpR	2.1	0.004	0.06	2.8
S4_SrpR	2.1	0.007	0.1	2.8

Figure 7.10: The integration flow of Cello UCF parameters and D-VASim.

7.2.3 Experimentation by Simulation

We performed simulations on the model of a genetic AND gate, obtained from [14] (Figure 7.7), using both the iBioSim's default and the modified kinetic laws (shown in Figure 7.9). The intermediate species used in this circuit are *SrpR*, *AmtR* and *PhlF*. The promoters corresponding to these species are used with the specific *ribosome binding sites* (RBS) denoted by $S2$, $A1$, and $P3$ for *SrpR*, *AmtR* and *PhlF*, respectively, as shown in Figure 7.9. Each of the RBS for their respective promoters have different values of y_{max}, y_{min}, K, and n, as shown in Figure 7.10. For example, it can be noticed in Figure 7.10 that the production of *PhlF* protein is possible with three different RBS i.e., $PhlF\_P1$, $PhlF\_P2$, and $PhlF\_P3$. Similarly, there are four possibilities for *SrpR* protein, and so on.

The analog and digital simulations plots of the genetic AND gate (in D-VASim), with default iBioSim's and the modified kinetic laws, are shown in Figure 7.11 and 7.12, respectively. The threshold value for all species are assumed to be 12 molecules in both cases. Also, instead of applying all possible input combinations in these experiments, only the input combination logic 11 (both inputs

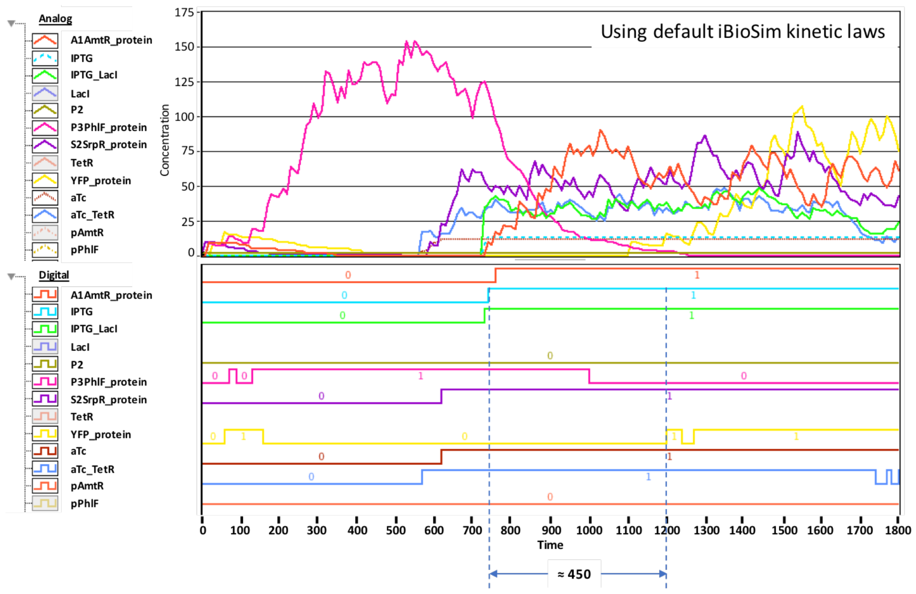


Figure 7.11: D-VASim simulation plots of the AND gate, using the default kinetic laws of iBioSim.

HIGH) is applied to observe the circuit's response. From equation 7.8, using the default value of $k_o = 0.05$ and the values of y_{max} and y_{min} shown in Figure 7.10, the degradation rates of $AmtR$, $SrpR$ and $PhlF$ proteins are found to be 0.0133, 0.0239 and 0.0073, respectively. Also the (K, n) values used for $AmtR$, $SrpR$ and $PhlF$ are, in order, $(0.07, 1.6)$, $(0.04, 2.6)$ and $(0.23, 4.2)$.

Due to the changes in these values, the most prominent effect which can be noticed, in Figure 7.12, is that the propagation delay of the circuit is increased when simulated with the modified kinetic laws and real parameter values. Also, 3.18x increase in the degradation rate of $SrpR$ (in comparison to its default value) makes it noisy and oscillate around logic 0 and 1, as shown in Figure 7.12. It is also noticed that the degradation rate of $PhlF$ protein (with $P3$ RBS) is found almost similar to its default value i.e., 0.0073. However, the concentration of $PhlF$ protein, during the first 700 time units, is found almost doubled when simulated with the modified kinetic laws. This is due to the higher value of n_c, i.e., 4.2, obtained for the $P3$ RBS used in the production of $PhlF$ protein. These observations suggests that the functionality of a circuit, i.e., AND logic, remains the same, however the behavior of species and their timings are changed.

The video demonstration of this approach can be seen at `https://www.youtube.`

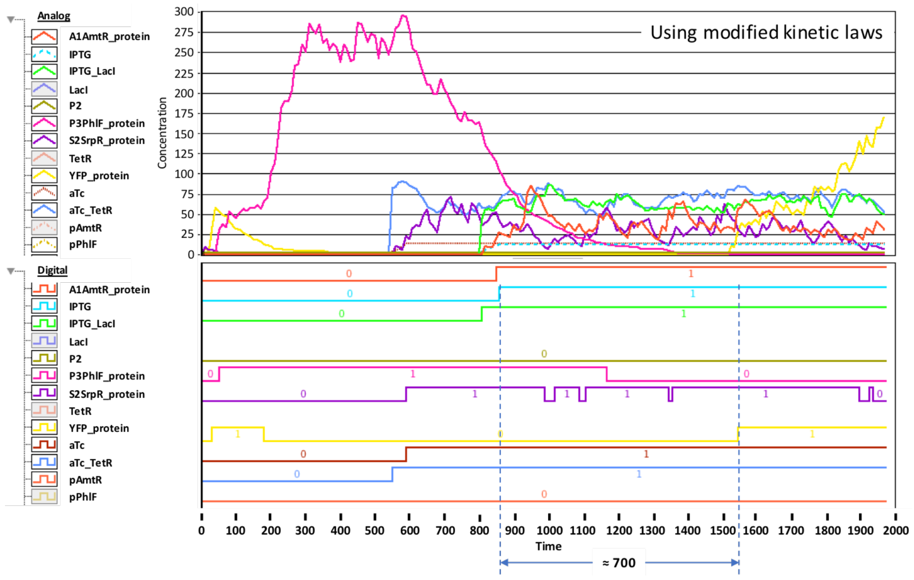


Figure 7.12: D-VASim simulation plots of the AND gate, using the modified kinetic laws.

com/watch?v=9Ds12Qb6PL4. In this video, the genetic AND gate circuit is simulated without the input sensor block. That is the inputs are applied by varying the activities of input promoters, P_{Tac} and P_{Tet}.

7.3 Discussion

This chapter is an extended and ongoing work of this thesis in which the sensitivity of model's parameters on its behavior is analyzed. First, an experimental workflow for checking genetic circuit models using statistical model checking and stochastic simulation is proposed. Two different-sized genetic circuit models [21] are used to demonstrate that the proposed workflow can be applied for the timing and threshold values analyses of any genetic circuit model. We varied the design parameters of the genetic circuits and checked their probabilities of working correctly. Furthermore, we analyzed the effects of changing design parameters on the behavior of a given circuit. The proposed workflow can be used to check any other property of a genetic circuit; such as the probability of a circuit to reach a certain state within a specific amount of time.

Next, we simulate the SBML model of a real genetic AND gate [14] using the real parameter values. To achieve this, we derived a relationship between the parameters used in iBioSim during the SBOL-SBML conversion process [21, 79] and the response function parameters mentioned in [14]. There are several important factors which have to be considered in order to simulate the model's behavior close to reality. The parameter values and their units use in simulation should be real (or at least their scaling factor should be known). For example, in case of the genetic AND gate simulation with the modified kinetic laws, the propagation delay was found to be approximately 700 time units. There must be a scaling factor, which should convert this 700 time units into a real time value. It is also important to know if the derived relationships (equation 7.8 - 7.10) holds true for more complex circuits. The effort made to derive the above mentioned relations is the very initial attempt to relate real parameter values with simulation parameters. We believe that simulating the circuit models with real parameters will improve the model's reliability and will help debugging the circuits more effectively.

CHAPTER 8

Conclusions and Future Work

This chapter summarizes the work presented in this dissertation and indicates some interesting directions in which this work can be extended.

8.1 Summary and Conclusions

The primary subject of this thesis is to present methods, algorithms, and automated tools for the simulation, analysis, verification and synthesis of genetic logic circuits.

First, a simulation tool, *D-VASim*, is developed, which allows a user to perform interactive run-time experiments in a virtual laboratory environment. It is solely designed for the simulation and analysis of genetic logic circuits, however other SBML-based models can also be analyzed. This tool has been extensively tested with several genetic circuit models presented in [21] and [14]. The experimental results presented in this thesis suggest that the concept of virtual experimentation may help users to perform more insightful *in-silico* experiments.

Furthermore, a methodology is proposed to perform the timing analysis of genetic logic circuits. This methodology is implemented as a plug-in tool in D-VASim. With the help of this feature, a user can analyze the input threshold value and the propagation delay of a genetic circuit. The effects of varying circuit parameters on the threshold value and propagation delay of different genetic circuits have been analyzed. We observed that the threshold value, propagation delay, and the design parameter (like degradation rate, k_d) are all interlinked. The propagation delay of a genetic circuit is inversely related to k_d. The output of a circuit becomes stable and unstable for lower and higher values of k_d, respectively. The large values of k_d also contributes in the reduction of threshold value because, for high k_d values, a circuit becomes faster and thus a small input threshold concentration is enough to trigger its output. It has also been observed that the value of k_d cannot be increased beyond a certain point as it makes the circuit's output unstable and thus distorts the intended functionality (see Chapter 4). We believe that the introduction of timing analysis of genetic logic circuits will be an interesting domain for further research. Similar to electronic circuits, the timing analysis of genetic circuit may become an essential design characteristic when it is required to make sure that the circuit produces the desired output within the specified interval of time.

Moreover, D-VASim with its support of automatic Boolean logic extraction has the potential to improve the productivity of researchers for the logic analysis of bio-models. With this capability, a user may not only be able to verify the input-output relation of genetic logic circuits but is also able to discover the hidden Boolean logic that exists in any other SBML-based biological model. We demonstrated that the algorithm estimates the correct logic of genetic circuits in terms of boolean expression along with its percentage fitness in the experimental data. We performed experimentation on some of the genetic circuit models disclosed by Nielsen *et al.* in [14]. These circuits are generated by an automated tool named *Cello*. We observed that the circuits shown in [14] are structurally as well as *behaviorally* different when they are generated again from Cello. The behavioral changes in these genetic circuits have been correctly verified through the proposed logic analysis algorithm in D-VASim. Apart from this, it has also been observed that the logical behavior of a genetic circuit seems to be effected by variations in the threshold values (see Chapter 5). The evaluated performance of the algorithm indicates that it took about 8.4 seconds to estimate the logic of a complex genetic circuit (having 3 inputs with up to 7 genetic gates).

Another tool, named *GeneTech*, is also developed which allows a user to synthesize genetic circuits with very high level description i.e. by specifying the Boolean expression only. *GeneTech* search for the right biological components from the genetic gates library [14] and gives a number of different possible genetic circuits to achieve the same functionality. This tool has also been experimented with a number of genetic circuit models presented in [14]. It has been shown that the *GeneTech* tool, using the genetic gates library, lists down all the possible solutions to achieve the desired functionality.

In addition to the abovementioned contributions of this thesis, an experimental workflow is also proposed for model checking of genetic circuits using Uppaal and D-VASim. Moreover, a methodology is presented to perform simulation using the real parameter values. To achieve this, efforts has been made to relate the iBioSim default parameters with the Cello response function parameters. We demonstrated how the behavior of a model changes using real parameter values. Though we have also included the video demonstration but this feature is not yet made available for public use, because it is an on-going work.

Ultimately, we believed that the tools and methods presented in this thesis can contribute in the advancement of Bio Design Automation industry. It is further hoped that the synthetic biologists and/or engineers could use these tools to design and analyze genetic circuits more efficiently.

8.2 Research Impact

D-VASim was released for public use in March 2016, and after being accepted for publication in Bioinformatics Journal in September 2016, it has been exposed to the synthetic biology community. It has been observed that D-VASim has already started to create impact on research world wide in a short span of time. The download metrics of D-VASim are mentioned below:

- Accessed from 54 different countries (202 different cities) across the globe
- Downloaded **434** times for Windows OS
- Downloaded **445** times for Mac OS
- Quick Start Guide (QSG) has been download **491** times

Figure 8.1 shows some other metrics about D-VASim obtained from google analytics for the period of March 01, 2016 - September 25, 2017. This figure shows that during the span of past 19 months, D-VASim download page has been accessed 3523 times by 884 users. Out of these 884 users, 74.28 percent of them are new users. Also, the highest number of sessions (i.e. users actively engaged to website) is made from United Kingdom.

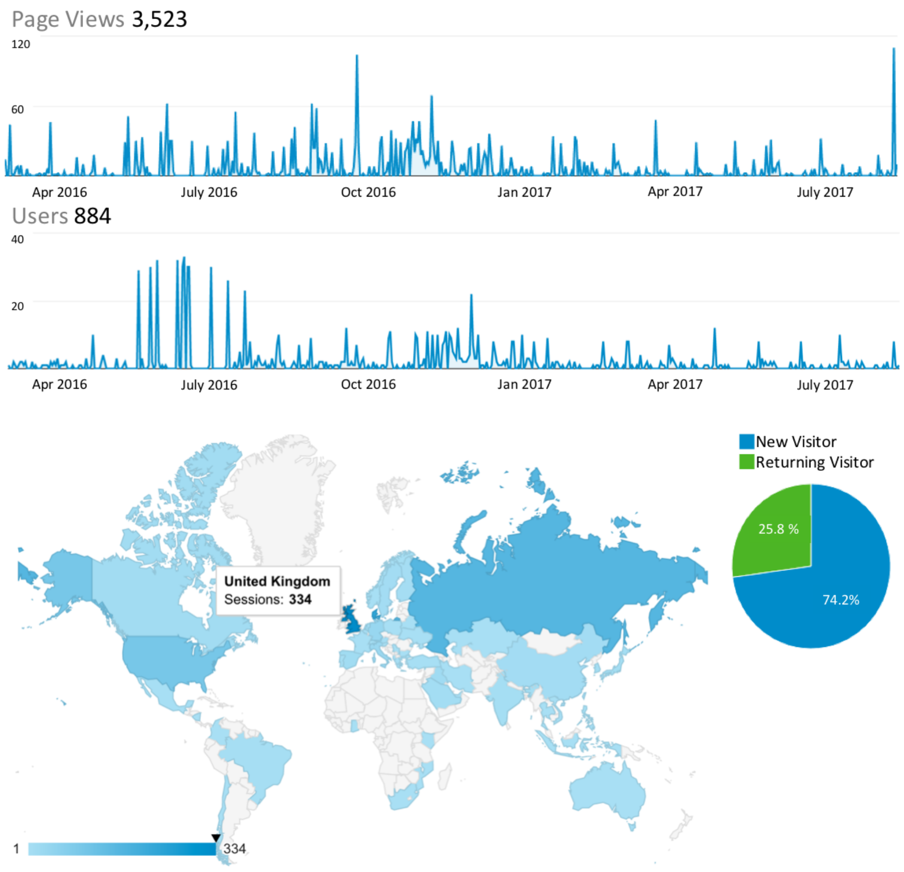


Figure 8.1: D-VASim download metrics during Mar 1, 2016 - Sept. 25, 2017.

8.3 Future Work

There are substantial possibilities to extend the work which has been presented in this thesis. These possibilities are categorized into *experimental* and *implementation* viewpoints.

8.3.1 Extension from an experimental viewpoint

The experimentation on these tools can be performed with more genetic circuit models. For example, as it is stated in Chapter 3 that the ODE simulation is tested with the first 400 cases of the SBML benchmark suite. After upgrading D-VASim to incorporate the remaining SBML components, other cases of SBML benchmark suite can be tested.

The experimentation related to timing analysis (see Chapter 4) is performed on the genetic circuit models proposed by Myers [21]. It can be further extended to analyze the timings of real genetic circuits developed by Alec A. K. Nielsen *et al.* [14]. Furthermore, the timing analyses of the separate components of each individual circuit can be performed to characterize them into parts library. This will help further in selecting the right genetic components while constructing a genetic circuit to meet the timing requirements.

Similarly, the technology mapping tool *GeneTech* can be tested with the remaining genetic circuits given in [14], for which it is required to incorporate the capability of supporting the multi-level output circuits.

The methodology of relating simulation parameters with the real ones (Cello UCF) can also be extended and tested on more complex genetic circuit models.

8.3.2 Extension from an implementation viewpoint

D-VASim and *GeneTech* both has the potential to be enhanced in many possible ways. Some of them are given below:

- **SBML Packages Support**
  D-VASim can be upgraded to support the SBML packages given at `http://sbml.org/Documents/Specifications/SBML_Level_3/Packages`.

- **SED-ML Support**
  The support of SED-ML (Simulation Experiment Description Markup Language) [81] can be included in D-VASim to exchange the descriptions of interactive simulation of computational models. This typically would be an important advancement in D-VASim because it supports dynamic user interaction with simulation.

- **Speeding up analysis on parallel processors**
  As mentioned in Chapter 4 that, depending on the parameter values, the timing analysis for complex circuits can take up to an hour. To speed-up the analysis process, the functionality can be included in D-VASim to run the algorithm on parallel processors, typically on embedded graphics processor units (GPUs).

- **Upgrades in *GeneTech***
  GeneTech is currently a prototype and immature tool. To make it user friendly, its GUI has to be developed. It could either be done on JAVA to make it a standalone tool, or it can be integrated as a plug-in to D-VASim.

  GeneTech can be further enhanced to process multi-output circuits. It requires upgrading the existing classes to process multiple expressions. Also, more design constraints can be added to increase the chances that the generated circuits would work correctly in the laboratory.

  The number of produced solutions may increase with the increase in size of gates library. To avoid this scenario, the design constraints can be defined to filter out the *bad-solutions*. Though, *GeneTech* filters out the *bad solutions* if it contains any unintended feedback loops (see description of Figure 6.10). Similarly, other constraints can be added to avoid having long list of solutions. For instance, the solution may also be categorized as a *bad solution* if the number of logical components in it is greater than the one specified by user.

  Furthermore, the *GeneTech* tool, at its current state, supports technology mapping of genetic gates based on repression. In future, it will be upgraded to support genetic gates based on other technologies, such as activation.

- **SBOL export** The current version of *GeneTech* uses multi-line text string to represent the structure of a genetic circuit. This is not a standard way to represent genetic circuits. It is therefore important to output circuit description in a standardized form, such as SBOL data or SBOLv.

Bibliography

[1] A. Arkin, "Setting the standard in synthetic biology," *Nature Biotechnology*, vol. 26, no. 7, pp. 771–774, 2008.

[2] J. J. Collins, T. S. Gardner, and C. R. Cantor, "Construction of a genetic toggle switch in Escherichia coli," *Nature*, vol. 403, no. 6767, pp. 339–342, 2000.

[3] R. W. Basu and Subhayu, "The device physics of cellular logic gates," *NSC-1: The First Workshop of Non-Silicon Computing*, vol. 158, pp. 39–41, 2002.

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

[5] S. Atsumi and J. C. Liao, "Metabolic engineering for advanced biofuels production from Escherichia coli," *Current Opinion in Biotechnology*, vol. 19, no. 5, pp. 414–419, 2008.

[6] D.-k. Ro, E. M. Paradise, M. Ouellet, K. J. Fisher, K. L. Newman, J. M. Ndungu, K. A. Ho, R. A. Eachus, T. S. Ham, J. Kirby, M. C. Y. Chang, S. T. Withers, Y. Shiba, R. Sarpong, and J. D. Keasling, "Production of the antimalarial drug precursor artemisinic acid in engineered yeast," *Nature*, vol. 440, no. 7086, pp. 940–943, 2006.

[7] M. A. Marchisio and J. Stelling, "Computational design tools for synthetic biology," *Current Opinion in Biotechnology*, vol. 20, no. 4, pp. 479–485, 2009.

[8] H. de Jong, “Modeling and Simulation of Genetic Regulatory Systems: A Literature Review,” *Journal of Computational Biology*, vol. 9, no. 1, pp. 67–103, 2002.

[9] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Nov??re, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang, “The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models,” *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.

[10] B. Bartley, J. Beal, K. Clancy, G. Misirli, N. Roehner, E. Oberortner, M. Pocock, M. Bissell, C. Madsen, T. Nguyen, Z. Zhang, J. H. Gennari, C. Myers, A. Wipat, and H. Sauro, “Synthetic Biology Open Language (SBOL) Version 2.0.0.,” *Journal of integrative bioinformatics*, vol. 12, no. 2, p. 272, 2015.

[11] H. McAdams and L. Shapiro, “Circuit simulation of genetic networks,” *Science*, vol. 269, no. 5224, pp. 650–656, 1995.

[12] D. Bernardi, J. T. Dejong, B. M. Montoya, and B. C. Martinez, “Biobricks: Biologically cemented sandstone bricks,” *Construction and Building Materials*, vol. 55, pp. 462–469, 2014.

[13] “SBML Software Matrix.” `http://sbml.org/SBML_Software_Guide/SBML_Software_Matrix`, 2010.

[14] A. A. K. Nielsen, B. S. Der, J. Shin, P. Vaidyanathan, V. Paralanov, E. A. Strychalski, D. Ross, D. Densmore, and C. A. Voigt, “Genetic circuit design automation,” *Science*, vol. 352, no. 6281, pp. aac7341–aac7341, 2016.

[15] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, “UPPAAL-a tool suite for automatic verification of real-time systems,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1066, pp. 232–243, 1996.

[16] H. Baig and J. Madsen, “D-VASim: an interactive virtual laboratory environment for the simulation and analysis of genetic circuits,” *Bioinformatics*, vol. 33, no. 2, pp. 297–299, 2017.

[17] C. J. Myers, N. Barker, K. Jones, H. Kuwahara, C. Madsen, and N. P. D. Nguyen, "iBioSim: A tool for the analysis and design of genetic circuits," *Bioinformatics*, vol. 25, no. 21, pp. 2848–2849, 2009.

[18] K. G. L. A. L. M. M. Peter Bulychev Alexandre David and D. B. Poulsen, "Checking {&}amp; Distributing Statistical Model Checking," in *4th NASA Formal Methods Symposium*, pp. 449–463, LNCS 7226, Springer, 2012.

[19] F. CRICK, "Central Dogma of Molecular Biology," *Nature*, vol. 227, no. 5258, pp. 561–563, 1970.

[20] F. JACOB and J. MONOD, "Genetic regulatory mechanisms in the synthesis of proteins.," *Journal of molecular biology*, vol. 3, pp. 318–56, 1961.

[21] C. J. Myers, *Engineering Genetic Circuits.* Chapman & Hall/CRC Press, 2009.

[22] D. Chandran, F. T. Bergmann, and H. M. Sauro, "TinkerCell: modular CAD tool for synthetic biology," *Journal of Biological Engineering*, vol. 3, no. 1, p. 19, 2009.

[23] J. Beal, T. Lu, and R. Weiss, "Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks," *PLoS ONE*, vol. 6, no. 8, 2011.

[24] C. Madsen, C. J. Myers, T. Patterson, N. Roehner, J. T. Stevens, and C. Winstead, "Design and test of genetic circuits using iBioSim," *IEEE Design and Test of Computers*, vol. 29, no. 3, pp. 32–39, 2012.

[25] J. Chen, D. Densmore, T. S. Ham, J. D. Keasling, and N. J. Hillson, "DeviceEditor visual biological CAD canvas," *Journal of Biological Engineering*, vol. 6, no. 1, p. 1, 2012.

[26] Y. Cai, M. L. Wilson, and J. Peccoud, "GenoCAD for iGEM: A grammatical approach to the design of standard-compliant constructs," *Nucleic Acids Research*, vol. 38, no. 8, pp. 2637–2644, 2010.

[27] G. Misirli, J. S. Hallinan, T. Yu, J. R. Lawson, S. M. Wimalaratne, M. T. Cooling, and A. Wipat, "Model annotation for synthetic biology: Automating model to nucleotide sequence conversion," *Bioinformatics*, vol. 27, no. 7, pp. 973–979, 2011.

[28] A. Villalobos, J. E. Ness, C. Gustafsson, J. Minshull, and S. Govindarajan, "Gene Designer: a synthetic biology tool for constructing artificial DNA segments.," *BMC bioinformatics*, vol. 7, p. 285, 2006.

[29] T. S. Ham, Z. Dmytriv, H. Plahar, J. Chen, N. J. Hillson, and J. D. Keasling, “Design, implementation and practice of JBEI-ICE: An open source biological part registry platform and tools,” *Nucleic Acids Research*, vol. 40, no. 18, 2012.

[30] J. T. MacDonald, C. Barnes, R. I. Kitney, P. S. Freemont, and G.-B. V. Stan, “Computational design approaches and tools for synthetic biology.,” *Integrative biology : quantitative biosciences from nano to macro*, vol. 3, no. 2, pp. 97–108, 2011.

[31] G. S. M. d. B. Heinz Koeppl Douglas Densmore, *Design and Analysis of Biomolecular Circuits: Engineering Approaches to Systems and Synthetic Biology.* Springer Science & Business Media, 2011.

[32] M. Zhang, J. A. McLaughlin, A. Wipat, and C. J. Myers, “SBOLDesigner 2: An Intuitive Tool for Structural Genetic Design,” *ACS Synthetic Biology*, 2017.

[33] S. M. Richardson, S. J. Wheelan, R. M. Yarrington, and J. D. Boeke, “GeneDesign: Rapid, automated design of multikilobase synthetic genes,” *Genome Research*, vol. 16, no. 4, pp. 550–556, 2006.

[34] P. Umesh, F. Naveen, C. U. M. Rao, and A. S. Nair, “Programming languages for synthetic biology,” *Systems and Synthetic Biology*, vol. 4, no. 4, pp. 265–269, 2010.

[35] H. B. F. Dixon, H. Bielka, and C. R. Cantor, “Nomenclature for incompletely specified bases in nucleic acid sequences. Recommendations 1984,” 1986.

[36] A. Funahashi, Y. Matsuoka, A. Jouraku, M. Morohashi, N. Kikuchi, and H. Kitano, “CellDesigner 3.5: A versatile modeling tool for biochemical networks,” *Proceedings of the IEEE*, vol. 96, no. 8, pp. 1254–1265, 2008.

[37] F. Kolpakov, M. Puzanov, and A. Koshukov, *BioUML: visual modeling, automated code generation and simulation of biological systems*, vol. 3. 2006.

[38] S. Hoops, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer, “COPASI - A COmplex PAthway SImulator,” *Bioinformatics*, vol. 22, no. 24, pp. 3067–3074, 2006.

[39] H. Baig and J. Madsen, “Simulation Approach for Timing Analysis of Genetic Logic Circuits,” *ACS Synthetic Biology*, p. acssynbio.6b00296, 2 2017.

[40] G. Rodrigo, J. Carrera, and A. Jaramillo, “Asmparts: Assembly of biological model parts,” *Systems and Synthetic Biology*, vol. 1, no. 4, pp. 167–170, 2007.

[41] M. Pedersen and A. Phillips, “Towards programming languages for genetic engineering of living cells,” *J R Soc Interface*, vol. 6 Suppl 4, pp. 437–50, 2009.

[42] S. Mirschel, K. Steinmetz, M. Rempel, M. Ginkel, and E. D. Gilles, “ProMoT: Modular modeling for systems biology,” *Bioinformatics*, vol. 25, no. 5, pp. 687–689, 2009.

[43] L. P. Smith, F. T. Bergmann, D. Chandran, and H. M. Sauro, “Antimony: A modular model definition language,” *Bioinformatics*, vol. 25, no. 18, pp. 2452–2454, 2009.

[44] A. D. Hill, J. R. Tomshine, E. M. B. Weeding, V. Sotiropoulos, and Y. N. Kaznessis, “SynBioSS: The synthetic biology modeling suite,” *Bioinformatics*, vol. 24, no. 21, pp. 2551–2553, 2008.

[45] S. Jha, E. Clarke, C. Langmead, A. Legay, A. Platzer, and P. Zuliani, “A bayesian approach to model checking biological systems,” *Lecture Notes In Computer Science*, vol. 5688, no. 2005, pp. 218–234, 2009.

[46] E. M. Clarke, J. R. Faeder, C. J. Langmead, L. A. Harris, S. K. Jha, and A. Legay, “Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway,” in *Computational Methods in Systems Biology: 6th International Conference CMSB 2008, Rostock, Germany, October 12-15, 2008. Proceedings*, pp. 231–250, LNCS, Springer, 2008.

[47] J. a. Goler, “BioJADE: A Design and Simulation Tool for Synthetic Biological Systems,” *DSpace MIT*, vol. Master of, no. May, p. 54, 2004.

[48] F. Yaman, S. Bhatia, A. Adler, D. Densmore, and J. Beal, “Automated selection of synthetic biology parts for genetic regulatory networks,” *ACS Synthetic Biology*, vol. 1, no. 8, pp. 332–344, 2012.

[49] L. Huynh, A. Tsoukalas, M. Koppe, and I. Tagkopoulos, “SBROME: A scalable optimization and module matching framework for automated biosystems design,” *ACS Synthetic Biology*, vol. 2, no. 5, pp. 263–273, 2013.

[50] N. Roehner and C. J. Myers, “Directed acyclic graph-based technology mapping of genetic circuit models,” *ACS Synthetic Biology*, vol. 3, no. 8, pp. 543–555, 2014.

[51] Hasan Baig and Jan Madsen, “A Top-down Approach to Genetic Circuit Synthesis and Optimized Technology Mapping,” in *9th IWBDA*, pp. 28–29, 2017.

[52] K. Keutzer, “DAGON: Technology Binding and Local Optimization by DAG Matching,” *24th ACM/IEEE Design Automation Conference*, pp. 341–347, 1987.

[53] H. Baig and J. Madsen, "D-VASim: Dynamic Virtual Analyzer and Simulator for Genetic Circuits," in *7th International Workshop on Bio-Design Automation (IWBDA)*, pp. 48–49, 2015.

[54] C. M. Lloyd, M. D. B. Halstead, and P. F. Nielsen, "CellML: Its future, present and past," in *Progress in Biophysics and Molecular Biology*, vol. 85(2-3), pp. 433–450, 2004.

[55] A. Drager, N. Rodriguez, M. Dumousseau, A. Dorr, C. Wrzodek, N. Le Novere, A. Zell, and M. Hucka, "JSBML: A flexible java library for working with SBML," *Bioinformatics*, vol. 27, no. 15, pp. 2167–2168, 2011.

[56] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *Journal of Computational Physics*, vol. 22, no. 4, pp. 403–434, 1976.

[57] B. Mark, *Complete Digital Design: A comprehensive guide to digital electronic and computer system architecture.* McGraw-Hill Press, 2003.

[58] H. Baig and J. Madsen, "Logic and Timing Analysis of Genetic Logic Circuits using D-VASim," in *8th IWBDA*, pp. 77–78, 2016.

[59] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *The Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.

[60] R. R. K. C. R. P. P. Visveswara Rao B. Bhaskara Rama Murty K., *Electronic devices and circuits.* Pearson Education, 2 ed., 2007.

[61] N. Roehner, Z. Zhang, T. Nguyen, and C. J. Myers, "Generating Systems Biology Markup Language Models from the Synthetic Biology Open Language," *ACS Synthetic Biology*, vol. 4, no. 8, pp. 873–879, 2014.

[62] H. Baig and J. Madsen, "Logic Analysis and Verification of n-input Genetic Logic Circuits," in *Design Automation and Test in Europe (DATE) 2017*, (Lausanne), pp. 654–657, IEEE, 2017.

[63] Hasan Baig and Jan Madsen, "GeneTech: A Technology Mapping Tool for Genetic Logic Circuits." 2017.

[64] Giovanni De Micheli, "Synthesis and Optimization of Digital Circuits," in *Synthesis and Optimization of Digital Circuits*, p. 576, 1994.

[65] P. Färm, *Integrated Logic Synthesis Using Simulated Annealing.* PhD Thesis, 2007.

[66] S. Kirkpatrick, C. Gelatt Jr, and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[67] Hasan Baig and Jan Madsen, "Taming Living Logic using Formal Methods," in *Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday* (L. Aceto, G. Bacci, G. Bacci, A. Ingoottir, A. Legay, and R. Mardare, eds.), pp. 489–501, Springer (LNCS), 2017.

[68] M. Calder, S. Gilmore, and J. Hillston, "Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA," *Transactions on Computational Systems Biology VII*, vol. 4230, pp. 1–23, 2006.

[69] M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton, "Analysis of signalling pathways using the PRISM model checker," *Proc. Computational Methods in Systems Biology (CMSB'05)*, 2005.

[70] L. Cardelli, "Abstract machines of systems biology," *Transactions on Computational Systems Biology Ii*, vol. 3737, pp. 145–168, 2005.

[71] J. Fisher, N. Piterman, E. J. A. Hubbard, M. J. Stern, and D. Harel, "Computational insights into Caenorhabditis elegans vulval development," *Proceedings of the National Academy of Sciences*, vol. 102, no. 6, pp. 1951–1956, 2005.

[72] L. Calzone, N. Chabrier-Rivier, François Fages, and S. Soliman, "Transactions on Computational Systems Biology VI," 2006.

[73] N. Chabrier and F. Fages, "Symbolic model checking of biochemical networks," in *Proceedings of the First International Workshop on Computational Methods in Systems Biology (CMSB 2003)*, vol. 2602, pp. 149–162, 2003.

[74] M. Kwiatkowska, G. Norman, D. Parker, O. Tymchyshyn, J. Heath, and E. Gaffney, "Simulation and Verification for Computational Modelling of Signalling Pathways," in *WSC '06: Proceedings of the 38th conference on Winter simulation*, pp. 1666–1674, IEEE, 2006.

[75] C. J. Langmead and S. K. Jha, "Predicting protein folding kinetics via model checking," in *The 7th Workshop on Algorithms in Bioinformatics (WABI)*, pp. 252–264, Lecture Notes in Bioinformatics, 2007.

[76] C. J. Langmead and S. K. Jha, "Symbolic approaches to finding control strategies in boolean networks," in *Proceedings of The Sixth Asia-Pacific Bioinformatics Conference (APBC)*, pp. 307–319, 2008.

[77] C. Madsen, Z. Zhang, N. Roehner, C. Myers, Z. Zhang, C. Myers, and C. Winstead, "Stochastic Model Checking of Genetic Circuits," *ACM Journal on Emerging Technologies in Computing Systems ACM J. Emerg. Technol. Comput. Syst*, vol. 11, no. 21, 2014.

[78] M. Hucka, M. Hucka, F. Bergmann, S. Hoops, S. Keating, S. Sahle, J. Schaff, L. Smith, and D. Wilkinson, "The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core," *Nature Precedings*, 2010.

[79] N. Roehner, *TECHNOLOGY MAPPING OF GENETIC CIRCUIT DESIGNS*. PhD thesis, University of Utah, 2014.

[80] A. V. Hill, "The possible effects of the aggregation of the molecules of hæmoglobin on its dissociation curves," *The Journal of Physiology*, vol. 40, no. Supplement - Proceedings of the physiological society, pp. iv–vii, 1910.

[81] D. Waltemath, R. Adams, F. T. Bergmann, M. Hucka, F. Kolpakov, A. K. Miller, I. I. Moraru, D. Nickerson, S. Sahle, J. L. Snoep, and N. Le Novère, "Reproducible computational biology experiments with SED-ML–the Simulation Experiment Description Markup Language.," *BMC systems biology*, vol. 5, no. 1, p. 198, 2011.

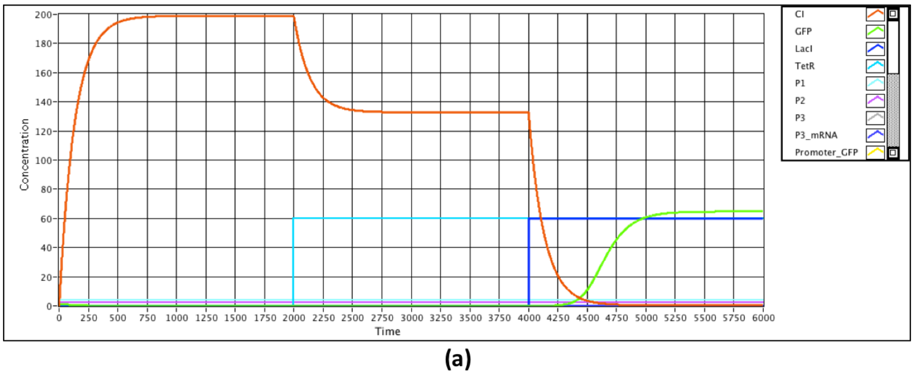
Appendices

Appendix A

D-VASim – *Supplementary Data*

A.1 ODE and Stochastic Simulation Results

The screen shots of ODE and stochastic simulations of genetic AND, NAND, NOR, NOT, and OR circuits [21] are given below. These screen-shots were captured automatically by D-VASim when the simulation was stopped by the user.



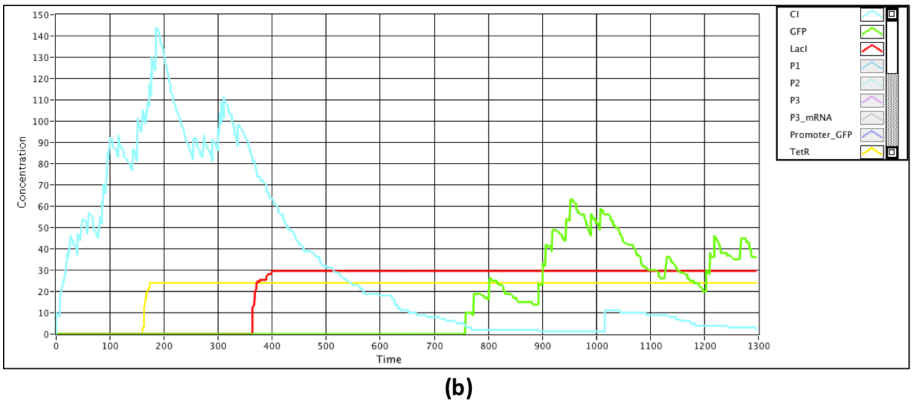
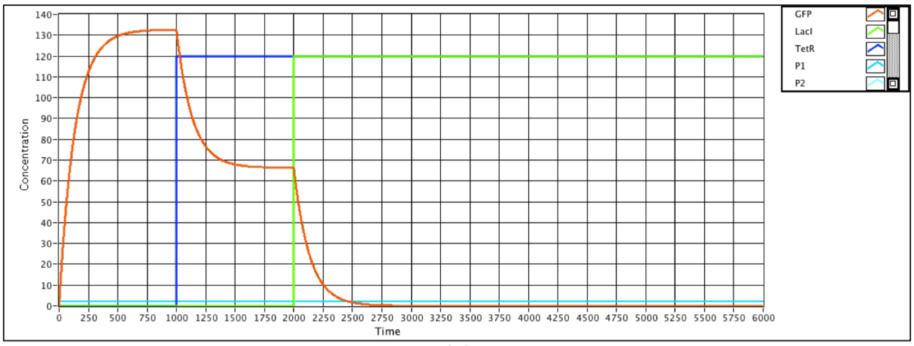
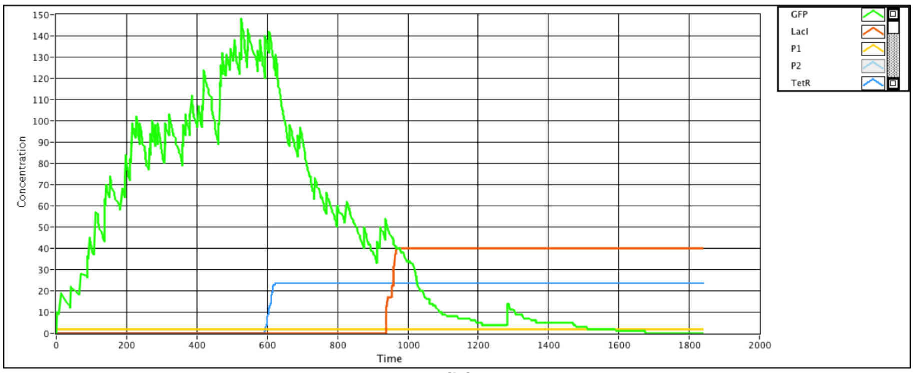


Figure A.1: Simulation of genetic AND gate circuit [21]. (a) ODE simulation. (b) Stochastic simulation.

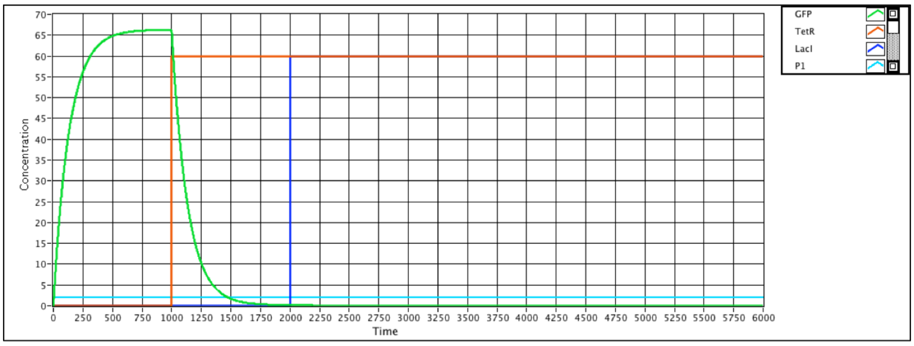


(a)

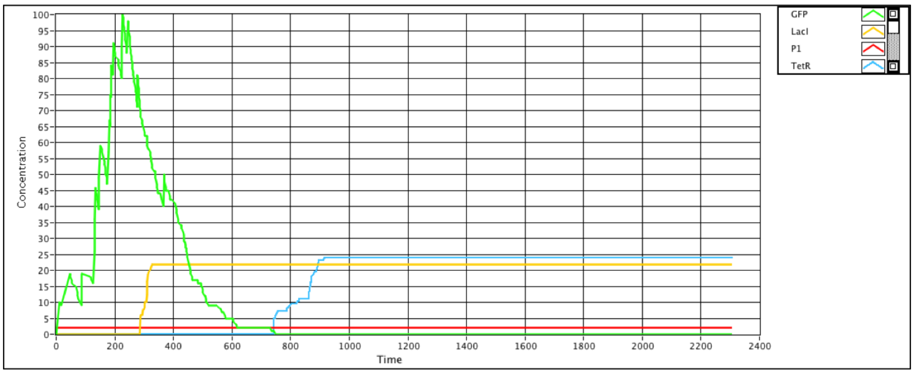


(b)

Figure A.2: Simulation of genetic NAND gate circuit [21]. (a) ODE simulation. (b) Stochastic simulation.

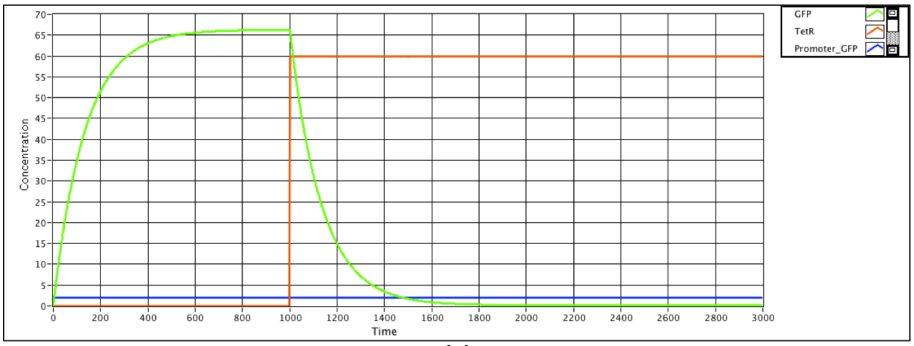


(a)

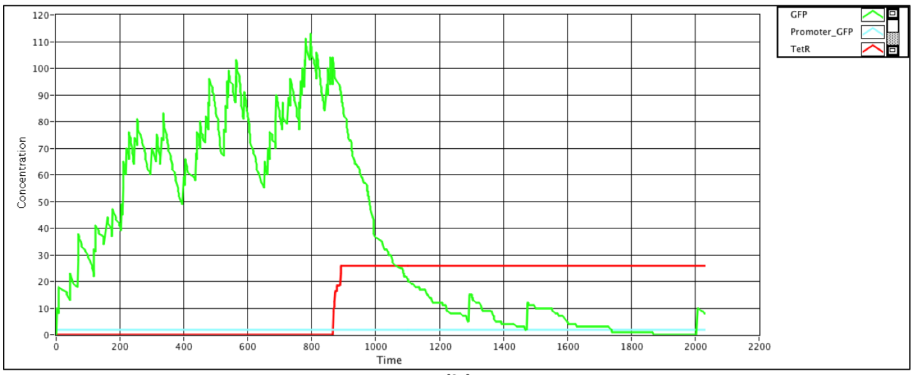


(b)

Figure A.3: Simulation of genetic NOR gate circuit [21]. (a) ODE simulation. (b) Stochastic simulation.



(a)



(b)

Figure A.4: Simulation of genetic NOT gate circuit [21]. (a) ODE simulation. (b) Stochastic simulation.

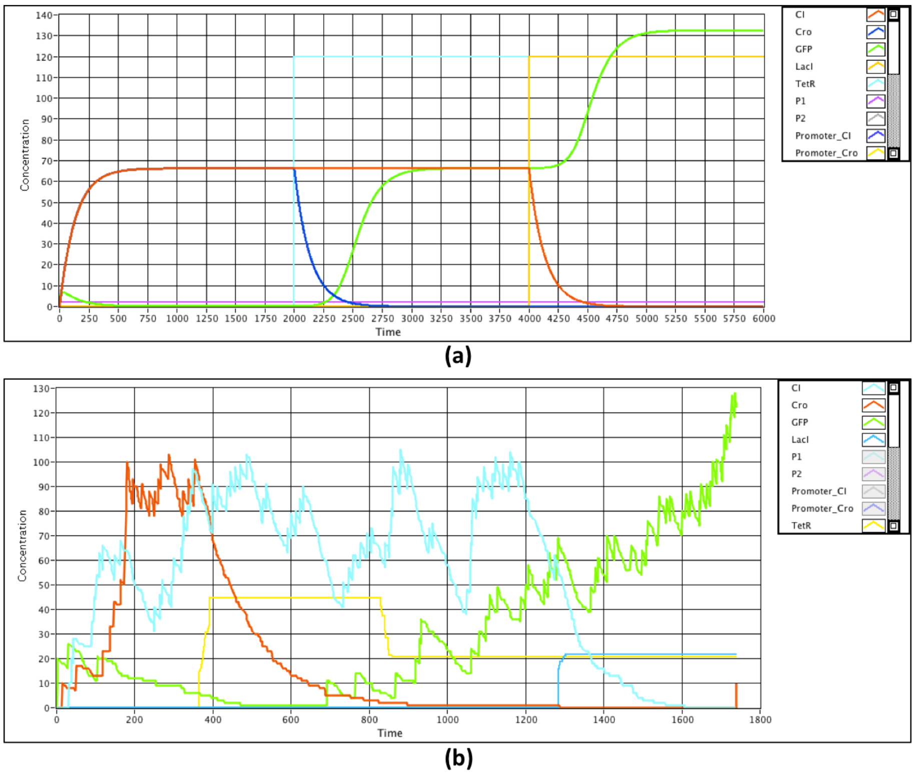


Figure A.5: Simulation of genetic OR gate circuit [21]. (a) ODE simulation. (b) Stochastic simulation.

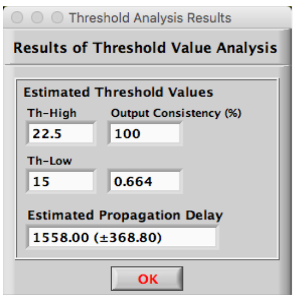
Appendix B

Timing Analysis – *Supplementary Data*

The simulation traces and the timing analysis results of all the nine circuits from [21] and an AND gate from [14] are given below.

B.1 Ckt 1 – NOT gate

The simulation results of the genetic *NOT gate* circuit for all five values of kd are shown below:


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

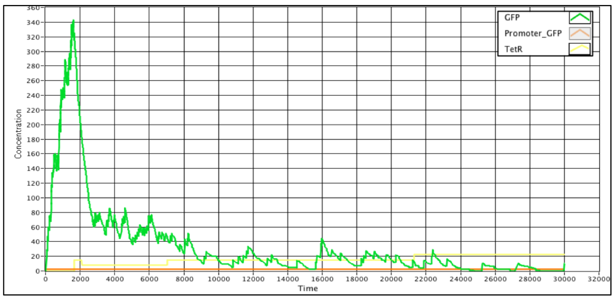
Th-High	Output Consistency (%)
22.5	100

Th-Low

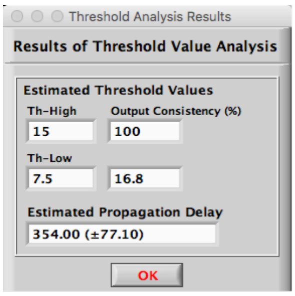
15	0.664

Estimated Propagation Delay

1558.00 (±368.80)



(a) At $kd = 0.0015$.



Threshold Analysis Results

Results of Threshold Value Analysis

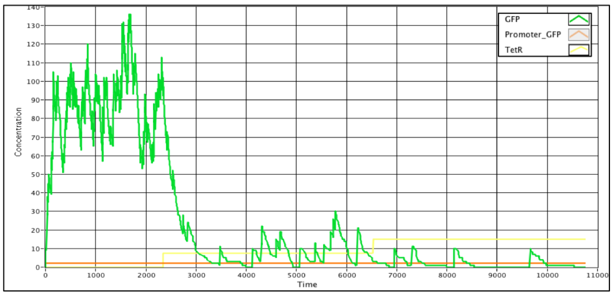
Estimated Threshold Values

Th-High	Output Consistency (%)
15	100
Th-Low	
7.5	16.8

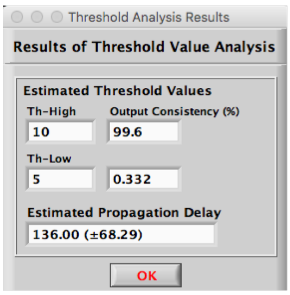
Estimated Propagation Delay

354.00 (±77.10)

OK



(b) At $kd = 0.0055$.



Threshold Analysis Results

Results of Threshold Value Analysis

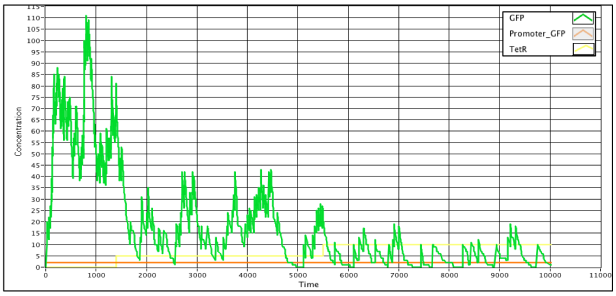
Estimated Threshold Values

Th-High	Output Consistency (%)
10	99.6
Th-Low	
5	0.332

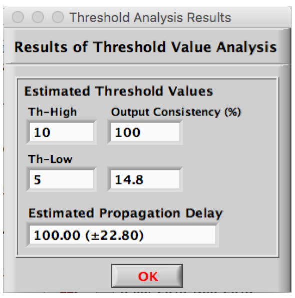
Estimated Propagation Delay

136.00 (±68.29)

OK



(c) At $kd = 0.0095$.



Threshold Analysis Results

Results of Threshold Value Analysis

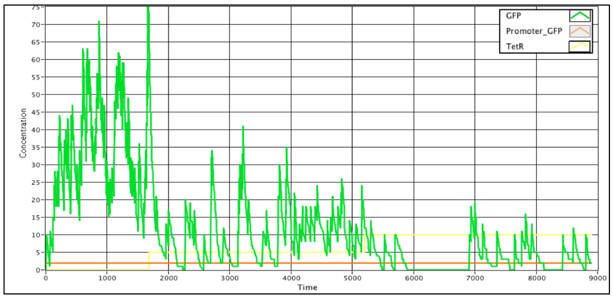
Estimated Threshold Values

Th-High	Output Consistency (%)
10	100
Th-Low	
5	14.8

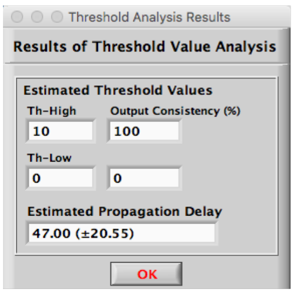
Estimated Propagation Delay

100.00 (±22.80)

OK



(d) At $kd = 0.0135$.


Threshold Analysis Results

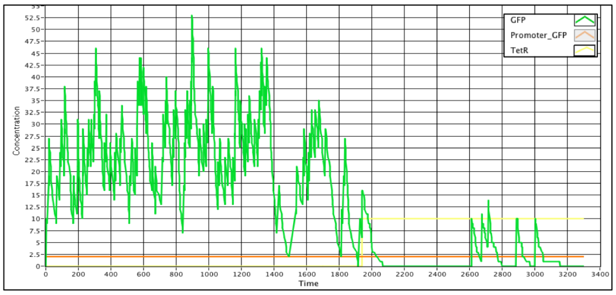
Results of Threshold Value Analysis

Estimated Threshold Values

Th-High	Output Consistency (%)
10	100
Th-Low	
0	0

Estimated Propagation Delay

47.00 (±20.55)

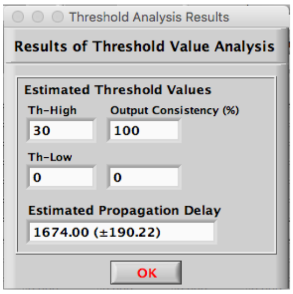


(e) At kd = 0.0215.

Figure B.1: Timing analysis and simulation traces of Ckt 1 - *NOT gate* at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.2 Ckt 2 – NAND gate

The timing simulation results of the genetic *NAND gate* circuit for all five values of *kd* are shown below:



Threshold Analysis Results

Results of Threshold Value Analysis

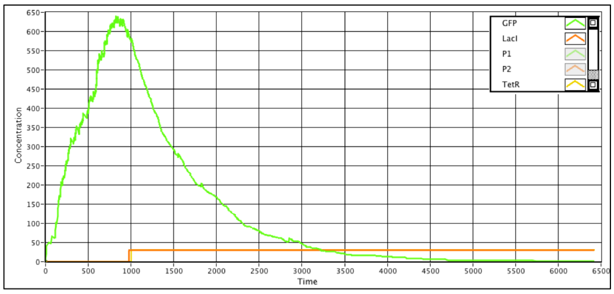
Estimated Threshold Values

Th-High	Output Consistency (%)
30	100
Th-Low	
0	0

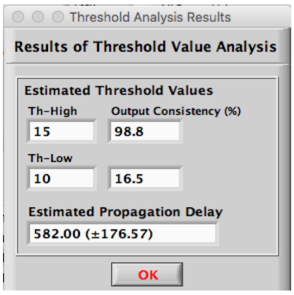
Estimated Propagation Delay

1674.00 (±190.22)

OK



(a) At $kd = 0.0015$.



Threshold Analysis Results

Results of Threshold Value Analysis

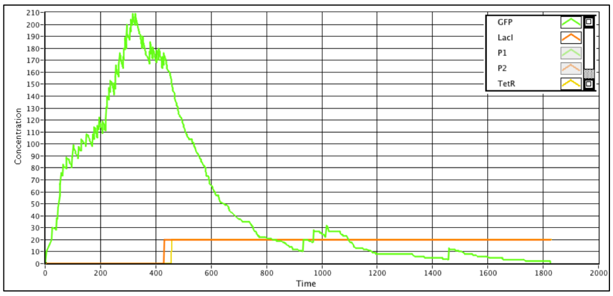
Estimated Threshold Values

Th-High	Output Consistency (%)
15	98.8
Th-Low	
10	16.5

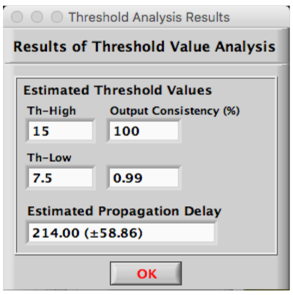
Estimated Propagation Delay

582.00 (±176.57)

OK



(b) At $kd = 0.0055$.



Threshold Analysis Results

Results of Threshold Value Analysis

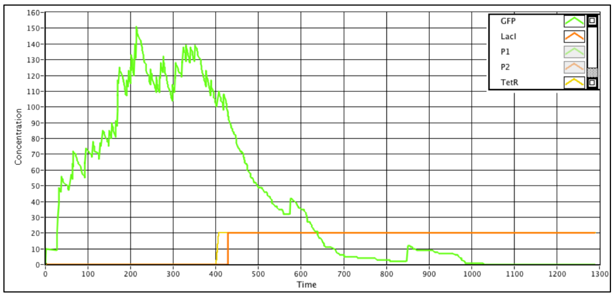
Estimated Threshold Values

Th-High	Output Consistency (%)
15	100
Th-Low	
7.5	0.99

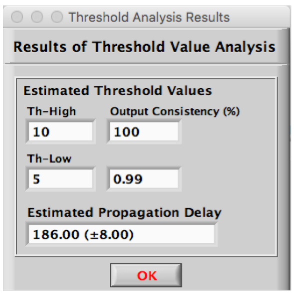
Estimated Propagation Delay

214.00 (±58.86)

OK



(c) At $kd = 0.0095$.


Threshold Analysis Results

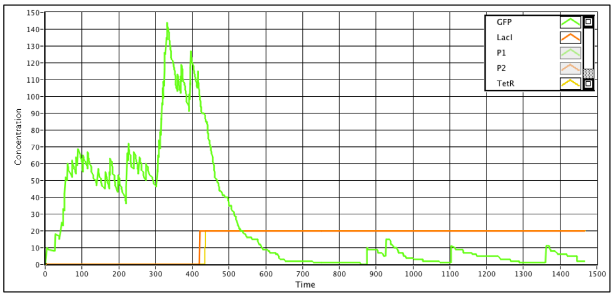
Results of Threshold Value Analysis

Estimated Threshold Values

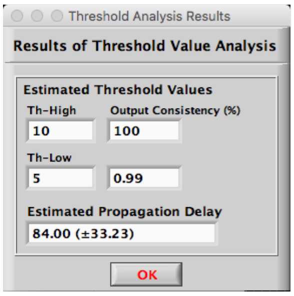
Th-High	Output Consistency (%)
10	100
Th-Low	
5	0.99

Estimated Propagation Delay

186.00 (±8.00)



(d) At $kd = 0.0135$.


Threshold Analysis Results

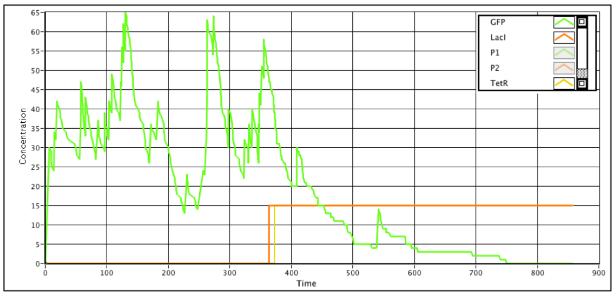
Results of Threshold Value Analysis

Estimated Threshold Values

Th-High	Output Consistency (%)
10	100
Th-Low	
5	0.99

Estimated Propagation Delay

84.00 (±33.23)

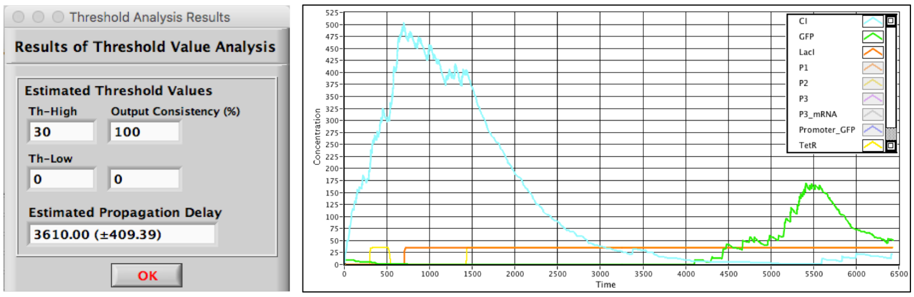


(e) At $kd = 0.0215$.

Figure B.2: Timing analysis and simulation traces of Ckt 2 - *NAND gate* at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.3 Ckt 3 – AND gate

The simulation results of the genetic *AND gate* circuit for all five values of kd are shown below:


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

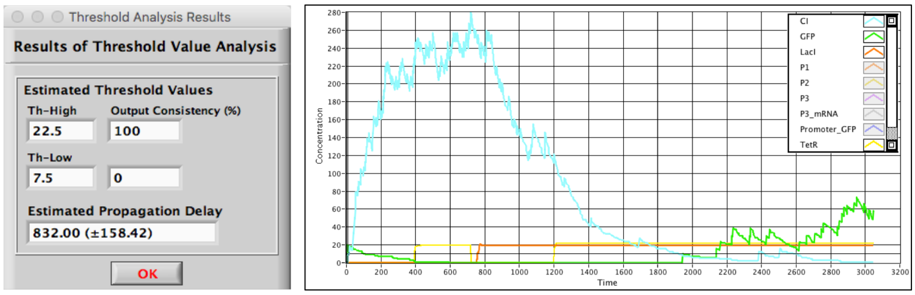
Th-High: 30 — Output Consistency (%): 100

Th-Low: 0 — 0

Estimated Propagation Delay: 3610.00 (±409.39)

OK

(a) At $kd = 0.0015$.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

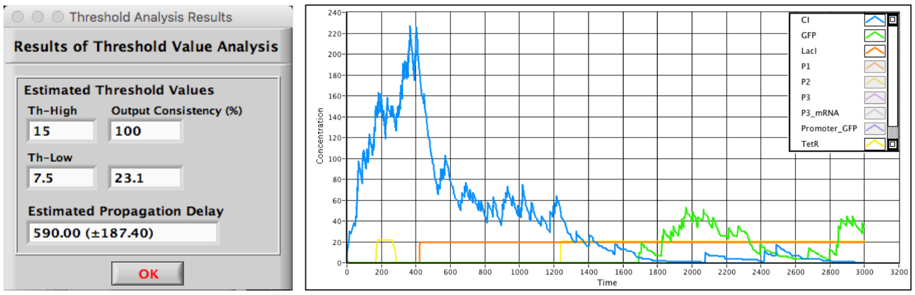
Th-High: 22.5 — Output Consistency (%): 100

Th-Low: 7.5 — 0

Estimated Propagation Delay: 832.00 (±158.42)

OK

(b) At $kd = 0.0055$.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

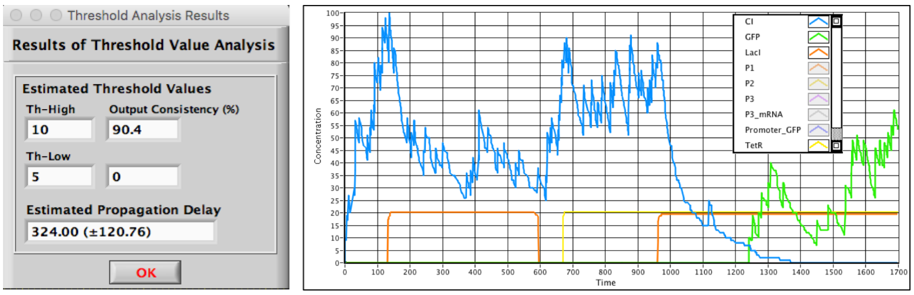
Th-High: 15 — Output Consistency (%): 100

Th-Low: 7.5 — 23.1

Estimated Propagation Delay: 590.00 (±187.40)

OK

(c) At $kd = 0.0095$.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

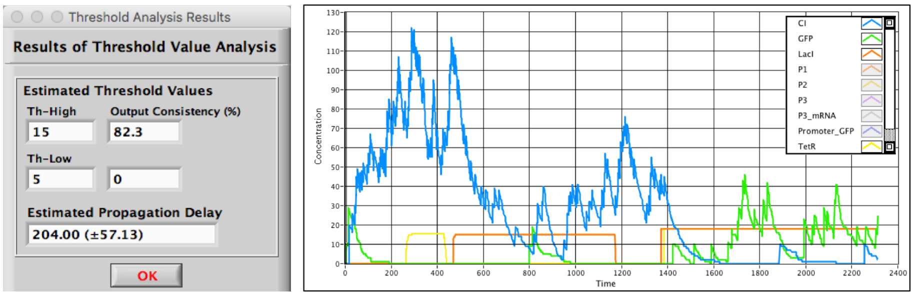
Th-High	Output Consistency (%)
10	90.4

Th-Low	
5	0

Estimated Propagation Delay

324.00 (±120.76)

(d) At kd = 0.0135.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

Th-High	Output Consistency (%)
15	82.3

Th-Low	
5	0

Estimated Propagation Delay

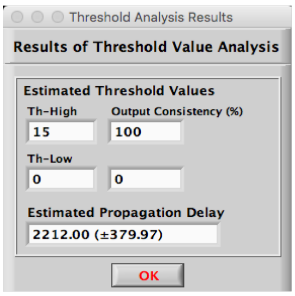
204.00 (±57.13)

(e) At kd = 0.0215.

Figure B.3: Timing analysis and simulation traces of Ckt 3 - *AND gate* at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.4 Ckt 4 – NOR gate

The simulation results of the genetic *NOR gate* circuit for all five values of kd are shown below:


Threshold Analysis Results

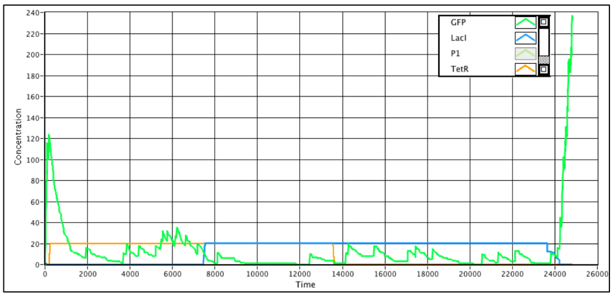
Results of Threshold Value Analysis

Estimated Threshold Values

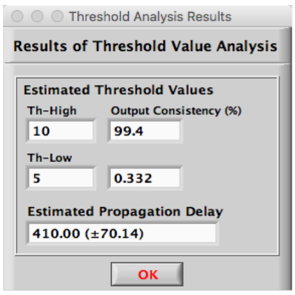
Th-High	Output Consistency (%)
15	100
Th-Low	
0	0

Estimated Propagation Delay

2212.00 (±379.97)



(a) At $kd = 0.0015$.


Threshold Analysis Results

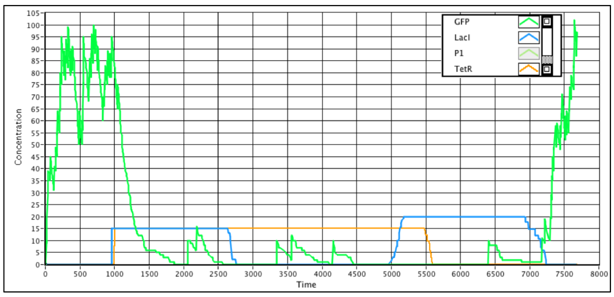
Results of Threshold Value Analysis

Estimated Threshold Values

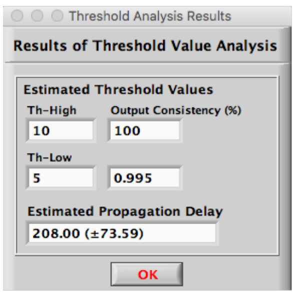
Th-High	Output Consistency (%)
10	99.4
Th-Low	
5	0.332

Estimated Propagation Delay

410.00 (±70.14)



(b) At $kd = 0.0055$.


Threshold Analysis Results

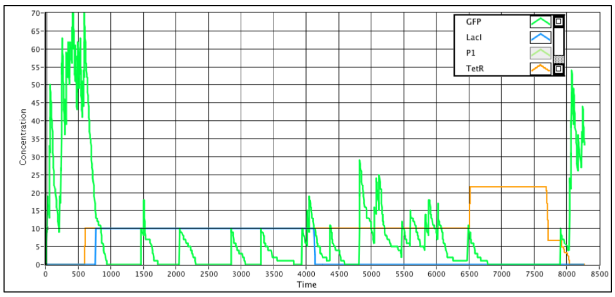
Results of Threshold Value Analysis

Estimated Threshold Values

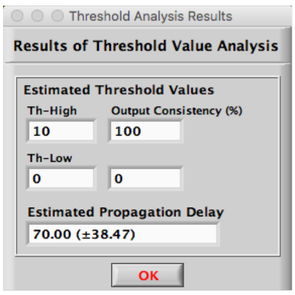
Th-High	Output Consistency (%)
10	100
Th-Low	
5	0.995

Estimated Propagation Delay

208.00 (±73.59)



(c) At $kd = 0.0095$.


Threshold Analysis Results

Results of Threshold Value Analysis

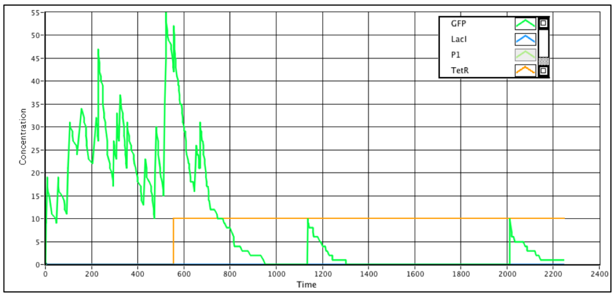
Estimated Threshold Values

Th-High	Output Consistency (%)
10	100
Th-Low	
0	0

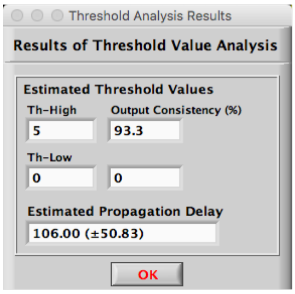
Estimated Propagation Delay

70.00 (±38.47)

OK



(d) At kd = 0.0135.


Threshold Analysis Results

Results of Threshold Value Analysis

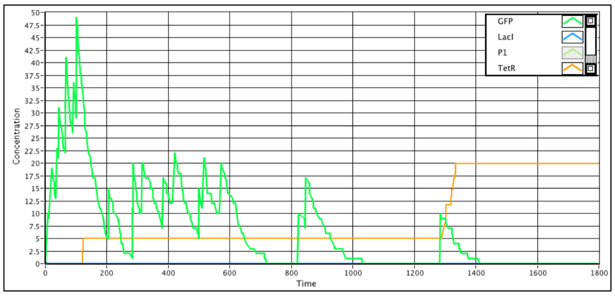
Estimated Threshold Values

Th-High	Output Consistency (%)
5	93.3
Th-Low	
0	0

Estimated Propagation Delay

106.00 (±50.83)

OK

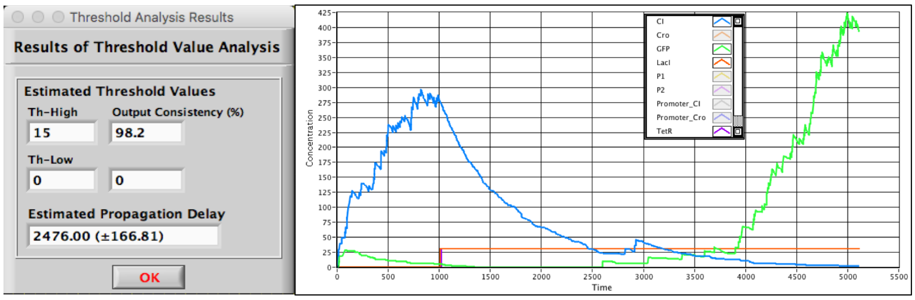


(e) At kd = 0.0215.

Figure B.4: Timing analysis and simulation traces of Ckt 4 - *NOR gate* at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.5 Ckt 5 – OR gate

The simulation results of the genetic *OR gate* circuit for all five values of *kd* are shown below:



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

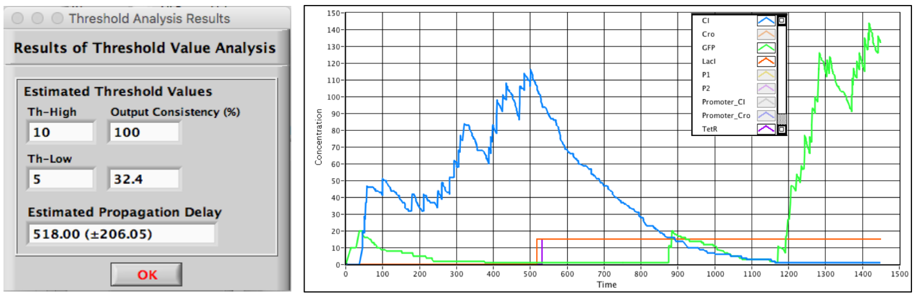
Th-High	Output Consistency (%)
15	98.2
Th-Low	
0	0

Estimated Propagation Delay

2476.00 (±166.81)

OK

(a) At $kd = 0.0015$.



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

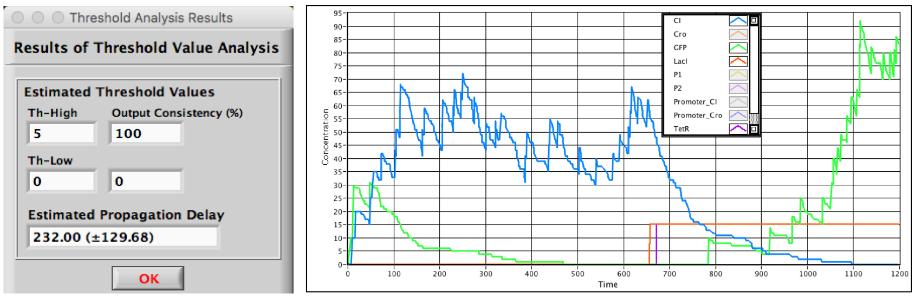
Th-High	Output Consistency (%)
10	100
Th-Low	
5	32.4

Estimated Propagation Delay

518.00 (±206.05)

OK

(b) At $kd = 0.0055$.



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

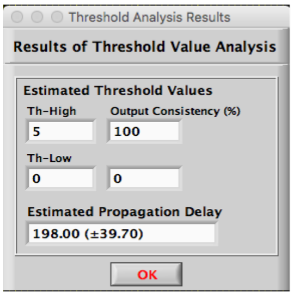
Th-High	Output Consistency (%)
5	100
Th-Low	
0	0

Estimated Propagation Delay

232.00 (±129.68)

OK

(c) At $kd = 0.0095$.


Threshold Analysis Results

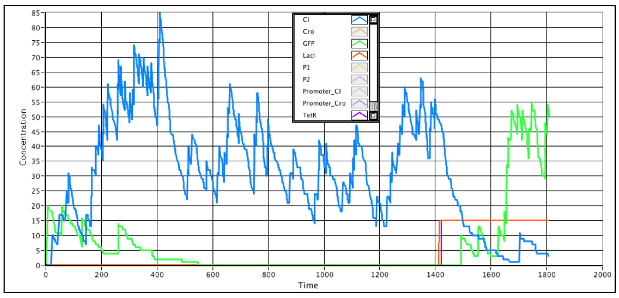
Results of Threshold Value Analysis

Estimated Threshold Values

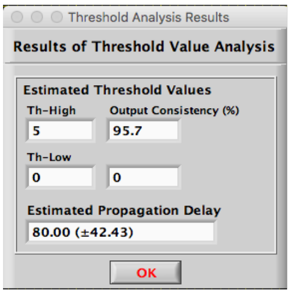
Th-High	Output Consistency (%)
5	100
Th-Low	
0	0

Estimated Propagation Delay

198.00 (±39.70)



(d) At kd = 0.0135.


Threshold Analysis Results

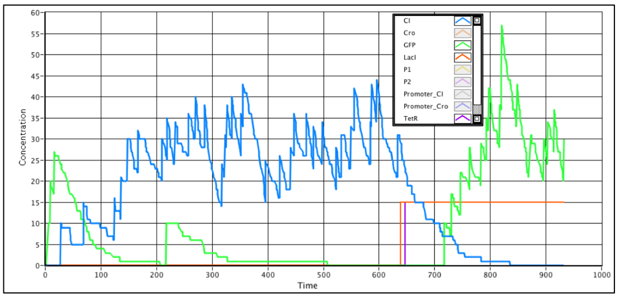
Results of Threshold Value Analysis

Estimated Threshold Values

Th-High	Output Consistency (%)
5	95.7
Th-Low	
0	0

Estimated Propagation Delay

80.00 (±42.43)

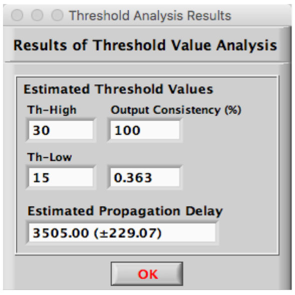


(e) At kd = 0.0215.

Figure B.5: Timing analysis and simulation traces of Ckt 5 - *OR gate* at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.6 Ckt 6 – Delay Circuit

The simulation results of the genetic *Delay Circuit* for all five values of *kd* are shown below:


Threshold Analysis Results

Results of Threshold Value Analysis

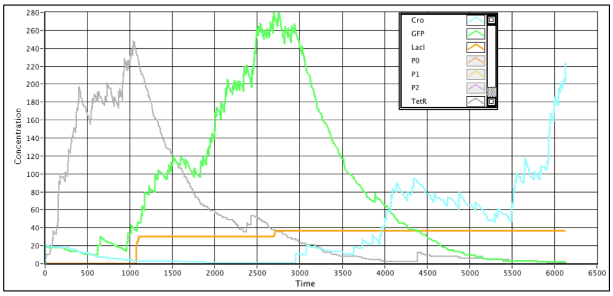
Estimated Threshold Values

Th-High	Output Consistency (%)
30	100
Th-Low	
15	0.363

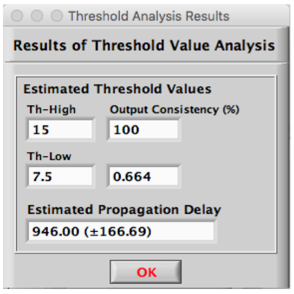
Estimated Propagation Delay

3505.00 (±229.07)

OK



(a) At $kd = 0.0015$.


Threshold Analysis Results

Results of Threshold Value Analysis

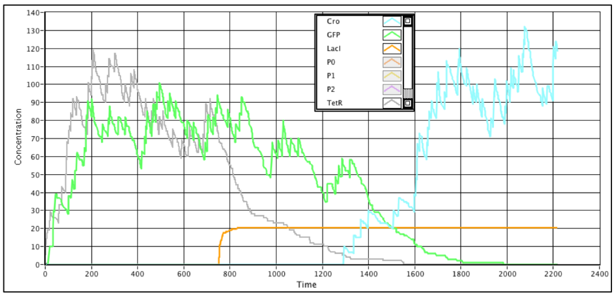
Estimated Threshold Values

Th-High	Output Consistency (%)
15	100
Th-Low	
7.5	0.664

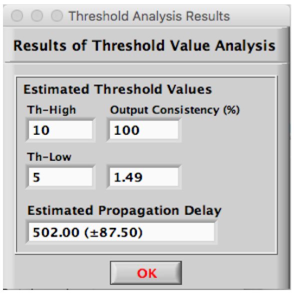
Estimated Propagation Delay

946.00 (±166.69)

OK



(b) At $kd = 0.0055$.


Threshold Analysis Results

Results of Threshold Value Analysis

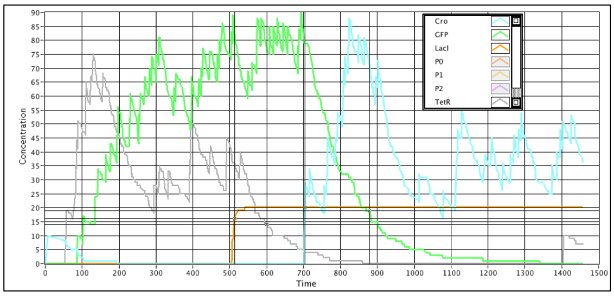
Estimated Threshold Values

Th-High	Output Consistency (%)
10	100
Th-Low	
5	1.49

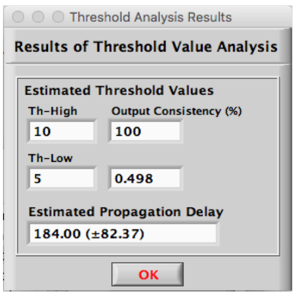
Estimated Propagation Delay

502.00 (±87.50)

OK



(c) At $kd = 0.0095$.



Results of Threshold Value Analysis

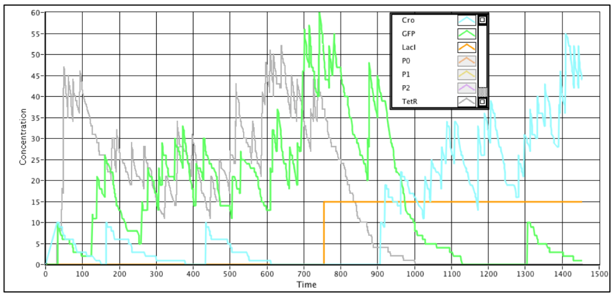
Estimated Threshold Values

Th-High	Output Consistency (%)
10	100

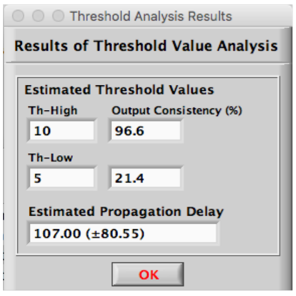
Th-Low	
5	0.498

Estimated Propagation Delay

184.00 (±82.37)



(d) At $kd = 0.0135$.



Results of Threshold Value Analysis

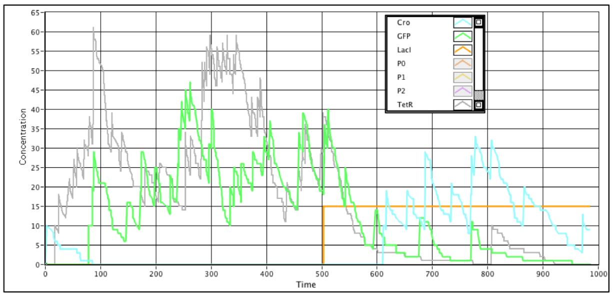
Estimated Threshold Values

Th-High	Output Consistency (%)
10	96.6

Th-Low	
5	21.4

Estimated Propagation Delay

107.00 (±80.55)

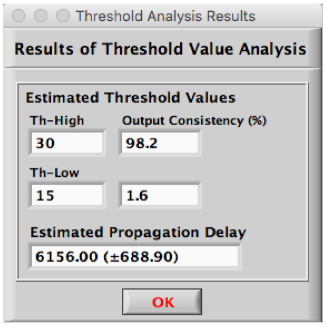


(e) At $kd = 0.0215$.

Figure B.6: Timing analysis and simulation traces of Ckt 6 - *Delay Circuit* at $kd =$ (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.7 Ckt 7 – Toggle Muller C-Element

The simulation results of the genetic *Toggle Muller C-Element* circuit for all five values of *kd* are shown below:


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

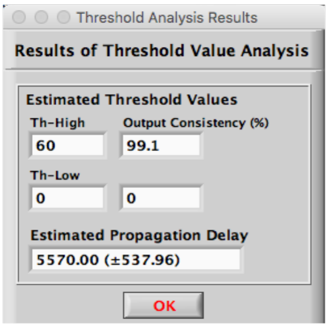
Th-High	Output Consistency (%)
30	98.2

Th-Low

15	1.6

Estimated Propagation Delay

6156.00 (±688.90)


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

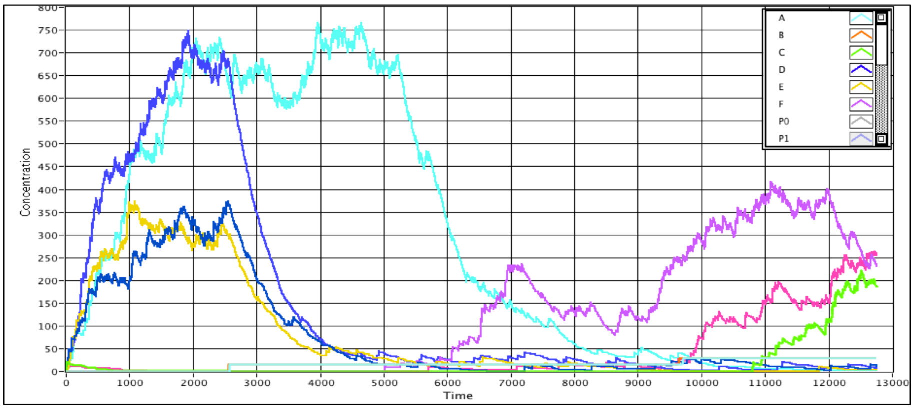
Th-High	Output Consistency (%)
60	99.1

Th-Low

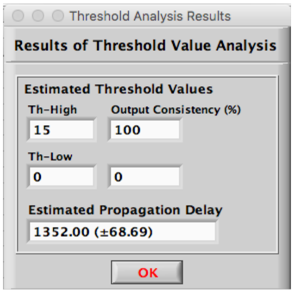
0	0

Estimated Propagation Delay

5570.00 (±537.96)



(a) At $kd = 0.0015$.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

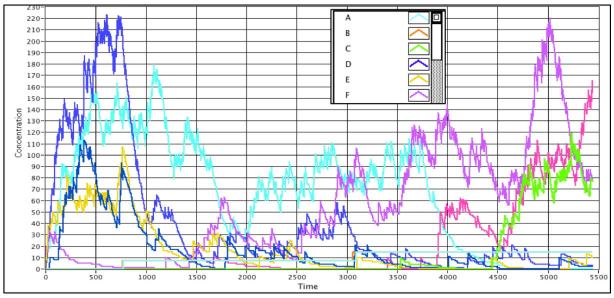
Th-High	Output Consistency (%)
15	100

Th-Low

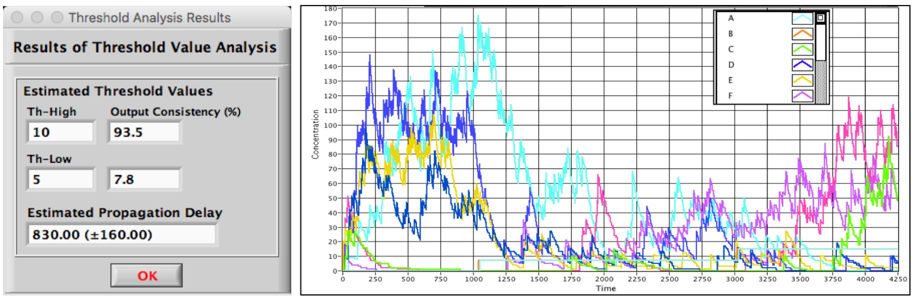
0	0

Estimated Propagation Delay

1352.00 (±68.69)



(b) At $kd = 0.0055$.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

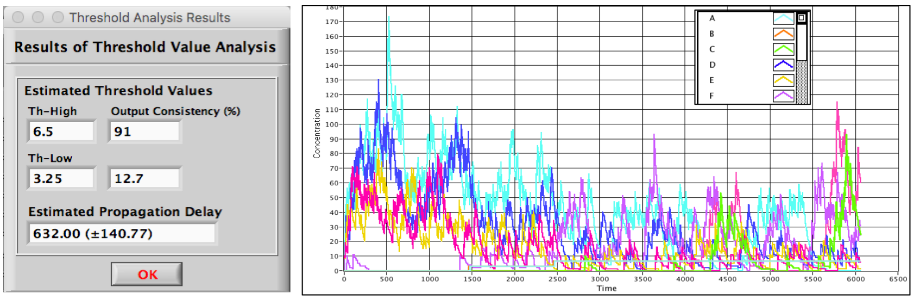
Th-High	Output Consistency (%)
10	93.5
Th-Low	
5	7.8

Estimated Propagation Delay

830.00 (±160.00)

OK

(c) At kd = 0.0095.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

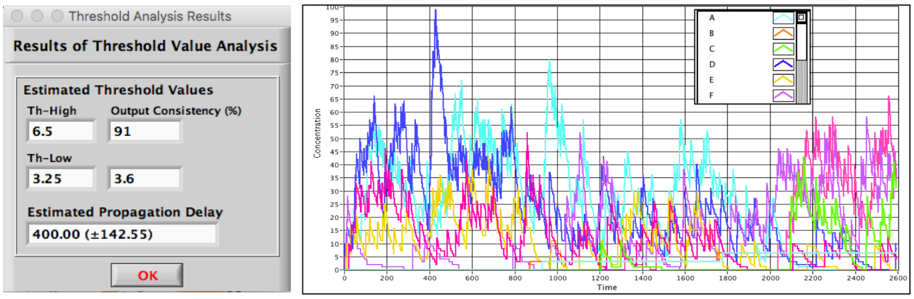
Th-High	Output Consistency (%)
6.5	91
Th-Low	
3.25	12.7

Estimated Propagation Delay

632.00 (±140.77)

OK

(d) At kd = 0.0135.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

Th-High	Output Consistency (%)
6.5	91
Th-Low	
3.25	3.6

Estimated Propagation Delay

400.00 (±142.55)

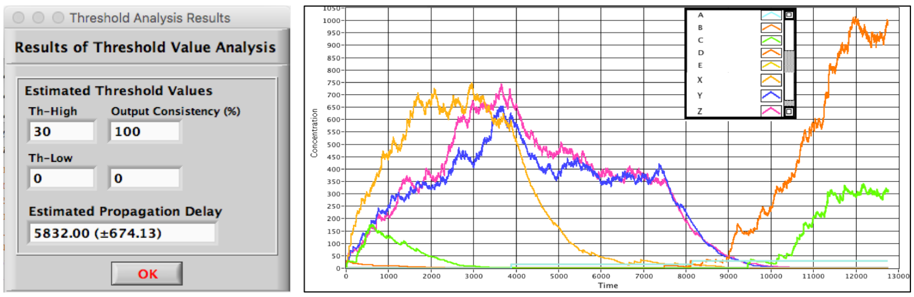
OK

(e) At kd = 0.0215.

Figure B.7: Timing analysis and simulation traces of Ckt 7 - *Toggle Muller C-Element* at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.8 Ckt 8 – Majority Muller C Element

The simulation results of the genetic *Majority Muller C Element* circuit for all five values of kd are shown below:


Threshold Analysis Results

Results of Threshold Value Analysis

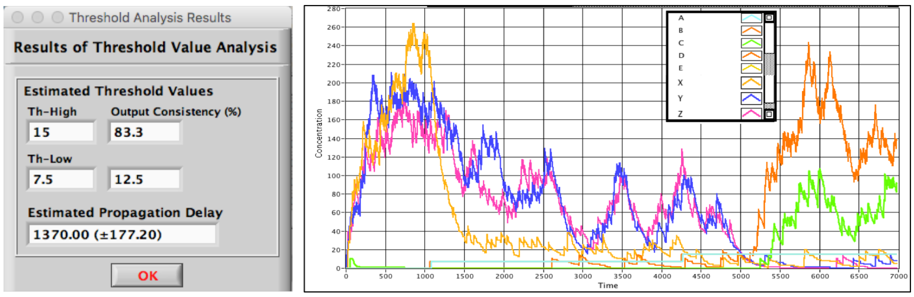
Estimated Threshold Values

Th-High	Output Consistency (%)
30	100
Th-Low	
0	0

Estimated Propagation Delay

5832.00 (±674.13)

(a) At $kd = 0.0015$.


Threshold Analysis Results

Results of Threshold Value Analysis

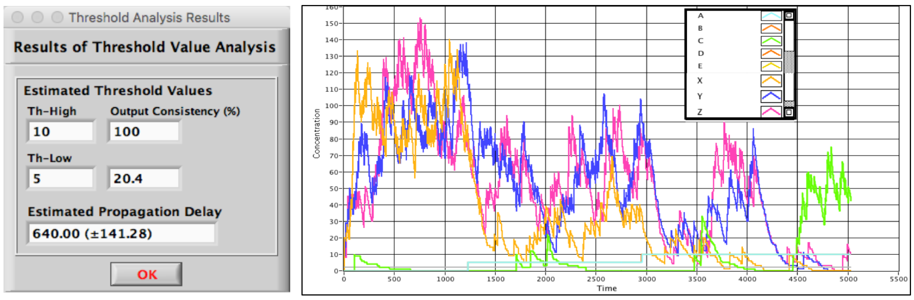
Estimated Threshold Values

Th-High	Output Consistency (%)
15	83.3
Th-Low	
7.5	12.5

Estimated Propagation Delay

1370.00 (±177.20)

(b) At $kd = 0.0055$.


Threshold Analysis Results

Results of Threshold Value Analysis

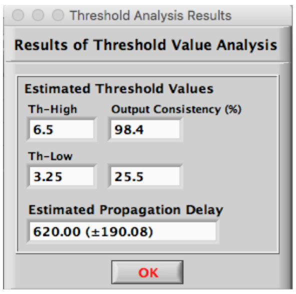
Estimated Threshold Values

Th-High	Output Consistency (%)
10	100
Th-Low	
5	20.4

Estimated Propagation Delay

640.00 (±141.28)

(c) At $kd = 0.0095$.


Threshold Analysis Results

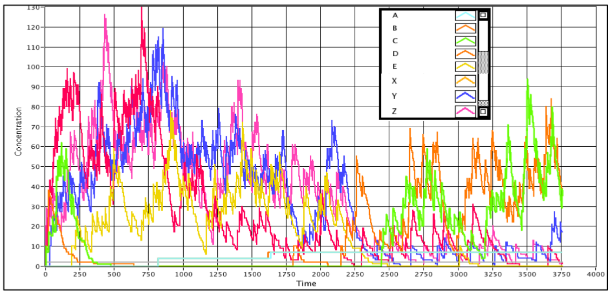
Results of Threshold Value Analysis

Estimated Threshold Values

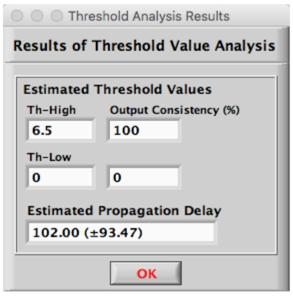
Th-High: 6.5 — Output Consistency (%): 98.4

Th-Low: 3.25 — 25.5

Estimated Propagation Delay: 620.00 (±190.08)



(d) At $kd = 0.0135$.


Threshold Analysis Results

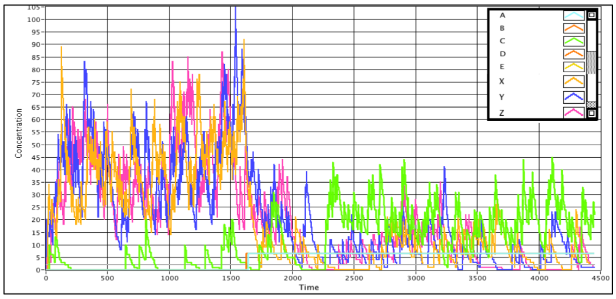
Results of Threshold Value Analysis

Estimated Threshold Values

Th-High: 6.5 — Output Consistency (%): 100

Th-Low: 0 — 0

Estimated Propagation Delay: 102.00 (±93.47)

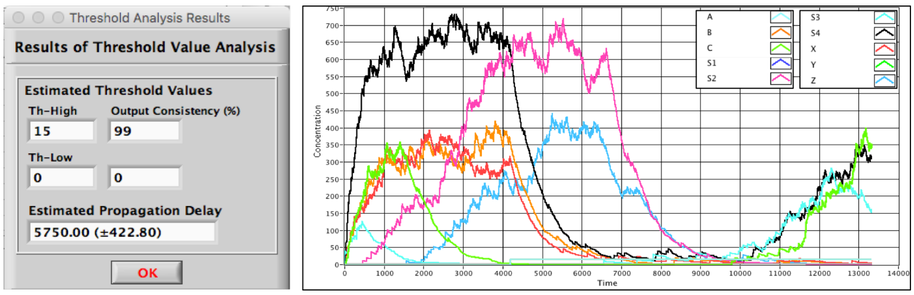


(e) At $kd = 0.0215$.

Figure B.8: Timing analysis and simulation traces of Ckt 8 - *Majority Muller C Element* at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.9 Ckt 9 – Speed Independent Muller C Element

The simulation results of the genetic *Speed Independent Muller C Element* circuit for all five values of kd are shown below:



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

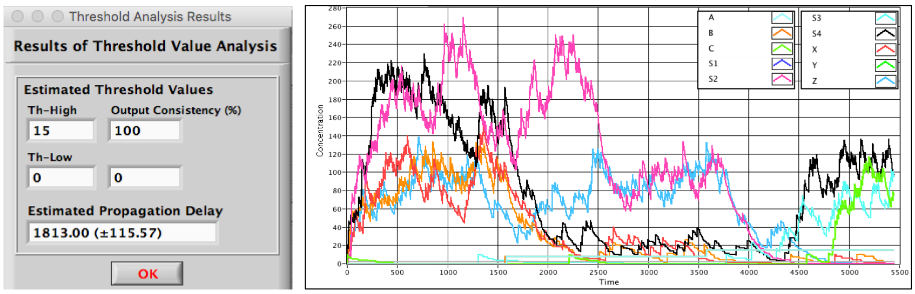
Th-High	Output Consistency (%)
15	99
Th-Low	
0	0

Estimated Propagation Delay

5750.00 (±422.80)

OK

(a) At $kd = 0.0015$.



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

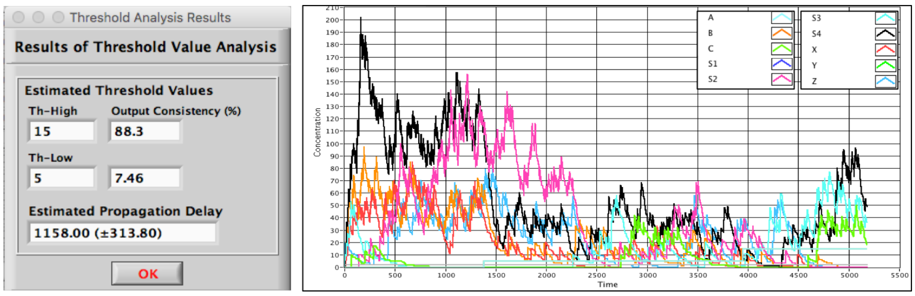
Th-High	Output Consistency (%)
15	100
Th-Low	
0	0

Estimated Propagation Delay

1813.00 (±115.57)

OK

(b) At $kd = 0.0055$.



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

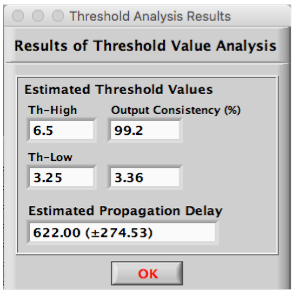
Th-High	Output Consistency (%)
15	88.3
Th-Low	
5	7.46

Estimated Propagation Delay

1158.00 (±313.80)

OK

(c) At $kd = 0.0095$.


Threshold Analysis Results

Results of Threshold Value Analysis

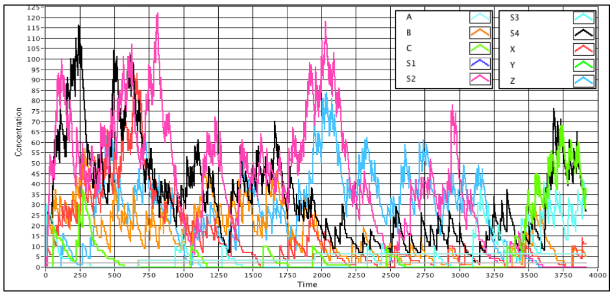
Estimated Threshold Values

Th-High: 6.5 — Output Consistency (%): 99.2

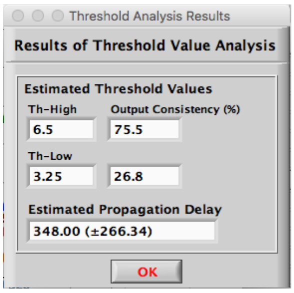
Th-Low: 3.25 — 3.36

Estimated Propagation Delay: 622.00 (±274.53)

OK



(d) At $kd = 0.0135$.


Threshold Analysis Results

Results of Threshold Value Analysis

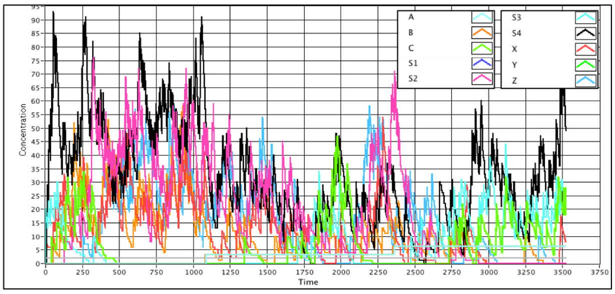
Estimated Threshold Values

Th-High: 6.5 — Output Consistency (%): 75.5

Th-Low: 3.25 — 26.8

Estimated Propagation Delay: 348.00 (±266.34)

OK

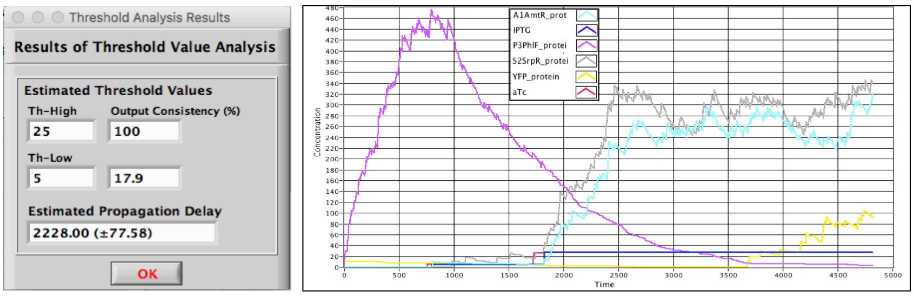


(e) At $kd = 0.0215$.

Figure B.9: Timing analysis and simulation traces of Ckt 8 - *Ckt 9 - Speed Independent Muller C Element* at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

B.10 Ckt 10 – Real Genetic AND Gate

The simulation results of a real genetic *AND gate* circuit (obtained from [14]) for all five values of *kd* are shown below:



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

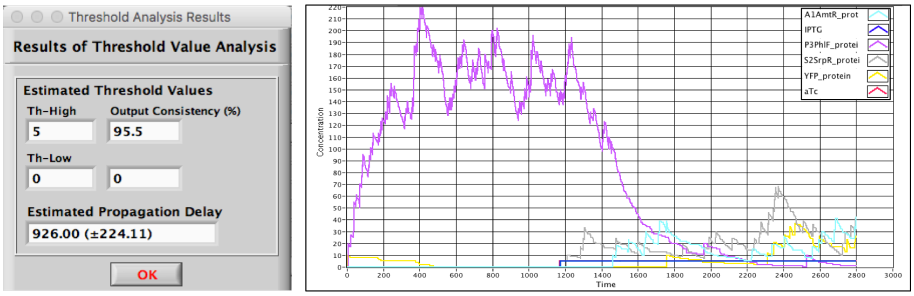
Th-High	Output Consistency (%)
25	100
Th-Low	
5	17.9

Estimated Propagation Delay

2228.00 (±77.58)

OK

(a) At $kd = 0.0015$.



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

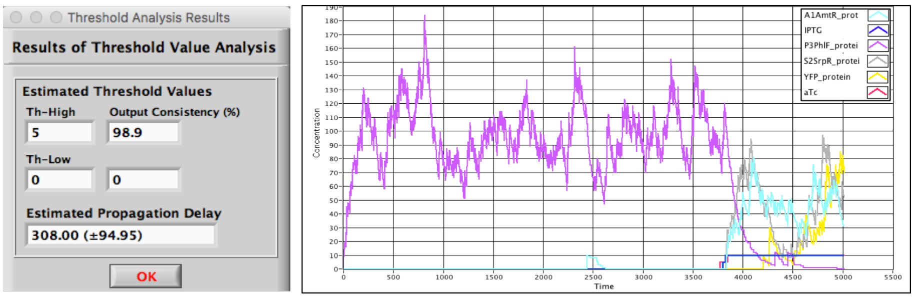
Th-High	Output Consistency (%)
5	95.5
Th-Low	
0	0

Estimated Propagation Delay

926.00 (±224.11)

OK

(b) At $kd = 0.0055$.



Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

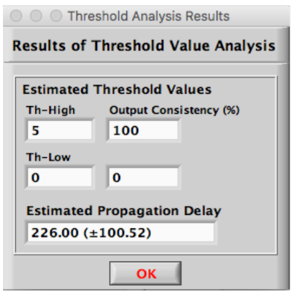
Th-High	Output Consistency (%)
5	98.9
Th-Low	
0	0

Estimated Propagation Delay

308.00 (±94.95)

OK

(c) At $kd = 0.0095$.


Threshold Analysis Results

Results of Threshold Value Analysis

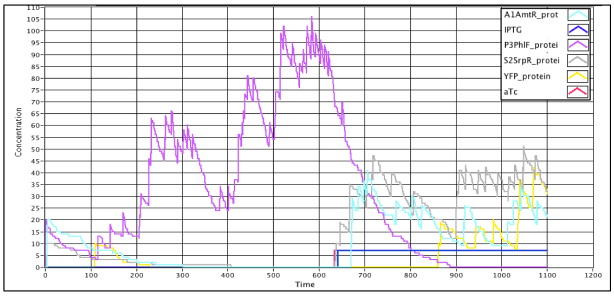
Estimated Threshold Values

Th-High: 5 Output Consistency (%): 100

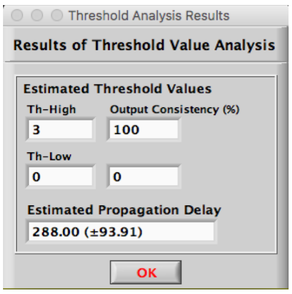
Th-Low: 0 Output Consistency (%): 0

Estimated Propagation Delay: 226.00 (±100.52)

OK



(d) At $kd = 0.0135$.


Threshold Analysis Results

Results of Threshold Value Analysis

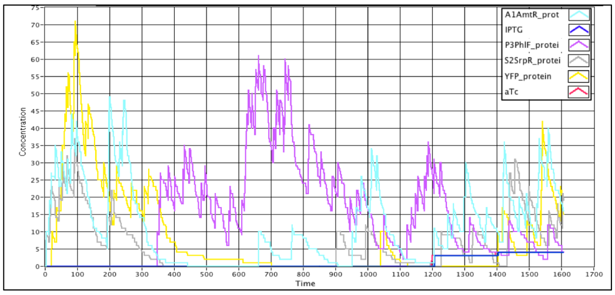
Estimated Threshold Values

Th-High: 3 Output Consistency (%): 100

Th-Low: 0 Output Consistency (%): 0

Estimated Propagation Delay: 288.00 (±93.91)

OK



(e) At $kd = 0.0215$.

Figure B.10: Timing analysis and simulation traces of Ckt 8 - *Ckt 9 - Real genetic AND gate* [14] at kd = (a) 0.0015. (b) 0.0055. (c) 0.0095. (d) 0.0135. (e) 0.0215.

APPENDIX C

Logic Analysis – *Supplementary Data*

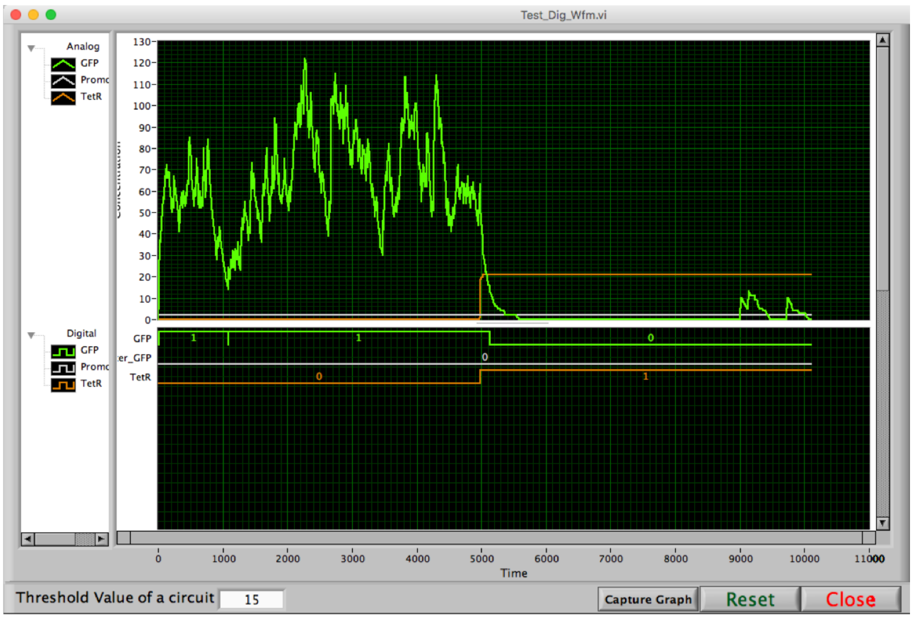
The experimental data for the logic analyses of remaining 14 circuits are given below in the following sections. As mentioned in Chapter 5, the upper threshold value = 15, and the propagation delay = 1000 time units are assumed in the experimentation for logic analyses of these circuits. Also, each circuit is experimented for about 10,000 simulation time units.

C.1 Experimentation on the genetic circuit models from University of Utah

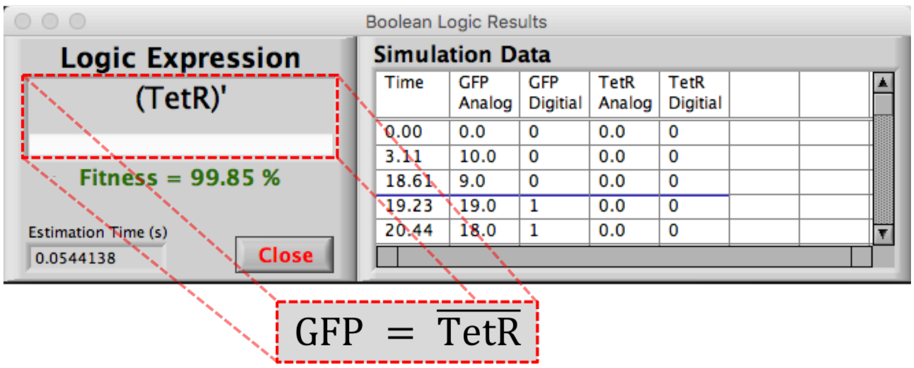
The experimental results of the genetic circuits obtained from [21] are given in Figures C1 - C5. Each of these figures contain two sub figures; (a) and (b). Sub figure (a) in all these figures is further divided into two images, which depicts (i) the simulation screen-shot with analog and digital waveforms; (ii) the estimated logic with percentage fitness and estimation time. Sub figure (b) shows the analytics of simulation data used by the algorithm to construct the Boolean expression in each case. In this figure, the input combinations, at which the circuit's output is expected to be high, are highlighted in green (on x-axis).

C.1.1 NOT gate circuit

The experimental results of the genetic **NOT** gate circuit are shown in the Figures C.1a and C.1b, respectively.



(i)



Boolean Logic Results

Logic Expression

(TetR)'

Fitness = 99.85 %

Estimation Time (s)

0.0544138

Close

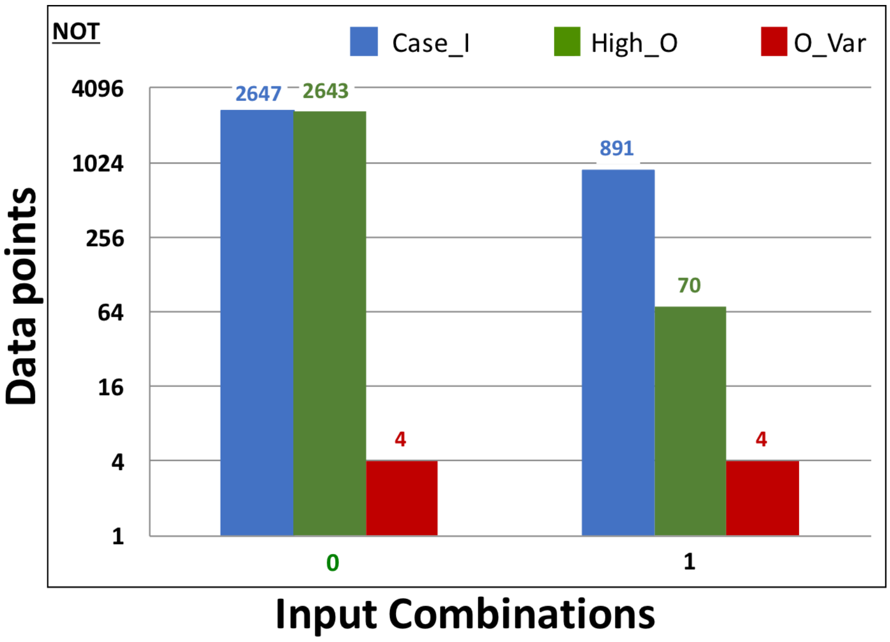
Simulation Data

Time	GFP Analog	GFP Digitial	TetR Analog	TetR Digitial
0.00	0.0	0	0.0	0
3.11	10.0	0	0.0	0
18.61	9.0	0	0.0	0
19.23	19.0	1	0.0	0
20.44	18.0	1	0.0	0

$GFP = \overline{TetR}$

(ii)

(a) (i) Circuit behavior when input is logic 0 and 1. (ii) Boolean expression estimated by the logic analysis algorithm.

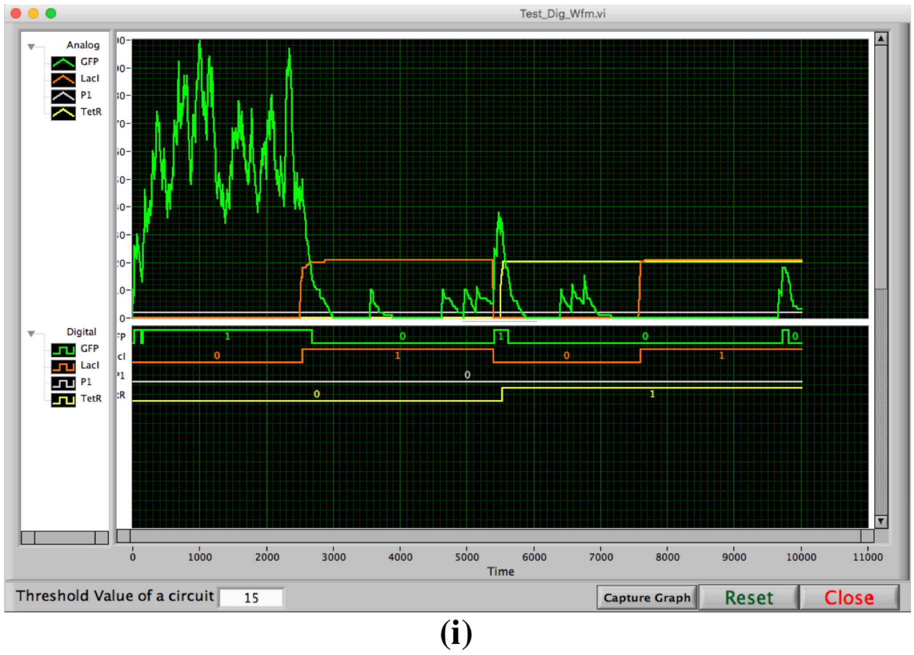


(b) Analytical simulation data for constructing the Boolean expression.

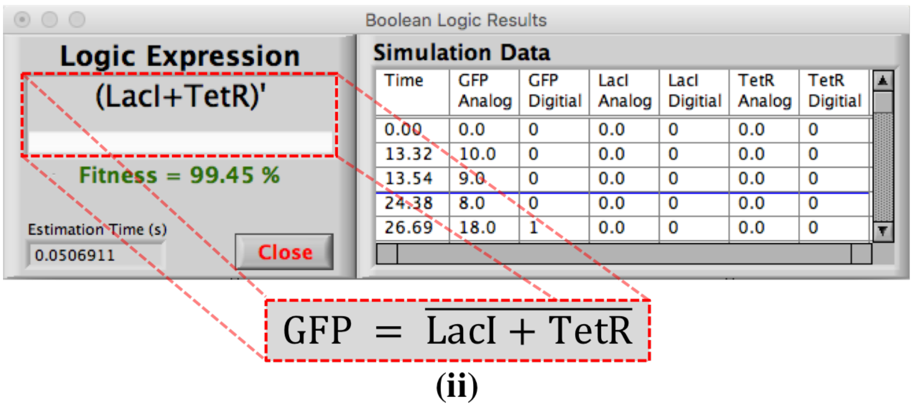
Figure C.1: Logic analysis and simulation results of the genetic NOT gate [21]. (a) Analog and digital simulation traces with Boolean logic estimation. (b) Analytical simulation data used for constructing the Boolean expression.

C.1.2 NOR gate circuit

The experimental results of the genetic **NOR** gate circuit are shown in the Figures C.2a and C.2b, respectively.



Threshold Value of a circuit 15



Logic Expression

(LacI+TetR)'

Fitness = 99.45 %

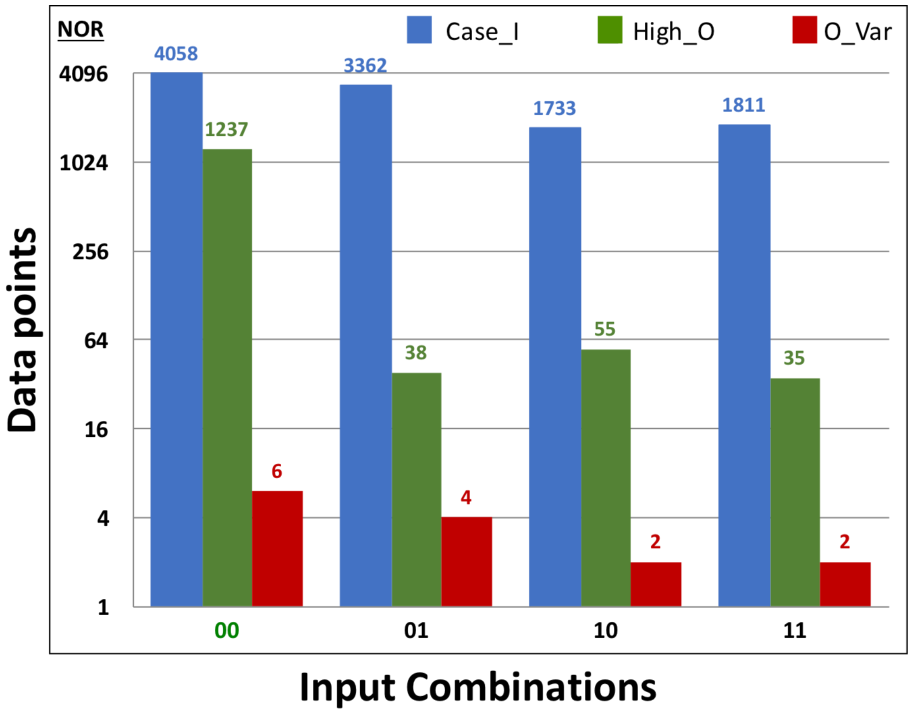
Estimation Time (s) 0.0506911

Simulation Data

Time	GFP Analog	GFP Digitial	LacI Analog	LacI Digitial	TetR Analog	TetR Digitial
0.00	0.0	0	0.0	0	0.0	0
13.32	10.0	0	0.0	0	0.0	0
13.54	9.0	0	0.0	0	0.0	0
24.38	8.0	0	0.0	0	0.0	0
26.69	18.0	1	0.0	0	0.0	0

$$\mathrm{GFP} = \overline{\mathrm{LacI} + \mathrm{TetR}}$$

(a) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

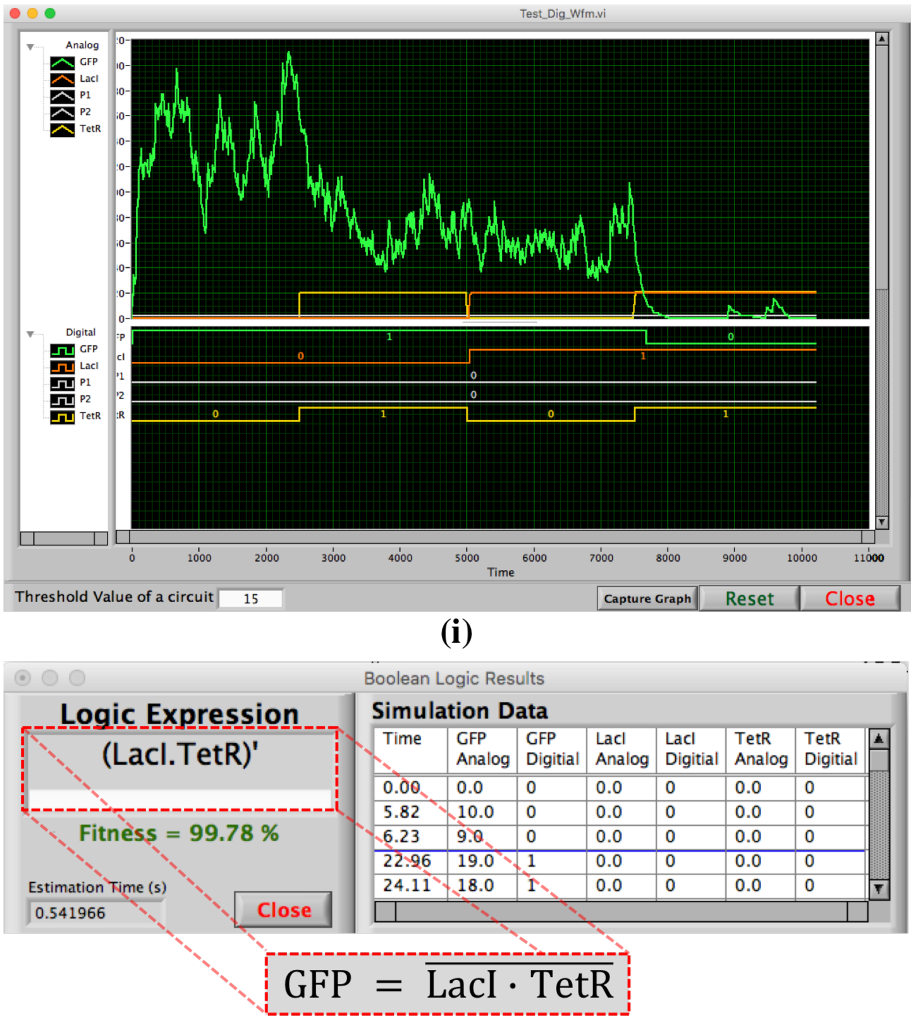


(b) Analytical simulation data for constructing the Boolean expression.

Figure C.2: Logic analysis and simulation results of the genetic NOR gate [21]. (a) Analog and digital simulation traces with Boolean logic estimation. (b) Analytical simulation data used for constructing the Boolean expression.

C.1.3 NAND gate circuit

The experimental results of the genetic **NAND** gate circuit are shown in the Figures C.3a and C.3b, respectively.



Threshold Value of a circuit 15

Logic Expression

(LacI.TetR)'

Fitness = 99.78 %

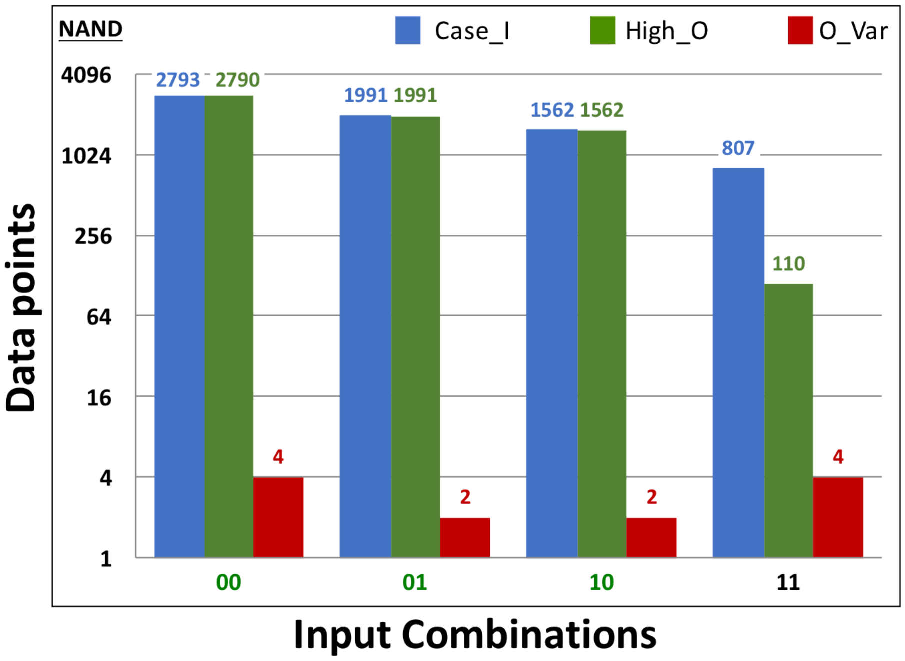
Estimation Time (s) 0.541966

Simulation Data

Time	GFP Analog	GFP Digitial	LacI Analog	LacI Digitial	TetR Analog	TetR Digitial
0.00	0.0	0	0.0	0	0.0	0
5.82	10.0	0	0.0	0	0.0	0
6.23	9.0	0	0.0	0	0.0	0
22.96	19.0	1	0.0	0	0.0	0
24.11	18.0	1	0.0	0	0.0	0

$$\mathrm{GFP} = \overline{\mathrm{LacI} \cdot \mathrm{TetR}}$$

(a) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

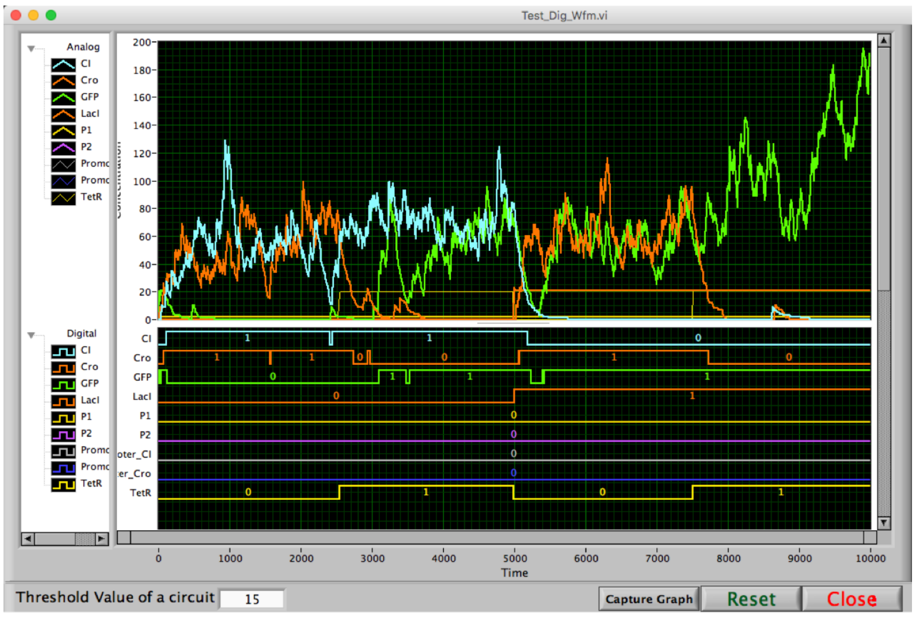


(b) Analytical simulation data for constructing the Boolean expression.

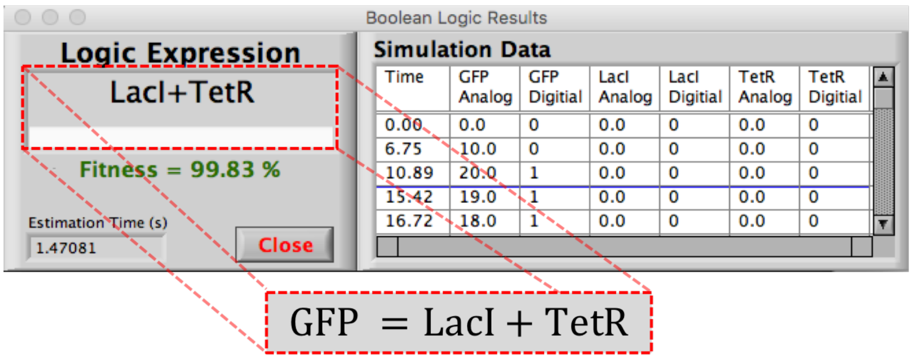
Figure C.3: Logic analysis and simulation results of the genetic NAND gate [21]. (a) Analog and digital simulation traces with Boolean logic estimation. (b) Analytical simulation data used for constructing the Boolean expression.

C.1.4 OR gate circuit

The experimental results of the genetic **OR** gate circuit are shown in the Figures C.4a and C.4b, respectively.



(i)



Logic Expression

LacI+TetR

Fitness = 99.83 %

Estimation Time (s) 1.47081

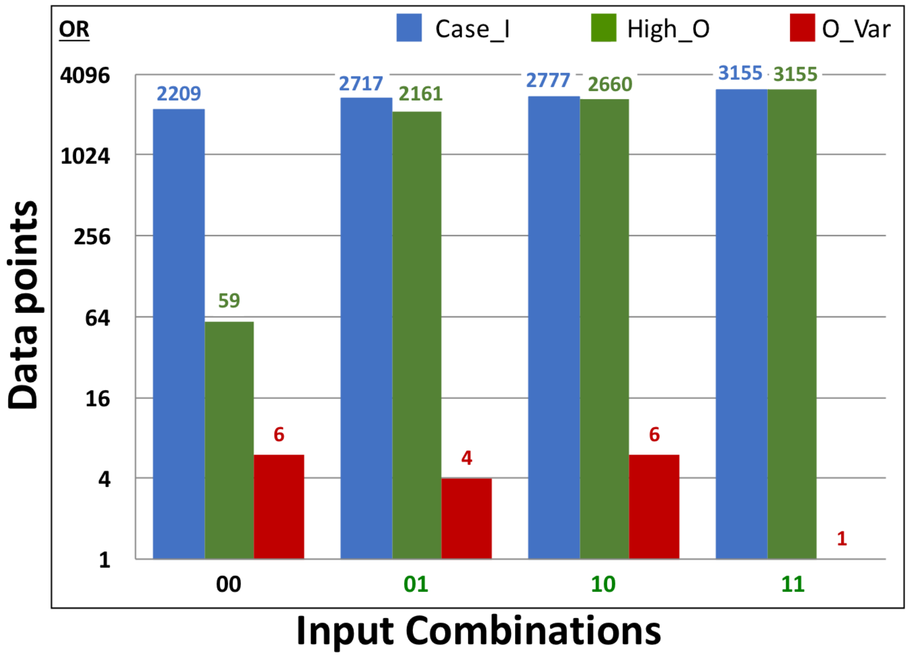
Simulation Data

Time	GFP Analog	GFP Digitial	LacI Analog	LacI Digitial	TetR Analog	TetR Digitial
0.00	0.0	0	0.0	0	0.0	0
6.75	10.0	0	0.0	0	0.0	0
10.89	20.0	1	0.0	0	0.0	0
15.42	19.0	1	0.0	0	0.0	0
16.72	18.0	1	0.0	0	0.0	0

$$\mathrm{GFP} = \mathrm{LacI} + \mathrm{TetR}$$

(ii)

(a) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

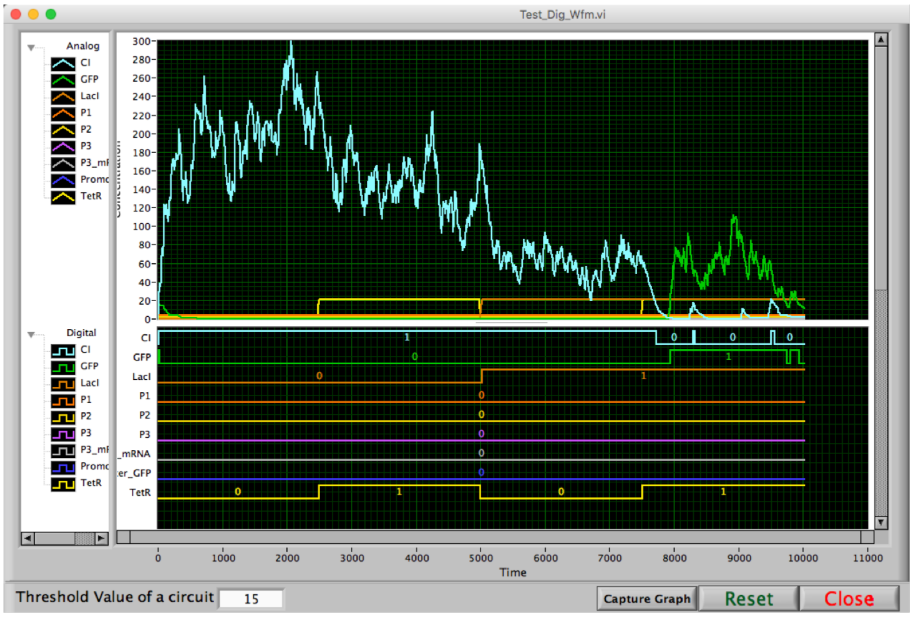


(b) Analytical simulation data for constructing the Boolean expression.

Figure C.4: Logic analysis and simulation results of the genetic OR gate [21]. (a) Analog and digital simulation traces with Boolean logic estimation. (b) Analytical simulation data used for constructing the Boolean expression.

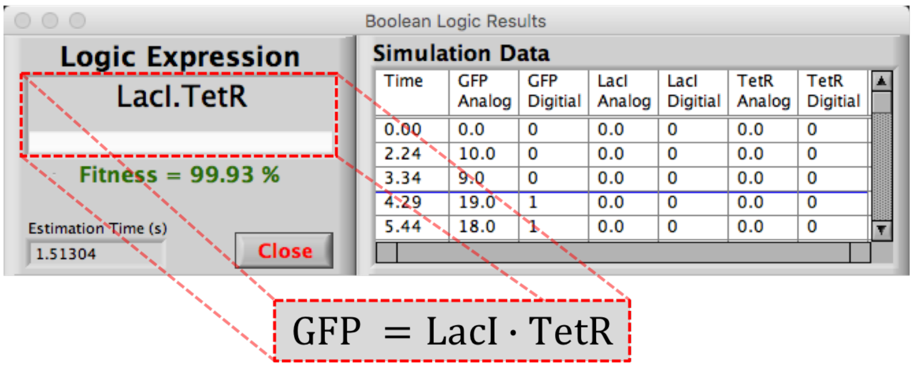
C.1.5 AND gate circuit

The experimental results of the genetic **AND** gate circuit are shown in the Figures C.5a and C.5b, respectively.



Threshold Value of a circuit 15

(i)



Logic Expression

LacI.TetR

Fitness = 99.93 %

Estimation Time (s) 1.51304

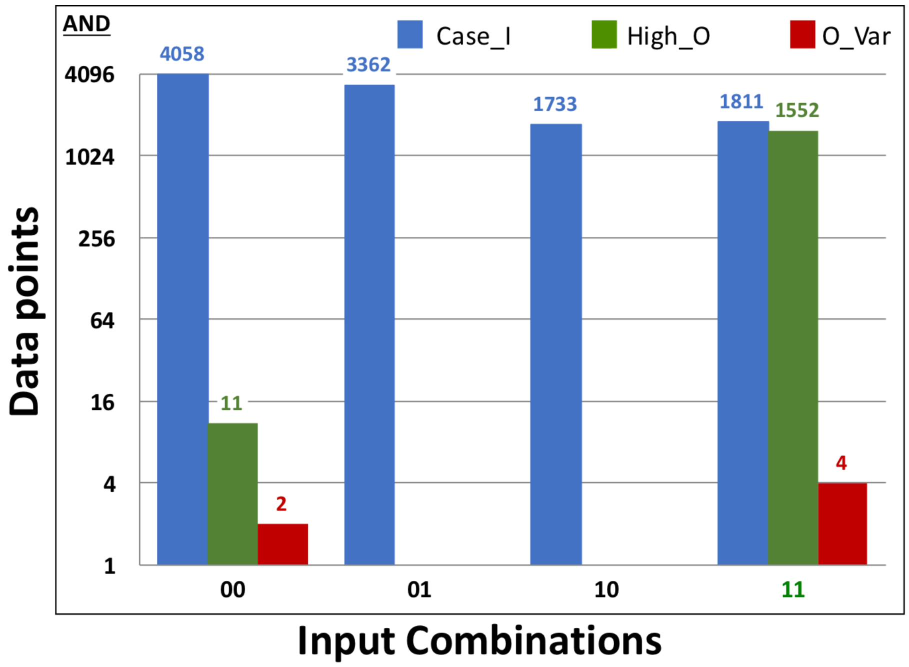
Simulation Data

Time	GFP Analog	GFP Digitial	LacI Analog	LacI Digitial	TetR Analog	TetR Digitial
0.00	0.0	0	0.0	0	0.0	0
2.24	10.0	0	0.0	0	0.0	0
3.34	9.0	0	0.0	0	0.0	0
4.29	19.0	1	0.0	0	0.0	0
5.44	18.0	1	0.0	0	0.0	0

$$\mathrm{GFP} = \mathrm{LacI} \cdot \mathrm{TetR}$$

(ii)

(a) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.



(b) Analytical simulation data for constructing the Boolean expression.

Figure C.5: Logic analysis and simulation results of the genetic AND gate [21]. (a) Analog and digital simulation traces with Boolean logic estimation. (b) Analytical simulation data used for constructing the Boolean expression.

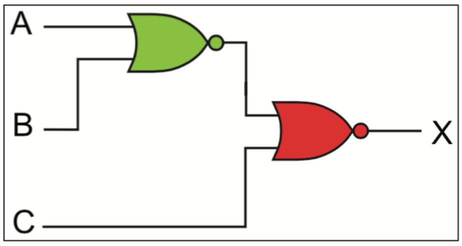
C.2 Experimentation on the genetic circuit models from MIT and BU

The experimental results of the genetic circuit models obtained from [14] are given in Figures C6 - C14. Each of these figures contain three sub figures; (a), (b) and (c). Sub figure (a) in all of these nine figures is further divided into four images, which depicts (i) the circuit schematic; (ii) the circuit's response (or truth table) and Boolean expression; (iii) the SBOL representation; and (iv) the SBML model converted from the circuit's SBOL file. The images shown as (i), (ii) and (iii) in sub figure (a) are the screen-captured images directly obtained from [14]. The image shown as (iv) in sub figure (a) is the screen-captured taken in iBioSim, where the sensor block having external input inducers (shown as red-dotted block) is added manually and rest of the model (shown as blue-dotted block) is the result of the SBOL-SBML conversion process [61].

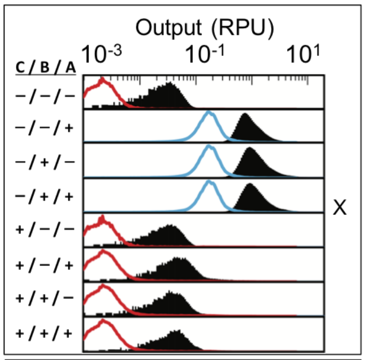
Sub figure (b) in all of these nine figures is also divided into two images, which depicts (i) the screen-shot of simulation with analog and digital waveforms; and (ii) the estimated logic with percentage fitness and estimation time. Sub figure (c) shows the analytics of simulation data used by the algorithm to construct the Boolean expression in each case. In sub figure (c), the input combinations, at which the circuit's output is expected to be high, are highlighted in green color along the x-axis.

C.2.1 Genetic circuit 0x70

The schematic circuit diagram of the genetic circuit, **0x70**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.6a(i-iv). The experimental results of this circuit are shown in the Figures C.6b and C.6c, respectively.

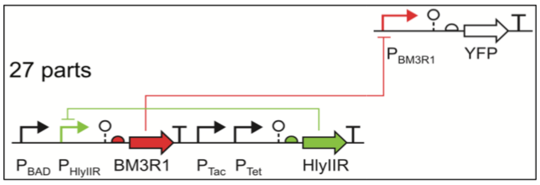


(i)

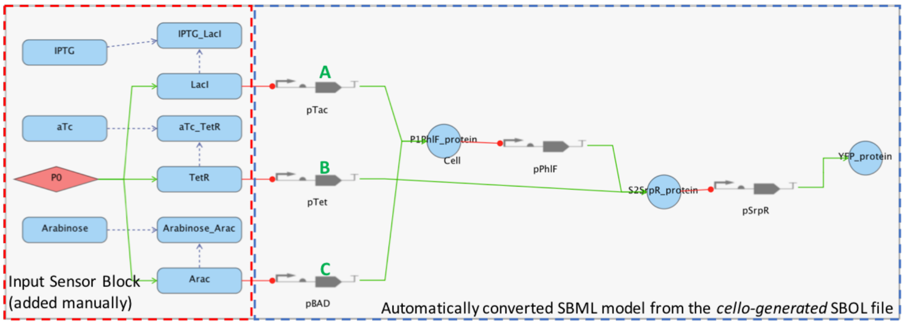


$$\mathrm{YFP} = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot \overline{C}$$

(ii)

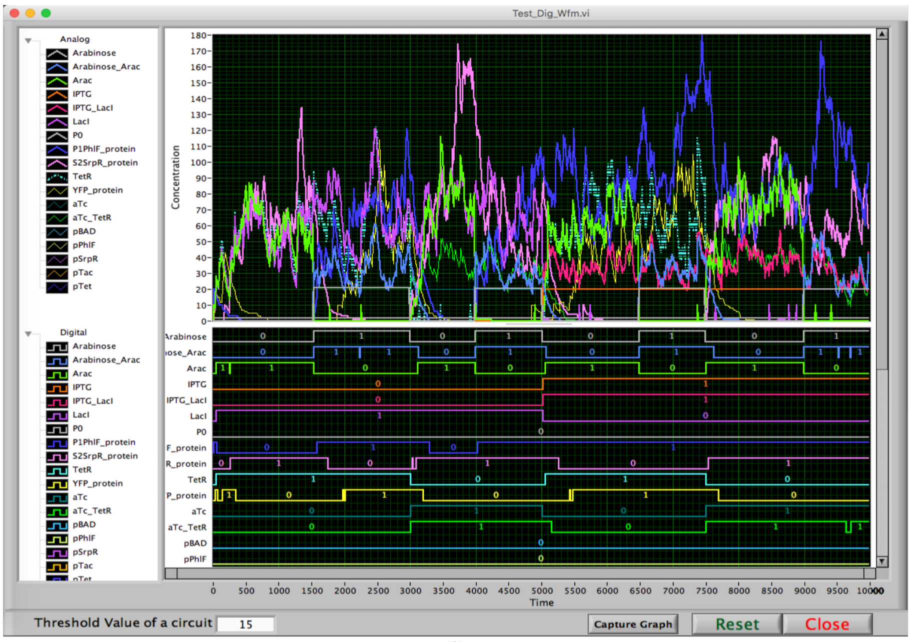


(iii)



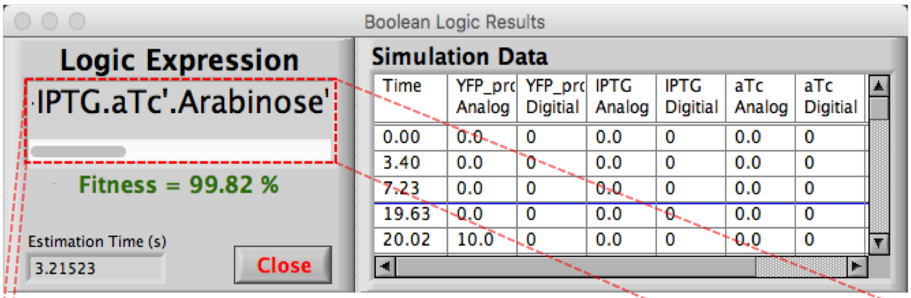
(iv)

(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



Threshold Value of a circuit 15

(i)



Logic Expression

IPTG.aTc'.Arabinose'

Fitness = 99.82 %

Estimation Time (s) 3.21523

Simulation Data

Time	YFP_pro Analog	YFP_pro Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
3.40	0.0	0	0.0	0	0.0	0
7.23	0.0	0	0.0	0	0.0	0
19.63	0.0	0	0.0	0	0.0	0
20.02	10.0	0	0.0	0	0.0	0

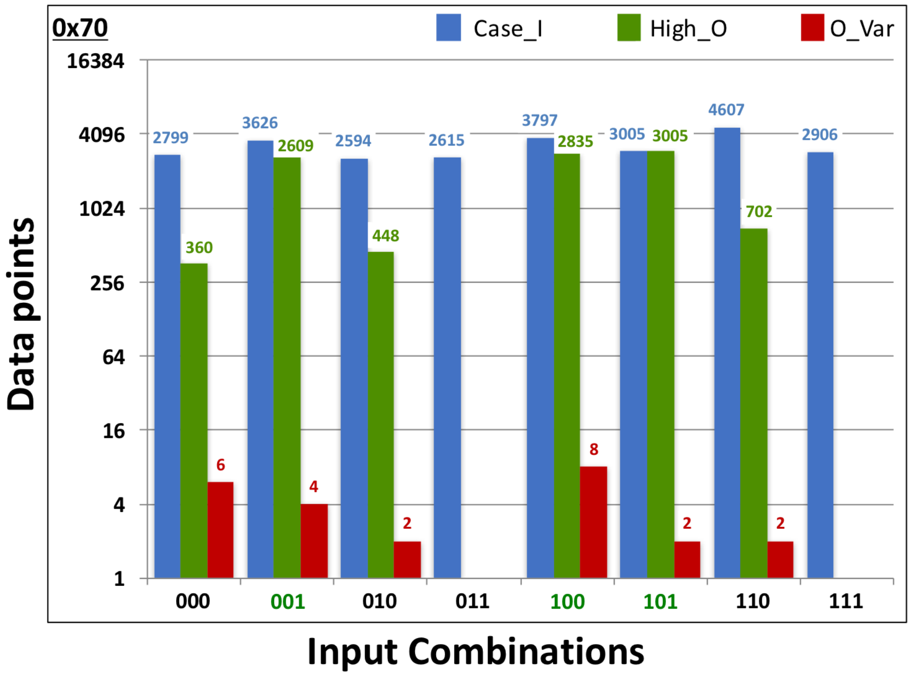
$$\mathrm{YFP} = \mathrm{IPTG}\cdot\overline{\mathrm{aTc}}\cdot\overline{\mathrm{Arabinose}} + \overline{\mathrm{IPTG}}\cdot\overline{\mathrm{aTc}}\cdot\mathrm{Arabinose} + \mathrm{IPTG}\cdot\overline{\mathrm{aTc}}\cdot\mathrm{Arabinose}$$

or

$$\mathrm{YFP} = \mathrm{A}\cdot\overline{\mathrm{B}}\cdot\overline{\mathrm{C}} + \overline{\mathrm{A}}\cdot\overline{\mathrm{B}}\cdot\mathrm{C} + \mathrm{A}\cdot\overline{\mathrm{B}}\cdot\mathrm{C}$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

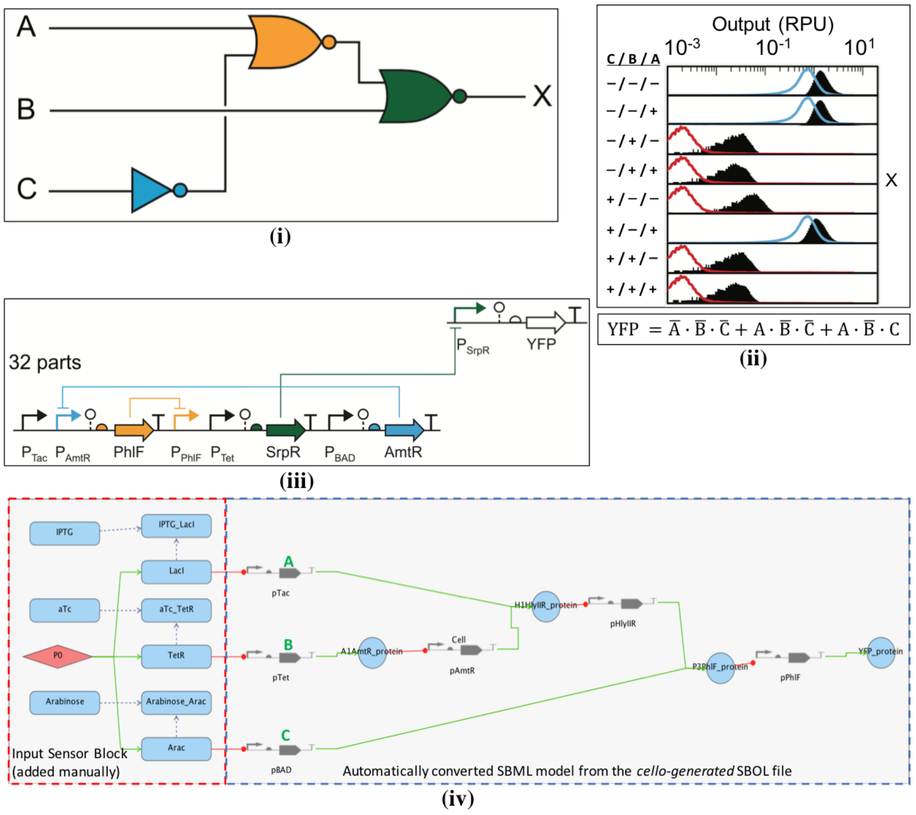


(c) Analytical simulation data for constructing the Boolean expression.

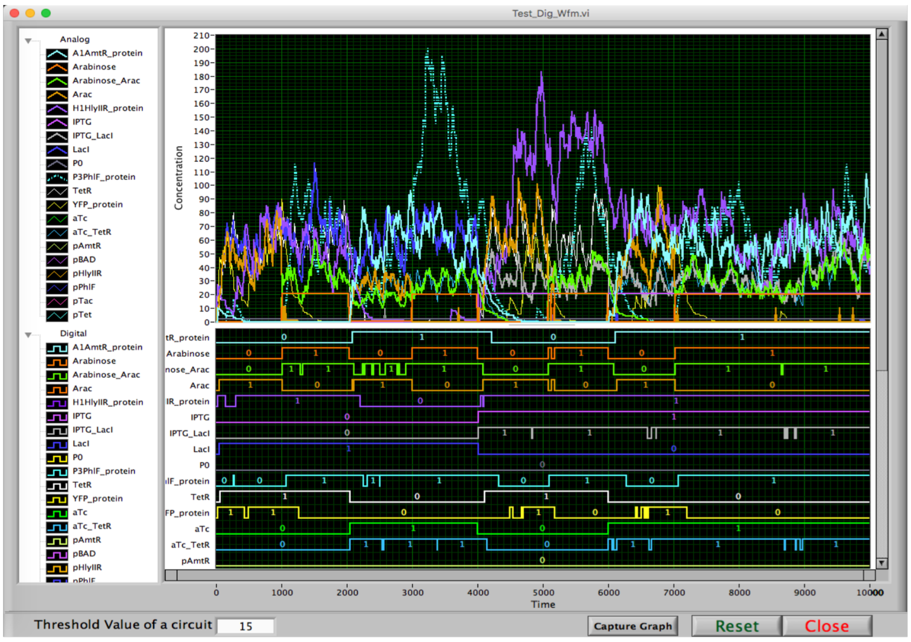
Figure C.6: Logic analysis and simulation results of the genetic 0x70 circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

C.2.2 Genetic circuit 0xC4

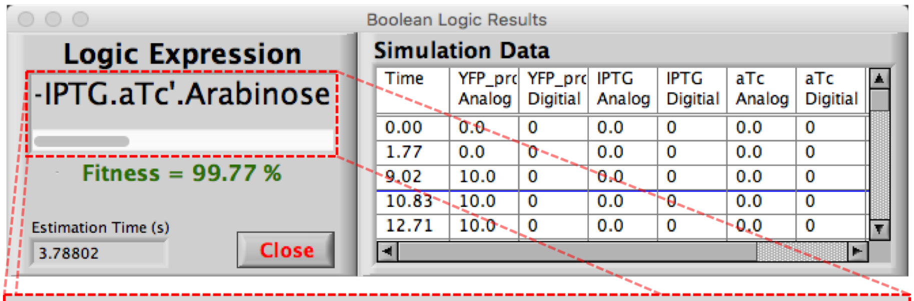
The schematic circuit diagram of the genetic circuit, **0xC4**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.7a(i-iv). The experimental results of this circuit are shown in the Figures C.7b and C.7c, respectively.



(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



(i)



Logic Expression

-IPTG.aTc'.Arabinose

Fitness = 99.77 %

Estimation Time (s)

3.78802

Simulation Data

Time	YFP_prc Analog	YFP_prc Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
1.77	0.0	0	0.0	0	0.0	0
9.02	10.0	0	0.0	0	0.0	0
10.83	10.0	0	0.0	0	0.0	0
12.71	10.0	0	0.0	0	0.0	0

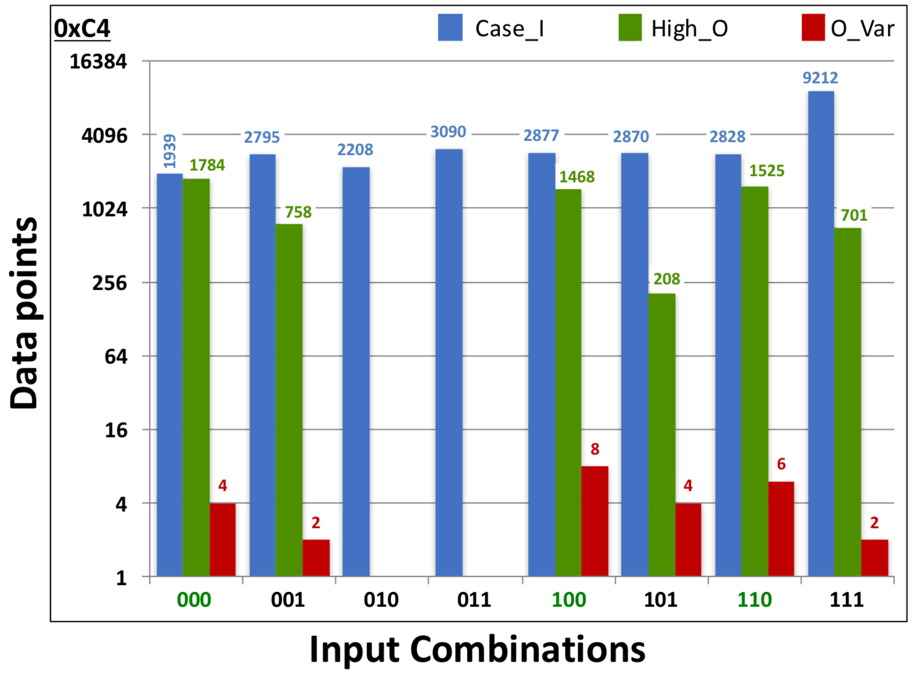
$$\mathrm{YFP} = \overline{\mathrm{IPTG}} \cdot \overline{\mathrm{aTc}} \cdot \overline{\mathrm{Arabinose}} + \mathrm{IPTG} \cdot \overline{\mathrm{aTc}} \cdot \overline{\mathrm{Arabinose}} + \mathrm{IPTG} \cdot \mathrm{aTc} \cdot \overline{\mathrm{Arabinose}}$$

or

$$\mathrm{YFP} = \overline{\mathrm{A}} \cdot \overline{\mathrm{B}} \cdot \overline{\mathrm{C}} + \mathrm{A} \cdot \overline{\mathrm{B}} \cdot \overline{\mathrm{C}} + \mathrm{A} \cdot \mathrm{B} \cdot \overline{\mathrm{C}}$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

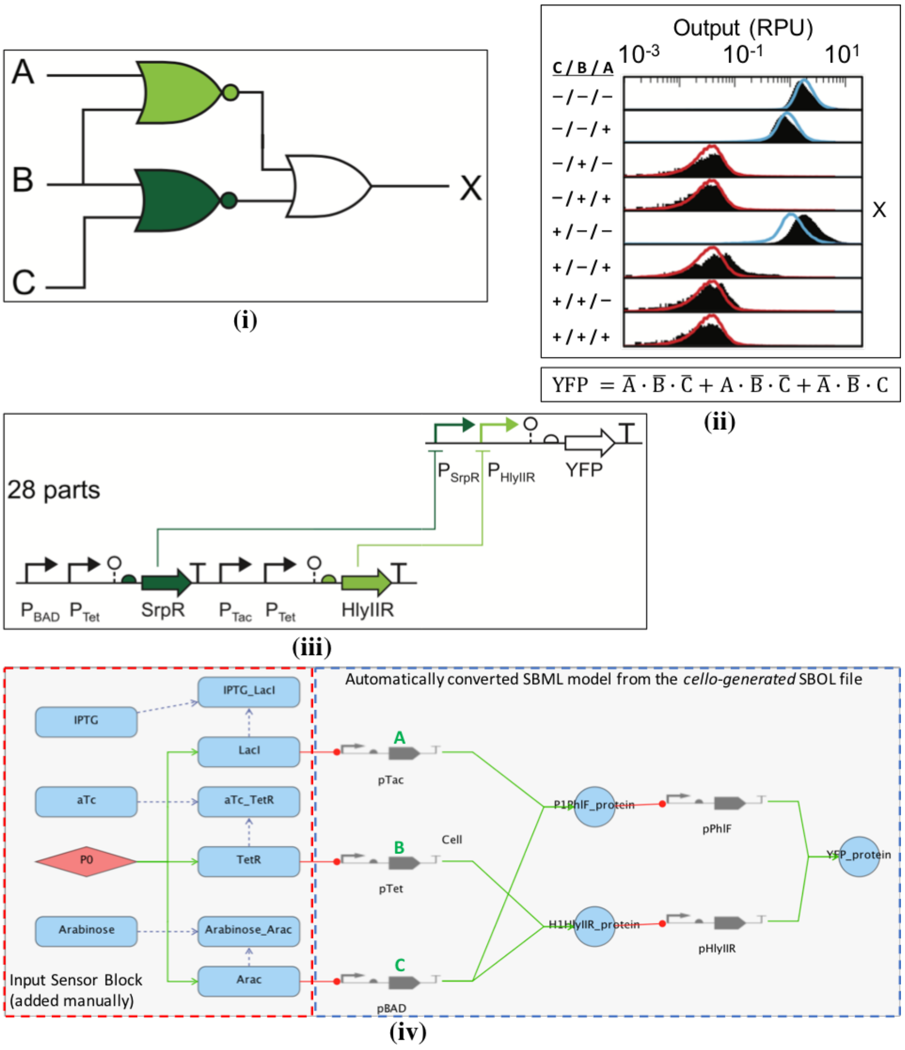


(c) Analytical simulation data for constructing the Boolean expression.

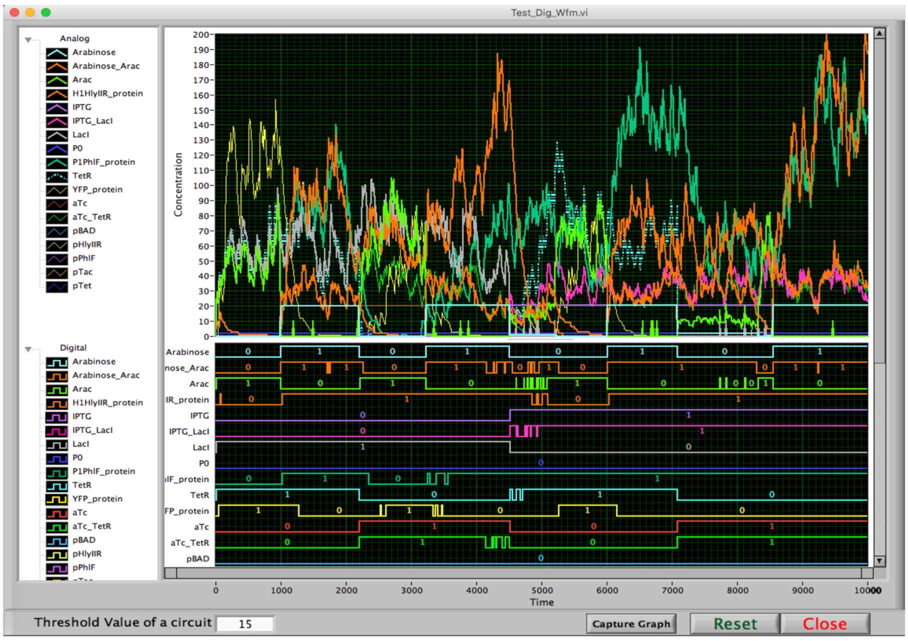
Figure C.7: Logic analysis and simulation results of the genetic 0xC4 circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

C.2.3 Genetic circuit 0xC8

The schematic circuit diagram of, **0xC8**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.8a(i-iv). The experimental results of this circuit are shown in the Figures C.8b and C.8c.

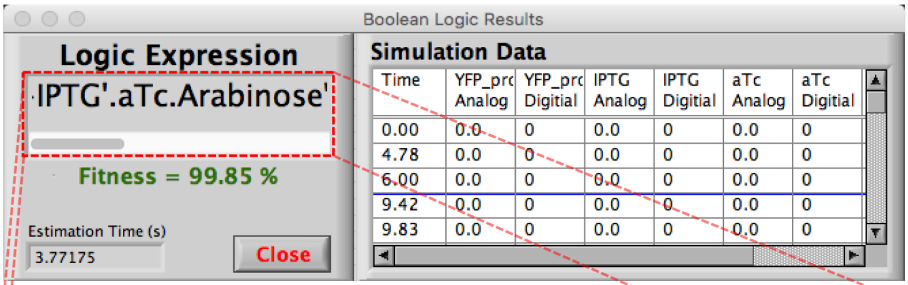


(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



Threshold Value of a circuit 15

(i)



Logic Expression

IPTG'.aTc.Arabinose'

Fitness = 99.85 %

Estimation Time (s) 3.77175

Simulation Data

Time	YFP_prc Analog	YFP_prc Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
4.78	0.0	0	0.0	0	0.0	0
6.00	0.0	0	0.0	0	0.0	0
9.42	0.0	0	0.0	0	0.0	0
9.83	0.0	0	0.0	0	0.0	0

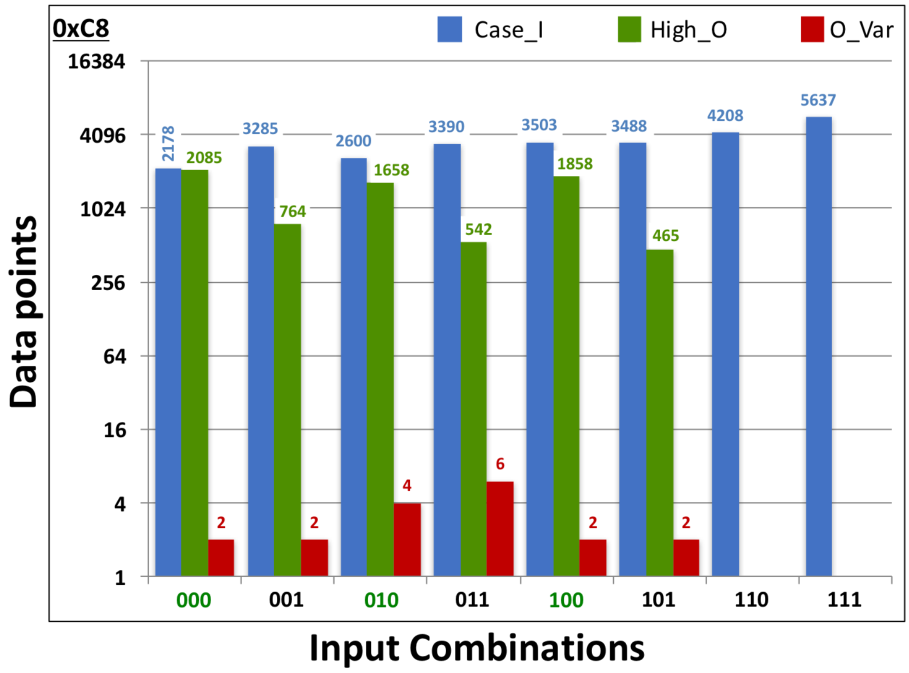
$$\mathrm{YFP} = \overline{\mathrm{IPTG}} \cdot \overline{\mathrm{aTc}} \cdot \overline{\mathrm{Arabinose}} + \mathrm{IPTG} \cdot \overline{\mathrm{aTc}} \cdot \overline{\mathrm{Arabinose}} + \overline{\mathrm{IPTG}} \cdot \mathrm{aTc} \cdot \overline{\mathrm{Arabinose}}$$

or

$$\mathrm{YFP} = \bar{A} \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot \bar{C}$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

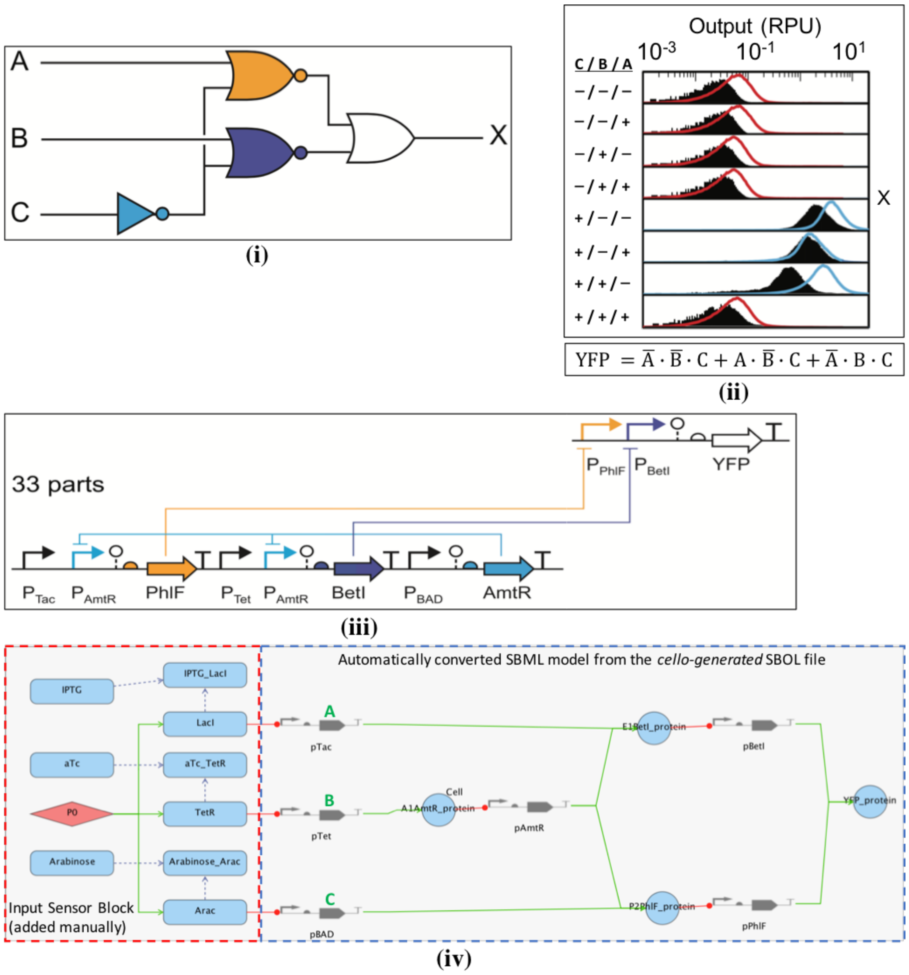


(c) Analytical simulation data for constructing the Boolean expression.

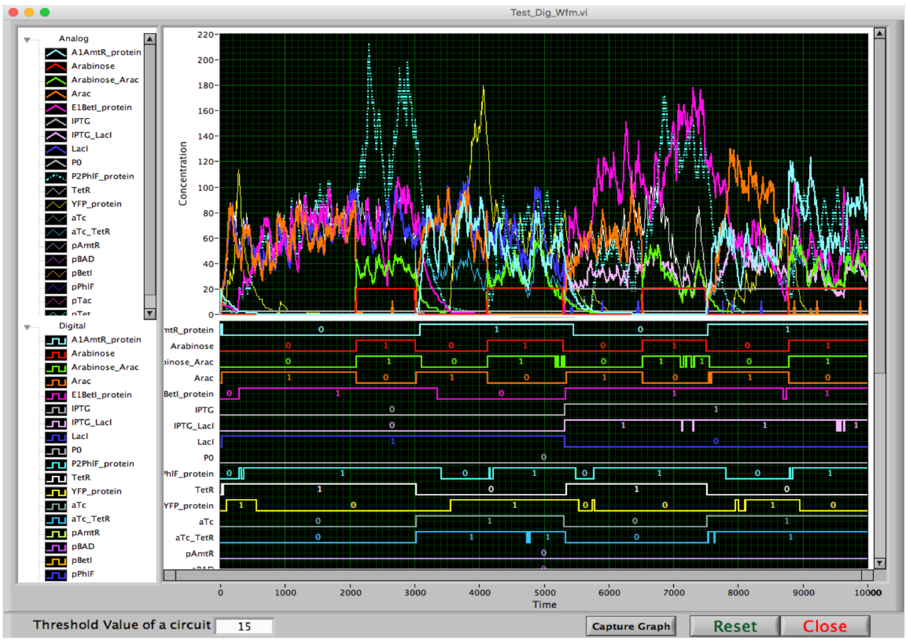
Figure C.8: Logic analysis and simulation results of the genetic 0xC8 circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

C.2.4 Genetic circuit 0x0E

The schematic circuit diagram of, **0x0E**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.9a(i-iv). The experimental results of this circuit are shown in the Figures C.9b and C.9c.

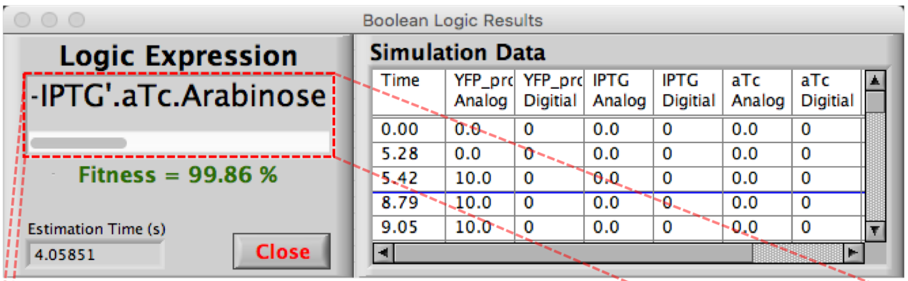


(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



Threshold Value of a circuit 15

(i)



Logic Expression

-IPTG'.aTc.Arabinose

Fitness = 99.86 %

Estimation Time (s) 4.05851

Simulation Data

Time	YFP_pro Analog	YFP_pro Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
5.28	0.0	0	0.0	0	0.0	0
5.42	10.0	0	0.0	0	0.0	0
8.79	10.0	0	0.0	0	0.0	0
9.05	10.0	0	0.0	0	0.0	0

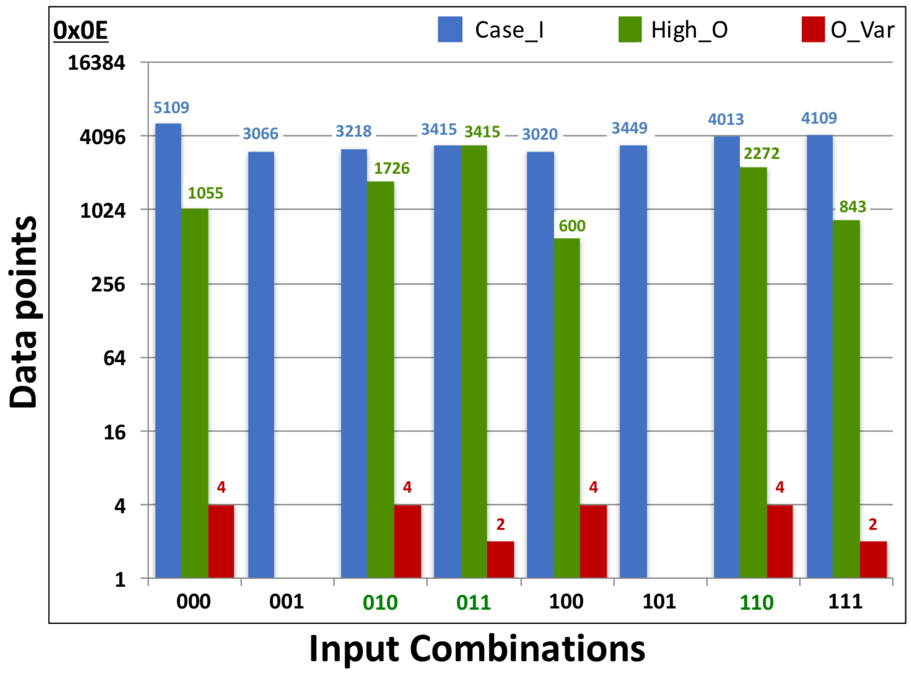
$$\mathrm{YFP} = \overline{\mathrm{IPTG}} \cdot \mathrm{aTc} \cdot \overline{\mathrm{Arabinose}} + \mathrm{IPTG} \cdot \mathrm{aTc} \cdot \overline{\mathrm{Arabinose}} + \overline{\mathrm{IPTG}} \cdot \mathrm{aTc} \cdot \mathrm{Arabinose}$$

or

$$\mathrm{YFP} = \bar{A} \cdot B \cdot \bar{C} + A \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.



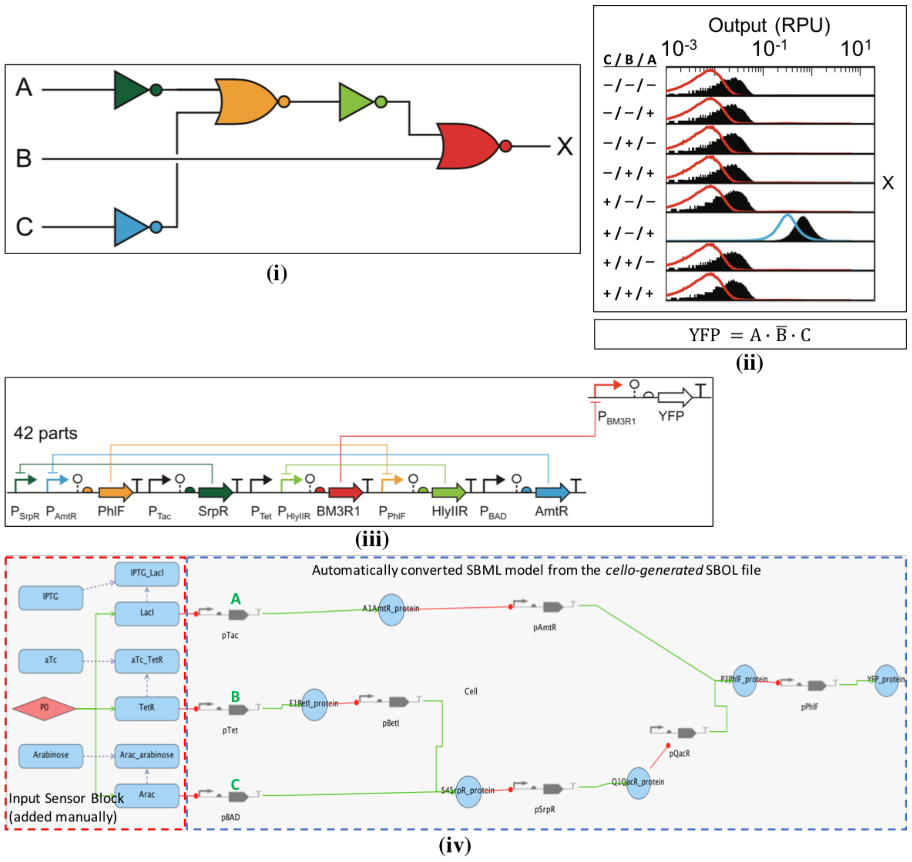
(c) Analytical simulation data for constructing the Boolean expression.

Figure C.9: Logic analysis and simulation results of the genetic 0x0E circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

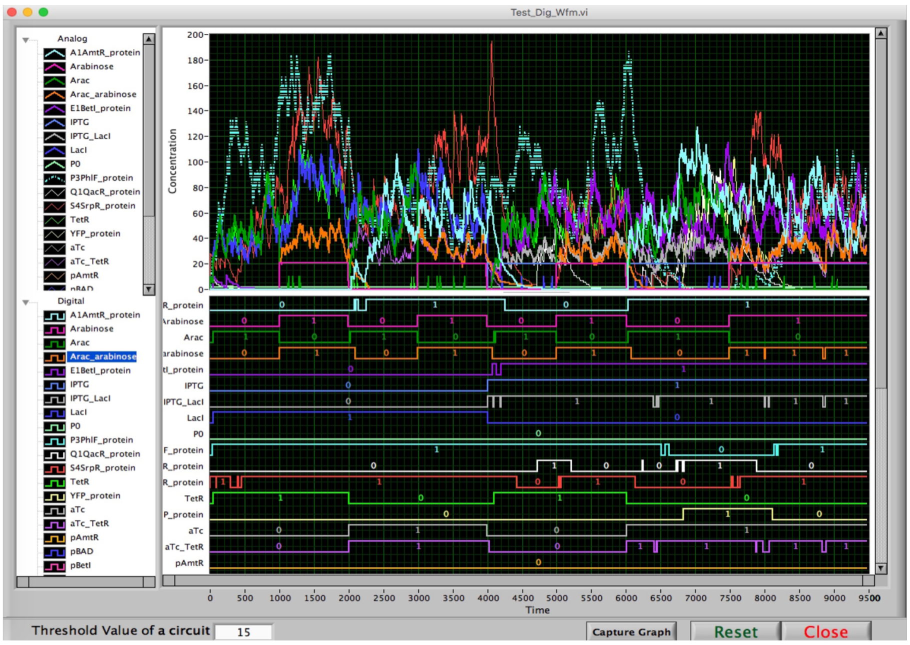
C.2.5 Genetic circuit 0x04

The schematic circuit diagram of, **0x04**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.10a(i-iv). The experimental results of this circuit are shown in the Figures C.10b and C.10c.

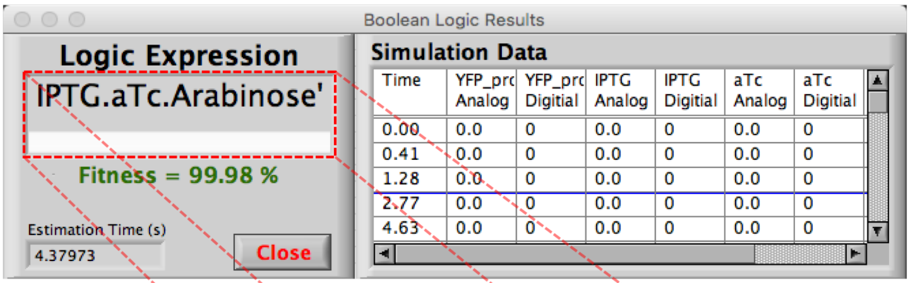
In this circuit, not only the inputs, B and C, are swapped but also the auto-generated SBML model (shown in Figure C.10a(iv)) appears to be different than the circuit diagram (shown in Figure C.10a(i)). However, this change in the circuit's structure still produce the same logic (shown in Figure C.10a(ii)), but with input B and C swapped as shown in Figure C.10b(ii).



(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



(i)



Boolean Logic Results

Logic Expression

IPTG.aTc.Arabinose'

Fitness = 99.98 %

Estimation Time (s): 4.37973

Close

Simulation Data

Time	YFP_prc Analog	YFP_prc Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
0.41	0.0	0	0.0	0	0.0	0
1.28	0.0	0	0.0	0	0.0	0
2.77	0.0	0	0.0	0	0.0	0
4.63	0.0	0	0.0	0	0.0	0

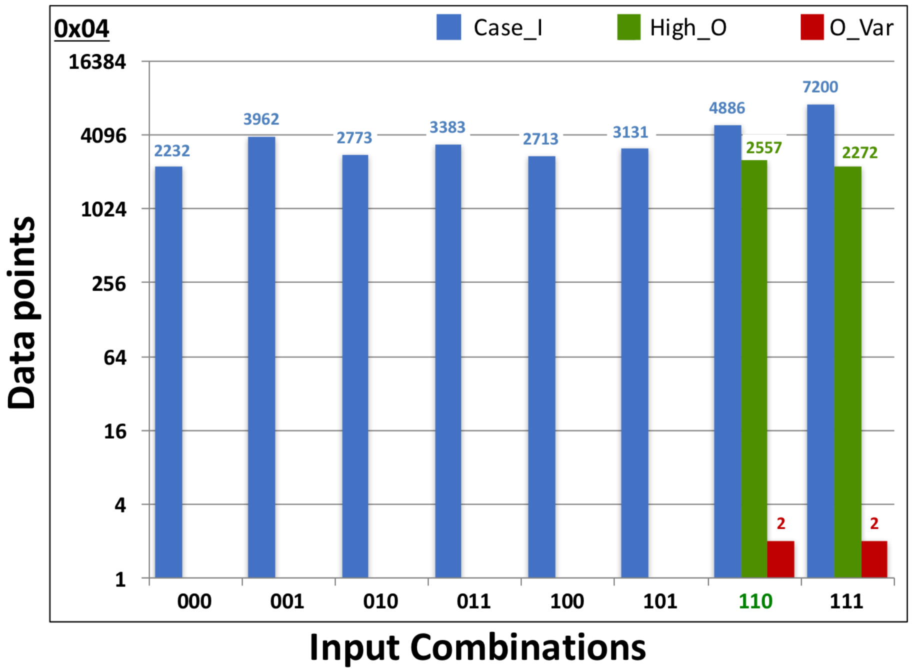
$$\mathrm{YFP} = \mathrm{IPTG} \cdot \mathrm{aTc} \cdot \overline{\mathrm{Arabinose}}$$

or

$$\mathrm{YFP} = \mathrm{A} \cdot \mathrm{B} \cdot \bar{\mathrm{C}}$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

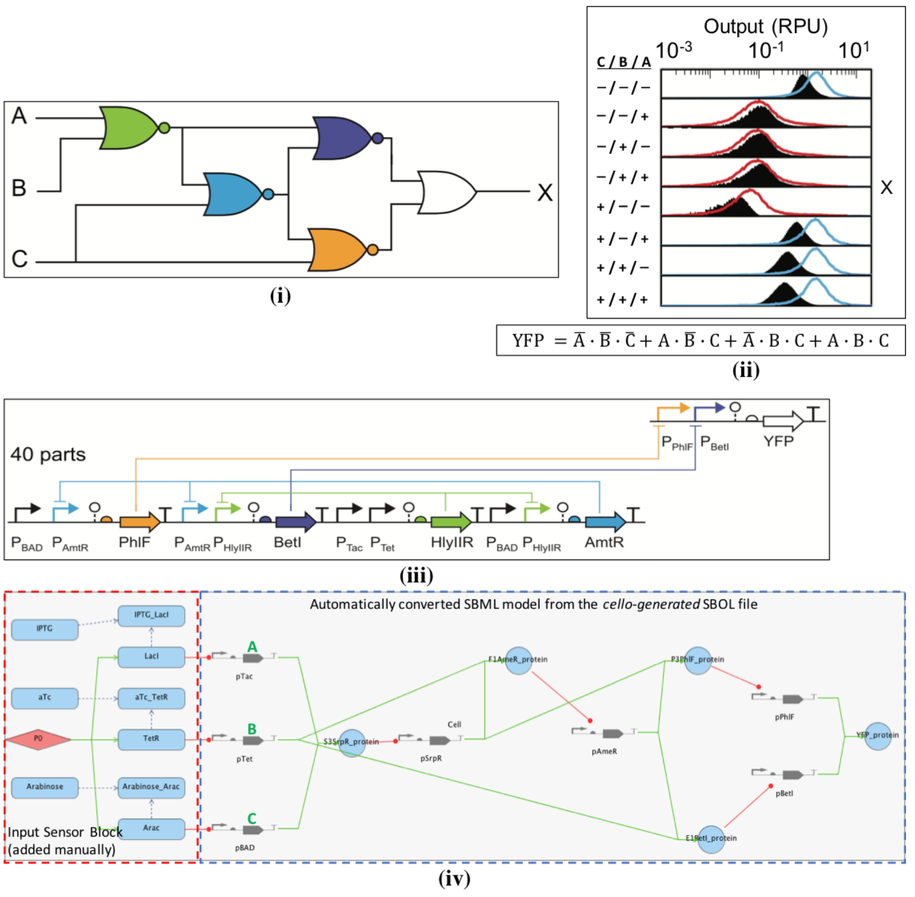


(c) Analytical simulation data for constructing the Boolean expression.

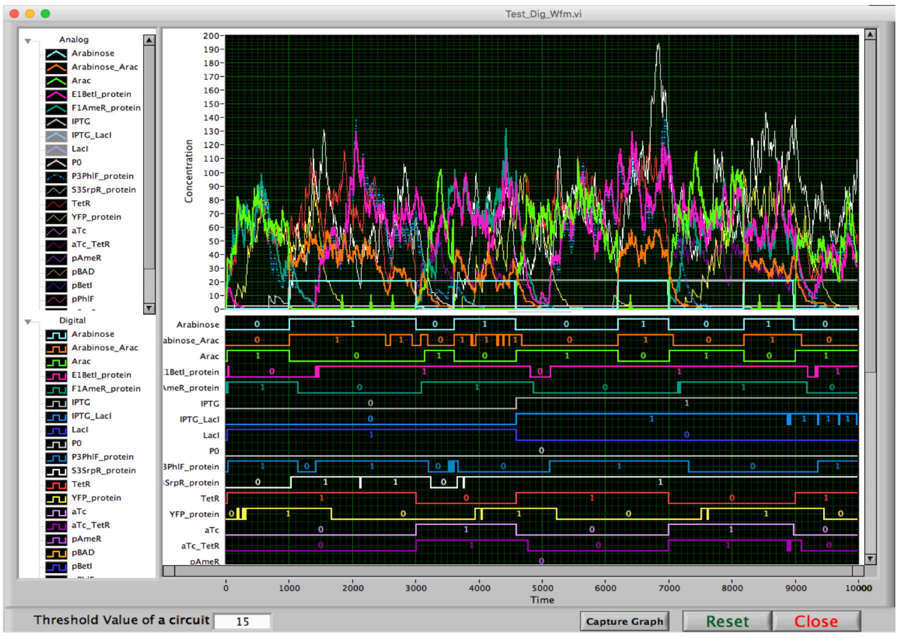
Figure C.10: Logic analysis and simulation results of the genetic 0x04 circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

C.2.6 Genetic circuit 0x87

The schematic circuit diagram of, **0x87**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.11a(i-iv). The experimental results of this circuit are shown in the Figures C.11b and C.11c.

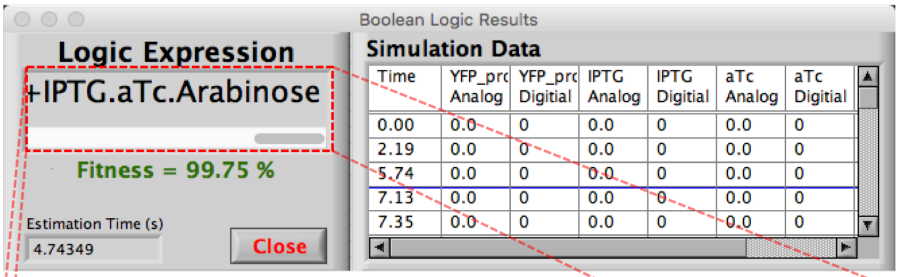

A
B
C
X
(i)
Output (RPU)
10^{-3} 10^{-1} 10^{1}
C / B / A
− / − / −
− / − / +
− / + / −
− / + / +
+ / − / −
+ / − / +
+ / + / −
+ / + / +
X
YFP $= \bar{A}\cdot\bar{B}\cdot\bar{C} + A\cdot\bar{B}\cdot C + \bar{A}\cdot B\cdot C + A\cdot B\cdot C$
(ii)
40 parts
P_{PhlF} P_{BetI} YFP
P_{BAD} P_{AmtR} PhlF P_{AmtR} P_{HlyIIR} BetI P_{Tac} P_{Tet} HlyIIR P_{BAD} P_{HlyIIR} AmtR
(iii)
Automatically converted SBML model from the *cello-generated* SBOL file
IPTG
IPTG_LacI
LacI
aTc
aTc_TetR
P0
TetR
Arabinose
Arabinose_Arac
Arac
Input Sensor Block (added manually)
A
pTac
B
pTet
C
pBAD
S3SrpR_protein
Cell
pSrpR
F1AmeR_protein
pAmeR
P3PhlF_protein
pPhlF
pBetI
E1BetI_protein
YFP_protein
(iv)

(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



Threshold Value of a circuit 15

(i)



Logic Expression

+IPTG.aTc.Arabinose

Fitness = 99.75 %

Estimation Time (s) 4.74349

Simulation Data

Time	YFP_prc Analog	YFP_prc Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
2.19	0.0	0	0.0	0	0.0	0
5.74	0.0	0	0.0	0	0.0	0
7.13	0.0	0	0.0	0	0.0	0
7.35	0.0	0	0.0	0	0.0	0

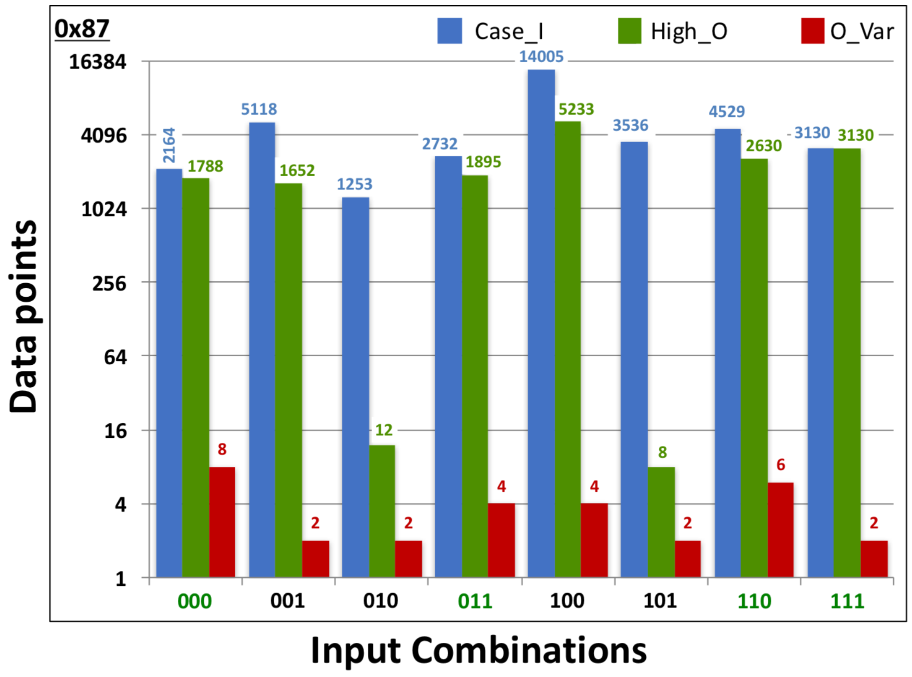
$$\mathrm{YFP} = \overline{\mathrm{IPTG}}\cdot\overline{\mathrm{aTc}}\cdot\overline{\mathrm{Arabinose}} + \mathrm{IPTG}\cdot\mathrm{aTc}\cdot\overline{\mathrm{Arabinose}} + \overline{\mathrm{IPTG}}\cdot\mathrm{aTc}\cdot\mathrm{Arabinose} + \mathrm{IPTG}\cdot\mathrm{aTc}\cdot\mathrm{Arabinose}$$

or

$$\mathrm{YFP} = \bar{A}\cdot\bar{B}\cdot\bar{C} + A\cdot B\cdot\bar{C} + \bar{A}\cdot B\cdot C + A\cdot B\cdot C$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

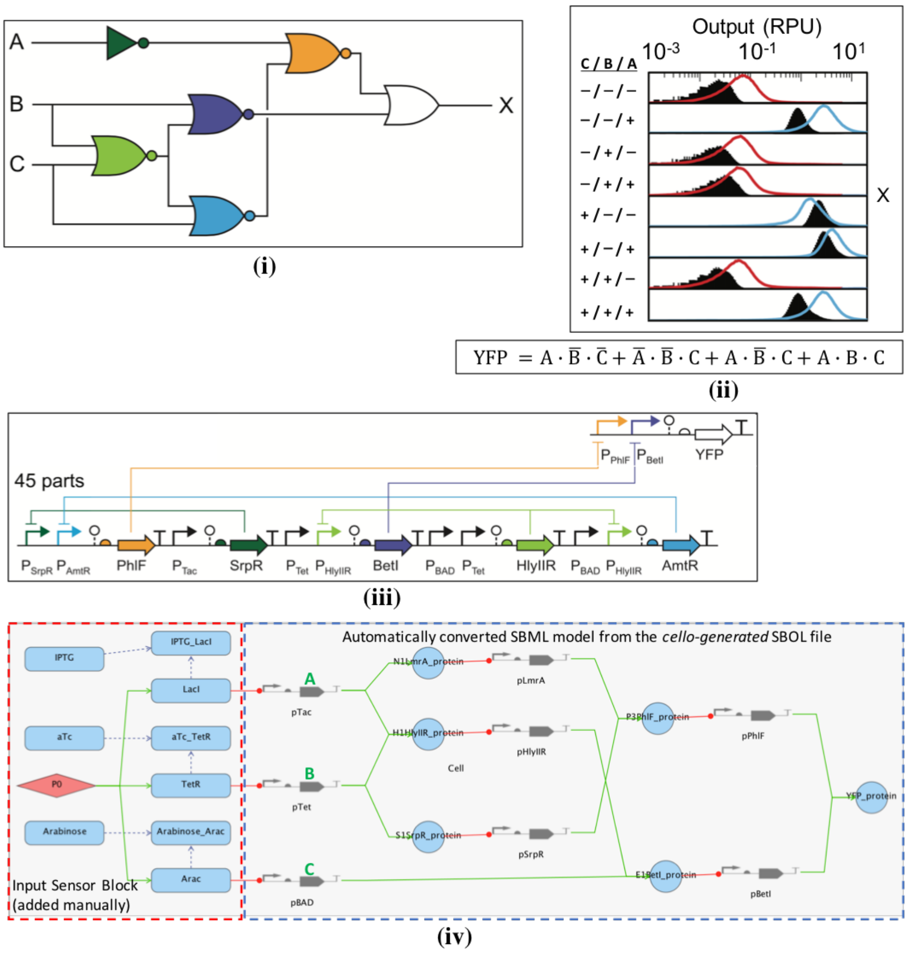


(c) Analytical simulation data for constructing the Boolean expression.

Figure C.11: Logic analysis and simulation results of the genetic 0x87 circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

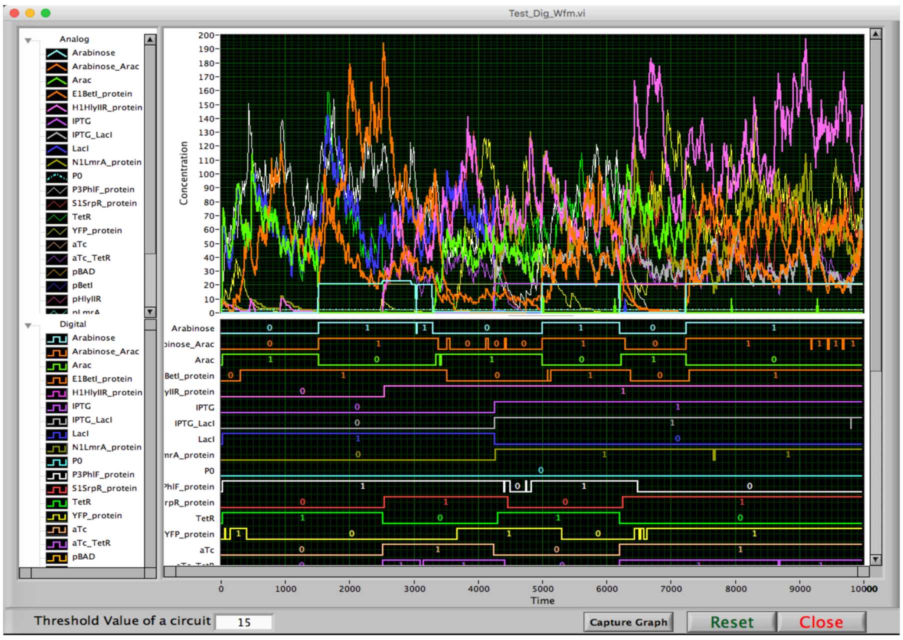
C.2.7 Genetic circuit 0x4D

The schematic circuit diagram of, **0x4D**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.12a(i-iv). The experimental results of this circuit are shown in the Figures C.12b and C.12c. The auto-generated SBML model (shown in Figure C.12a(iv)) appears to be different than the circuit diagram (shown in Figure C.12a(i)). However, this change in the circuit's structure still produce the same logic (shown in Figure C.12a(ii)), but with inputs B and C swapped as shown in Figure C.12b(ii).



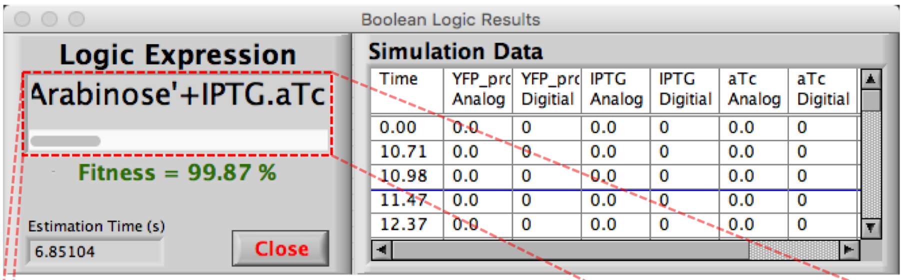
$$\mathrm{YFP} = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot C$$

(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



Threshold Value of a circuit 15

(i)



Logic Expression

Arabinose'+IPTG.aTc

Fitness = 99.87 %

Estimation Time (s) 6.85104

Simulation Data

Time	YFP_pro Analog	YFP_pro Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
10.71	0.0	0	0.0	0	0.0	0
10.98	0.0	0	0.0	0	0.0	0
11.47	0.0	0	0.0	0	0.0	0
12.37	0.0	0	0.0	0	0.0	0

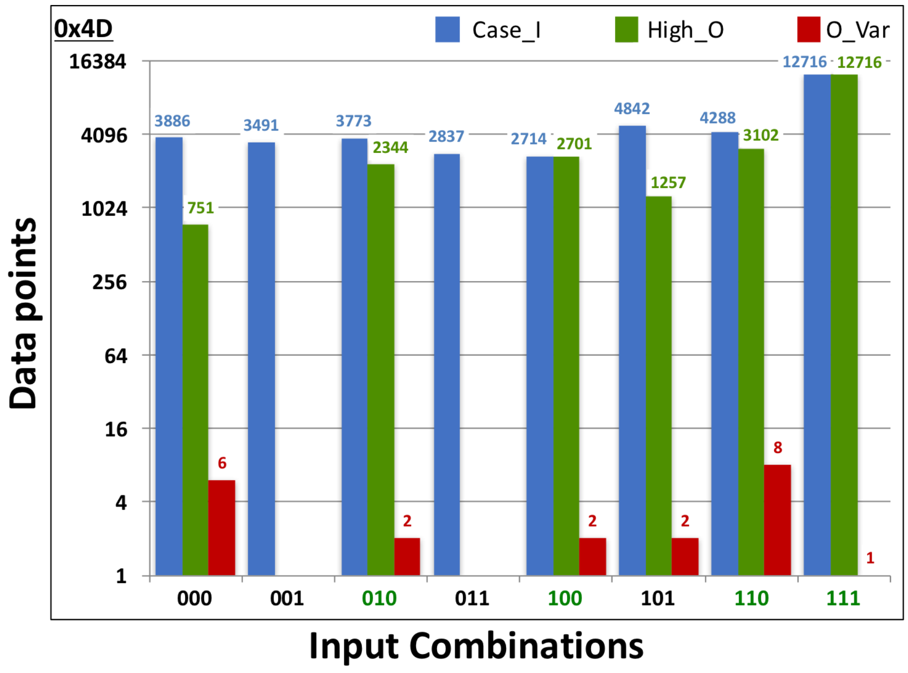
$$\mathrm{YFP} = \mathrm{IPTG}\cdot\overline{\mathrm{aTc}}\cdot\overline{\mathrm{Arabinose}} + \overline{\mathrm{IPTG}}\cdot\mathrm{aTc}\cdot\overline{\mathrm{Arabinose}} + \mathrm{IPTG}\cdot\mathrm{aTc}\cdot\overline{\mathrm{Arabinose}} + \mathrm{IPTG}\cdot\mathrm{aTc}\cdot\mathrm{Arabinose}$$

or

$$\mathrm{YFP} = \mathrm{A}\cdot\overline{\mathrm{B}}\cdot\overline{\mathrm{C}} + \overline{\mathrm{A}}\cdot\mathrm{B}\cdot\overline{\mathrm{C}} + \mathrm{A}\cdot\mathrm{B}\cdot\overline{\mathrm{C}} + \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{C}$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

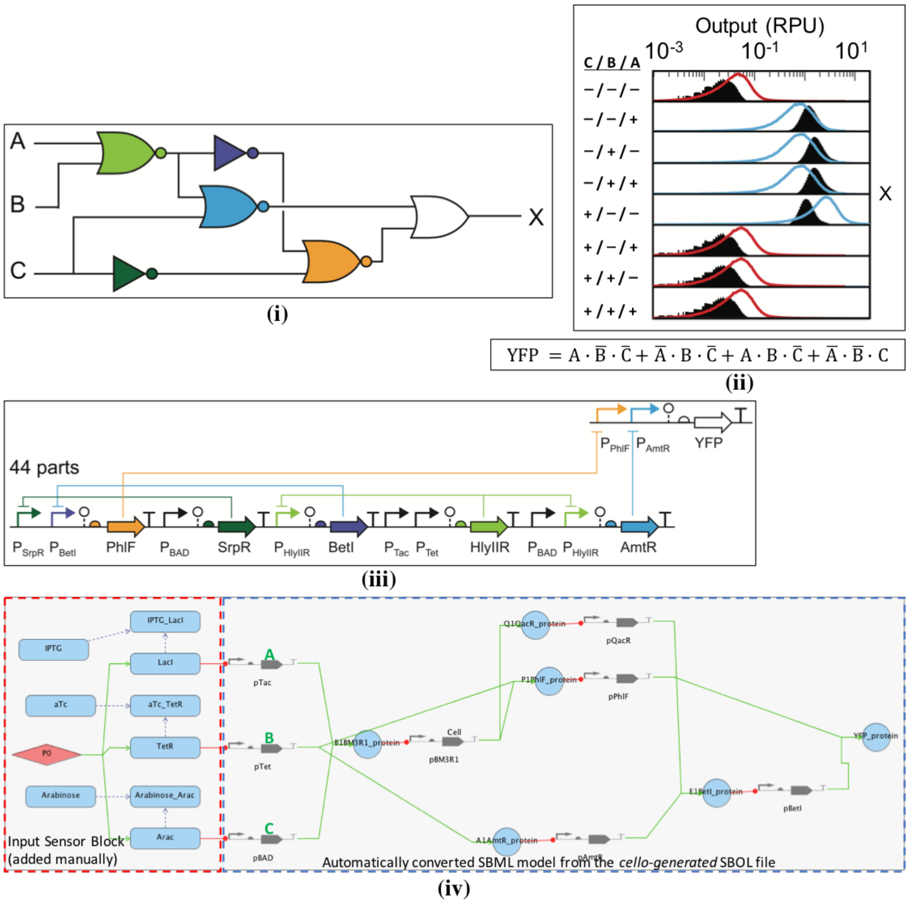


(c) Analytical simulation data for constructing the Boolean expression.

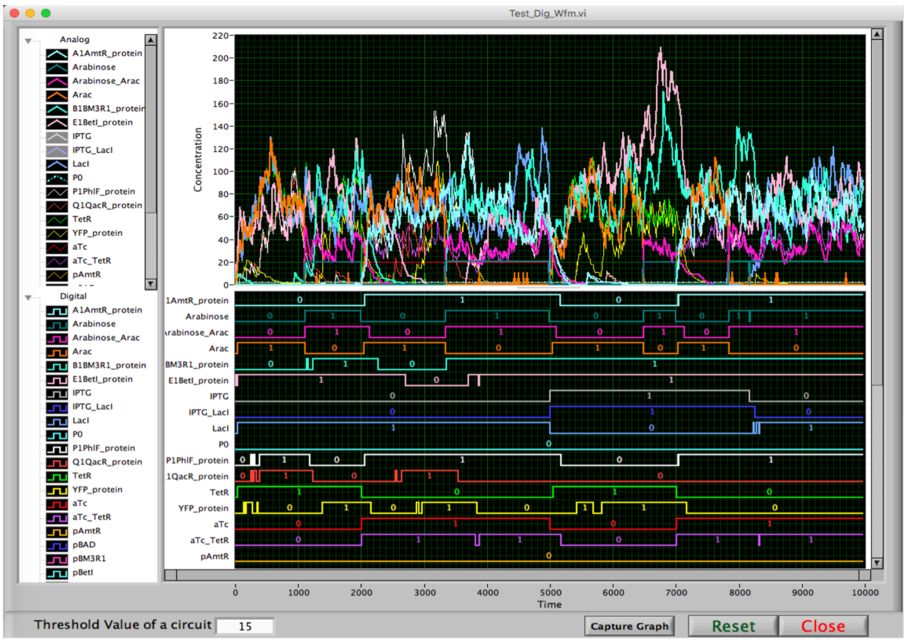
Figure C.12: Logic analysis and simulation results of the genetic 0x4D circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

C.2.8 Genetic circuit 0x78

The schematic circuit diagram of, **0x78**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.13a(i-iv). The experimental results of this circuit are shown in the Figures C.13b and C.13c.

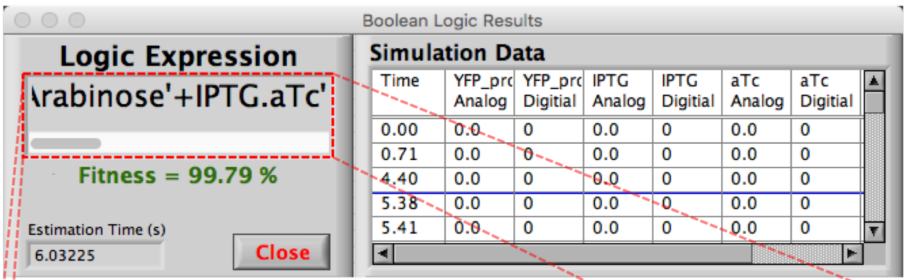

A
B
C
X
(i)
Output (RPU)
C / B / A
10^{-3}
10^{-1}
10^{1}
−/−/−
−/−/+
−/+/−
−/+/+
+/−/−
+/−/+
+/+/−
+/+/+
X
YFP $= A\cdot\bar{B}\cdot\bar{C} + \bar{A}\cdot B\cdot\bar{C} + A\cdot B\cdot\bar{C} + \bar{A}\cdot\bar{B}\cdot C$
(ii)
44 parts
P_{PhlF} P_{AmtR} YFP
P_{SrpR} P_{BetI} PhlF P_{BAD} SrpR P_{HlyIIR} BetI P_{Tac} P_{Tet} HlyIIR P_{BAD} P_{HlyIIR} AmtR
(iii)
IPTG
IPTG_LacI
LacI
aTc
aTc_TetR
TetR
P0
Arabinose
Arabinose_Arac
Arac
Input Sensor Block (added manually)
A
pTac
B
pTet
C
pBAD
B1BM3R1_protein
Cell
pBM3R1
Q1QacR_protein
pQacR
P1PhlF_protein
pPhlF
A1AmtR_protein
pAmtR
E1BetI_protein
pBetI
YFP_protein
Automatically converted SBML model from the *cello-generated* SBOL file
(iv)

(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



Threshold Value of a circuit 15

(i)



Logic Expression

Arabinose'+IPTG.aTc'

Fitness = 99.79 %

Estimation Time (s) 6.03225

Simulation Data

Time	YFP_prc Analog	YFP_prc Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
0.71	0.0	0	0.0	0	0.0	0
4.40	0.0	0	0.0	0	0.0	0
5.38	0.0	0	0.0	0	0.0	0
5.41	0.0	0	0.0	0	0.0	0

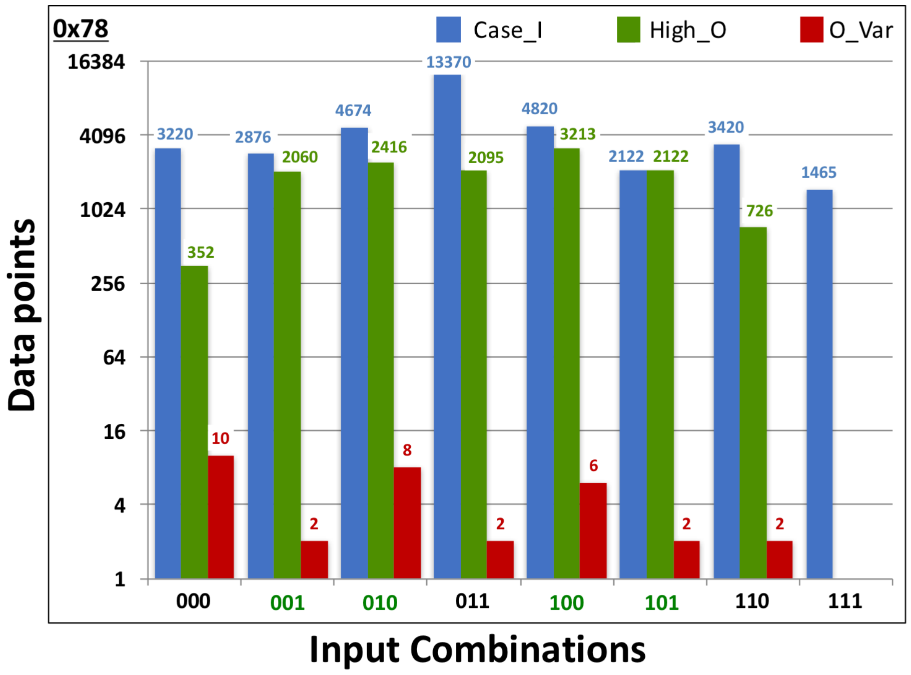
$$\mathrm{YFP} = \mathrm{IPTG}\cdot\overline{\mathrm{aTc}}\cdot\overline{\mathrm{Arabinose}} + \overline{\mathrm{IPTG}}\cdot\overline{\mathrm{aTc}}\cdot\mathrm{Arabinose} + \mathrm{IPTG}\cdot\overline{\mathrm{aTc}}\cdot\mathrm{Arabinose} + \overline{\mathrm{IPTG}}\cdot\mathrm{aTc}\cdot\overline{\mathrm{Arabinose}}$$

or

$$\mathrm{YFP} = \mathrm{A}\cdot\overline{\mathrm{B}}\cdot\overline{\mathrm{C}} + \overline{\mathrm{A}}\cdot\overline{\mathrm{B}}\cdot\mathrm{C} + \mathrm{A}\cdot\overline{\mathrm{B}}\cdot\mathrm{C} + \overline{\mathrm{A}}\cdot\mathrm{B}\cdot\overline{\mathrm{C}}$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.

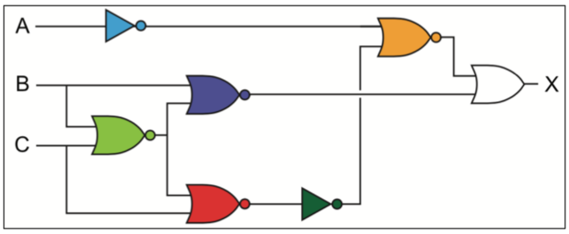


(c) Analytical simulation data for constructing the Boolean expression.

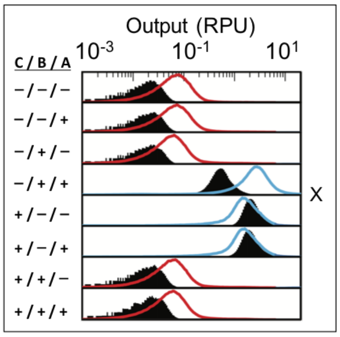
Figure C.13: Logic analysis and simulation results of the genetic 0x78 circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

C.2.9 Genetic circuit 0x1C

The schematic circuit diagram of, **0x1C**, including its response table, the SBOL representation and the corresponding converted SBML representation are shown in Figure C.14a(i-iv). The experimental results of this circuit are shown in the Figures C.14b and C.14c. The auto-generated SBML model (shown in Figure C.14a(iv)) appears to be different than the circuit diagram (shown in Figure C.14a(i)). However, this change in the circuit's structure still produce the same logic (shown in Figure C.14a(ii)), but with inputs B and C swapped as shown in Figure C.14b(ii).

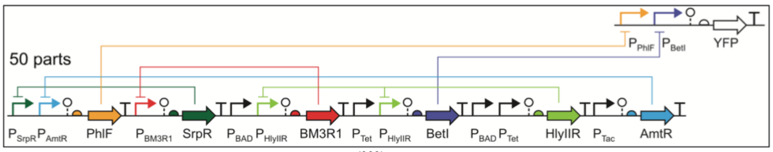


(i)

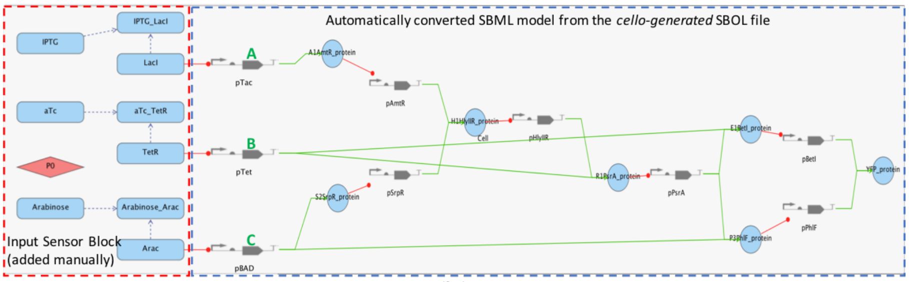


$$\text{YFP} = A \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C$$

(ii)

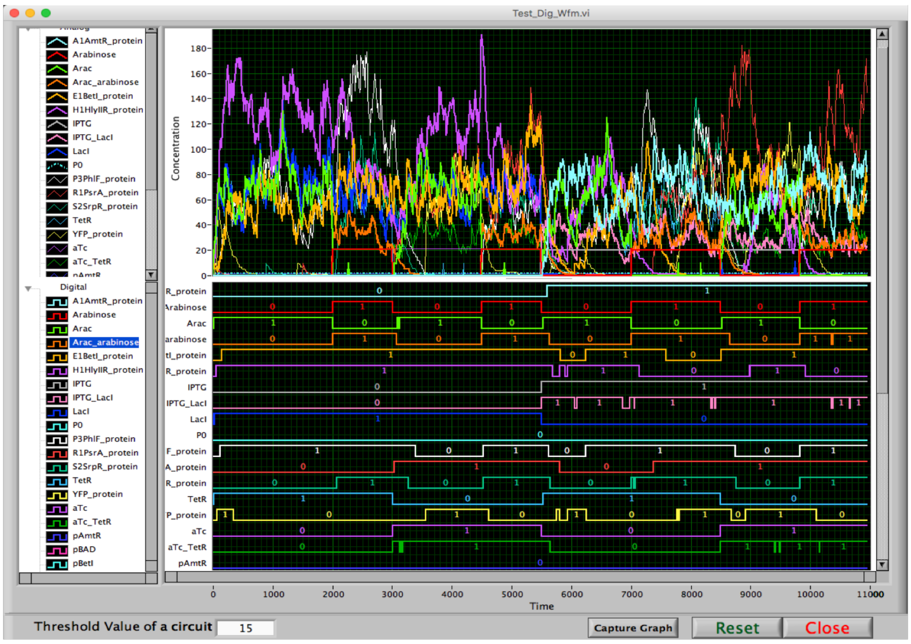


(iii)

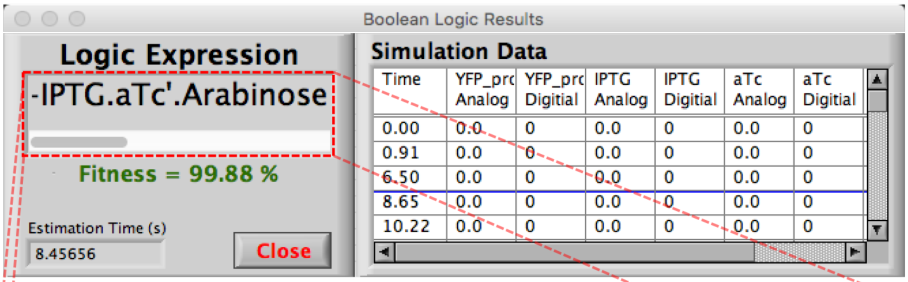


(iv)

(a) (i) Circuit schematic. (ii) Circuit behavior and its Boolean expression. (iii) SBOL representation. (iv) Converted SBML model.



(i)


Boolean Logic Results

Logic Expression

-IPTG.aTc'.Arabinose

Fitness = 99.88 %

Estimation Time (s)

8.45656

Close

Simulation Data

Time	YFP_prc Analog	YFP_prc Digitial	IPTG Analog	IPTG Digitial	aTc Analog	aTc Digitial
0.00	0.0	0	0.0	0	0.0	0
0.91	0.0	0	0.0	0	0.0	0
6.50	0.0	0	0.0	0	0.0	0
8.65	0.0	0	0.0	0	0.0	0
10.22	0.0	0	0.0	0	0.0	0

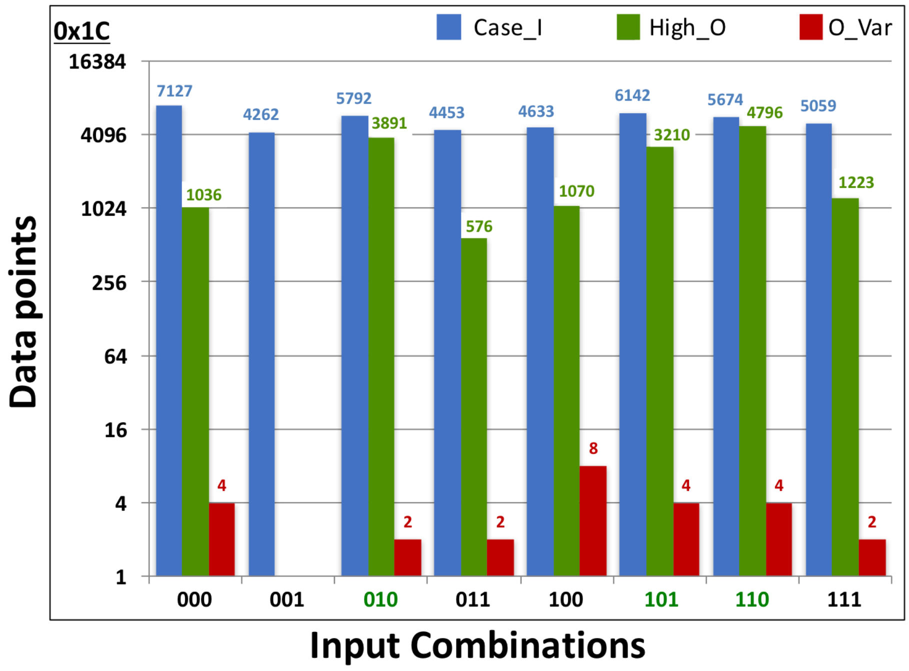
$$\text{YFP} = \text{IPTG}\cdot\overline{\text{aTc}}\cdot\text{Arabinose} + \overline{\text{IPTG}}\cdot\text{aTc}\cdot\overline{\text{Arabinose}} + \text{IPTG}\cdot\text{aTc}\cdot\overline{\text{Arabinose}}$$

or

$$\text{YFP} = \text{A}\cdot\overline{\text{B}}\cdot\text{C} + \overline{\text{A}}\cdot\text{B}\cdot\overline{\text{C}} + \text{A}\cdot\text{B}\cdot\overline{\text{C}}$$

(ii)

(b) (i) Circuit behavior for all possible input combinations. (ii) Boolean expression estimated by the logic analysis algorithm.



(c) Analytical simulation data for constructing the Boolean expression.

Figure C.14: Logic analysis and simulation results of the genetic 0x1C circuit [14]. (a) Circuit description. (b) Analog and digital simulation traces with Boolean logic estimation. (c) Analytical simulation data used for constructing the Boolean expression.

C.2.10 Discussion

The logic analyses of these circuit indicates that the proposed algorithm obtained the correct Boolean logic with inputs B and C swapped with each other. This interchange of signals B and C is also reflected in the auto-generated SBML models of all circuits as shown in the image (iv) in each of the Figures C.6a-C.14a.

The auto-generated SBML models of the circuits, 0x04, 0x4D, and 0x1C, indicate that their internal structures are different than those shown in [14]. However, the structural changes in these circuits do not entirely change the functionality of these circuits except that, similar to other circuits, the inputs B and C are interchanged. Thus, these logic analyses experimentation, using D-VASim, is found very useful in identifying or verifying the correct Boolean logic of a circuit.

Appendix D

GeneTech – *Supplementary Data*

The experimental results of the remaining four circuits, 0x70, 0xC4, 0xC8 and 0x0E are given below in Figures D.1, D.2, D.3 and D.4, respectively.

In case of circuit x70 (as mentioned before in Chapter 6), no circuit is generated for the Boolean logic obtained from D-VASim (with inputs B and C swapped), because there is no NOR gate available in the genetic gates library (shown in Figure 6.7) with P_{Tac} and P_{Bad} promoters, as inputs.

Also, in the case of 0xC8 (for the logic expression obtained from Cello paper [14]), the components in circuit 1 and 2 are same, however the input source of generating SrpR and AmtR proteins are interchanged in both circuits.

Similarly for 0xC8, the circuit components in circuit 4 (in case of Cello original expression [14]) seems similar to the circuit components in circuit 1 (for the logic expression obtained from D-VASim) as shown in Figure D.3. However, the difference is in the input promoters generating the proteins - the activity of promoters P_{HlYllR} and P_{Tet} controls the production of protein, *BM3R1* in the former case, and the protein *BM3R1* is controlled by the activities of promoters P_{HlYllR} and P_{Bad} in the later case. Similar observations can also be made for other circuits.

Circuit 0x70

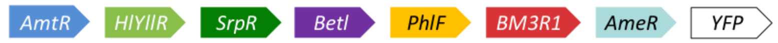
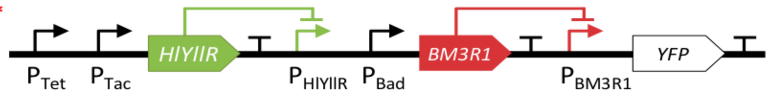
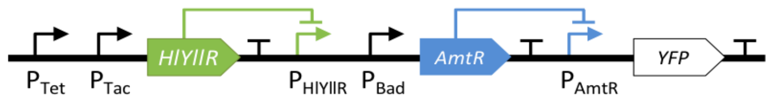
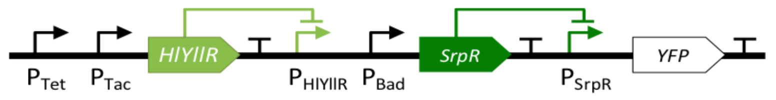
Boolean Logic obtained from	Optimization			Synthesis	Technology Mapping
	t_E (ms)	$\mathrm{Exp}_{\mathrm{Init}}$	$\mathrm{Exp}_{\mathrm{Opt}}$		 AmtR HlYlIR SrpR BetI PhlF BM3R1 AmeR YFP
Cello Paper	20	$AB\bar{C}+\bar{A}B\bar{C}+A\bar{B}\bar{C}$	$\bar{C}(A\bar{B}+B)$	$\overline{(\mathbf{C}+\overline{(\mathbf{B}+\mathbf{A})})}$	1.*  HlYlIR BM3R1 YFP P_{Tet} P_{Tac} P_{HlYlIR} P_{Bad} P_{BM3R1}
					2.  HlYlIR AmtR YFP P_{Tet} P_{Tac} P_{HlYlIR} P_{Bad} P_{AmtR}
					3.  HlYlIR SrpR YFP P_{Tet} P_{Tac} P_{HlYlIR} P_{Bad} P_{SrpR}
D-VASim Logic Analyzer	20	$A\bar{B}C+\bar{A}\bar{B}C+A\bar{B}\bar{C}$	$\bar{B}(A\bar{C}+C)$	$\overline{(\mathbf{B}+\overline{(\mathbf{C}+\mathbf{A})})}$	**There is no NOR gate available in the gates library with A and C (P_{Tac} and P_{Bad}) as inputs.**

Figure D.1: Experimental results of *Gene***Tech** for the optimization, synthesis and technology mapping of 0x70 circuit.

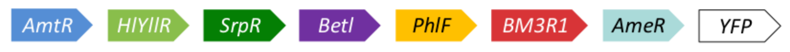
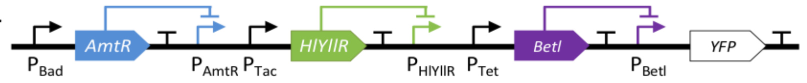
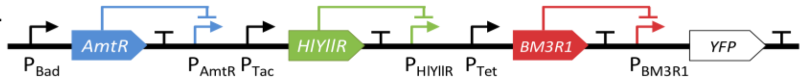
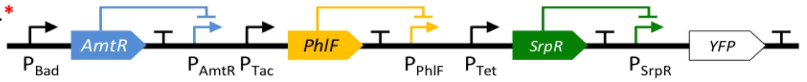
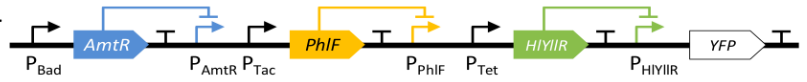
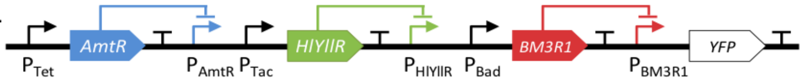
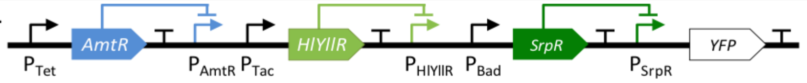
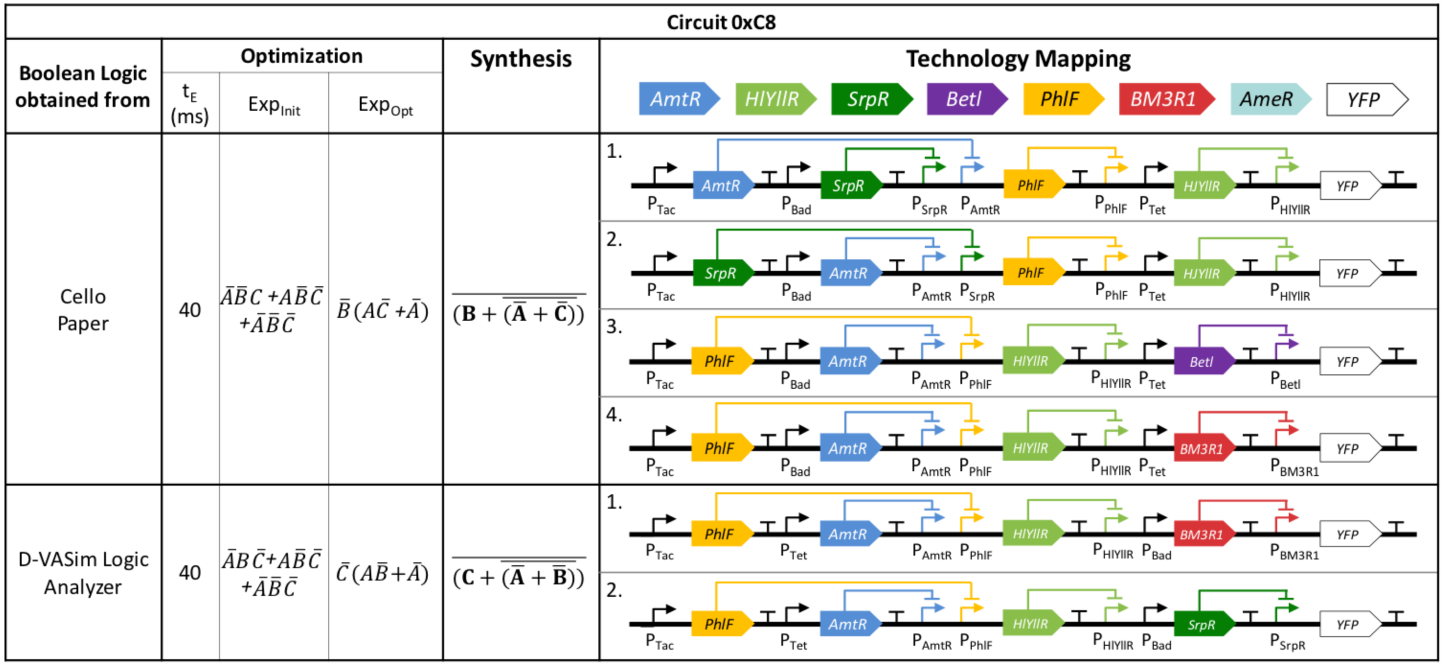
Circuit 0xC4					
Boolean Logic obtained from	Optimization			Synthesis	Technology Mapping  AmtR HlYlIR SrpR BetI PhlF BM3R1 AmeR YFP
	t_E (ms)	Exp_{Init}	Exp_{Opt}		
Cello Paper	30	$\bar{A}\bar{B}\bar{C}+A\bar{B}\bar{C}+A\bar{B}C$	$\bar{B}(\bar{A}\bar{C}+A)$	$\overline{(\mathbf{B}+\overline{(\mathbf{A}+\bar{\mathbf{C}})})}$	1.  AmtR HlYlIR BetI YFP P_{Bad} P_{AmtR} P_{Tac} P_{HlYlIR} P_{Tet} P_{BetI}
					2.  AmtR HlYlIR BM3R1 YFP P_{Bad} P_{AmtR} P_{Tac} P_{HlYlIR} P_{Tet} P_{BM3R1}
					3.*  AmtR PhlF SrpR YFP P_{Bad} P_{AmtR} P_{Tac} P_{PhlF} P_{Tet} P_{SrpR}
					4.  AmtR PhlF HlYlIR YFP P_{Bad} P_{AmtR} P_{Tac} P_{PhlF} P_{Tet} P_{HlYlIR}
D-VASim Logic Analyzer	15	$\bar{A}\bar{B}\bar{C}+A\bar{B}\bar{C}+AB\bar{C}$	$\bar{C}(AB+\bar{B})$	$\overline{(\mathbf{C}+\overline{(\bar{\mathbf{B}}+\mathbf{A})})}$	1.  AmtR HlYlIR BM3R1 YFP P_{Tet} P_{AmtR} P_{Tac} P_{HlYlIR} P_{Bad} P_{BM3R1}
					2.  AmtR HlYlIR SrpR YFP P_{Tet} P_{AmtR} P_{Tac} P_{HlYlIR} P_{Bad} P_{SrpR}

Figure D.2: Experimental results of *Gene*Tech for the optimization, synthesis and technology mapping of 0xC4 circuit.



Circuit 0xC8					
Boolean Logic obtained from	**Optimization**			**Synthesis**	**Technology Mapping** AmtR HlYIIR SrpR BetI PhlF BM3R1 AmeR YFP
	t_E (ms)	Exp_{Init}	Exp_{Opt}		
Cello Paper	40	$\bar{A}\bar{B}C+A\bar{B}\bar{C}+\bar{A}\bar{B}\bar{C}$	$\bar{B}(A\bar{C}+\bar{A})$	$\overline{(\mathbf{B}+\overline{(\bar{\mathbf{A}}+\bar{\mathbf{C}})})}$	1. P_{Tac} AmtR P_{Bad} SrpR P_{SrpR} P_{AmtR} PhlF P_{PhlF} P_{Tet} HlYIIR P_{HlYIIR} YFP
					2. P_{Tac} SrpR P_{Bad} AmtR P_{AmtR} P_{SrpR} PhlF P_{PhlF} P_{Tet} HlYIIR P_{HlYIIR} YFP
					3. P_{Tac} PhlF P_{Bad} AmtR P_{AmtR} P_{PhlF} HlYIIR P_{HlYIIR} P_{Tet} BetI P_{BetI} YFP
					4. P_{Tac} PhlF P_{Bad} AmtR P_{AmtR} P_{PhlF} HlYIIR P_{HlYIIR} P_{Tet} BM3R1 P_{BM3R1} YFP
D-VASim Logic Analyzer	40	$\bar{A}B\bar{C}+A\bar{B}\bar{C}+\bar{A}\bar{B}\bar{C}$	$\bar{C}(A\bar{B}+\bar{A})$	$\overline{(\mathbf{C}+\overline{(\bar{\mathbf{A}}+\bar{\mathbf{B}})})}$	1. P_{Tac} PhlF P_{Tet} AmtR P_{AmtR} P_{PhlF} HlYIIR P_{HlYIIR} P_{Bad} BM3R1 P_{BM3R1} YFP
					2. P_{Tac} PhlF P_{Tet} AmtR P_{AmtR} P_{PhlF} HlYIIR P_{HlYIIR} P_{Bad} SrpR P_{SrpR} YFP

Figure D.3: Experimental results of *Gene***Tech** for the optimization, synthesis and technology mapping of 0xC8 circuit.

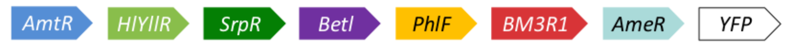
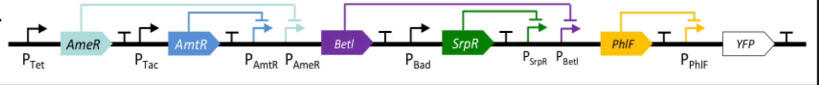
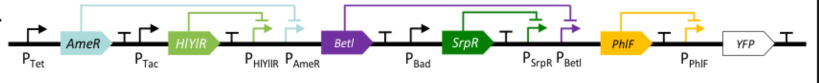
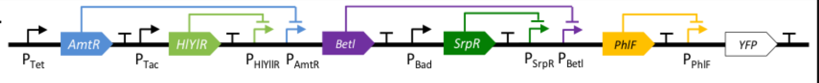
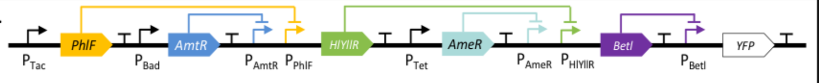
Circuit 0x0E					
Boolean Logic obtained from	Optimization			Synthesis	Technology Mapping 
	t_E (ms)	$\mathrm{Exp}_{\mathrm{Init}}$	$\mathrm{Exp}_{\mathrm{Opt}}$		
Cello Paper	30	$\bar{A}\bar{B}C+\bar{A}BC+A\bar{B}C$	$C(\bar{A}B+\bar{B})$	$\overline{(\bar{\mathbf{C}}+\overline{(\bar{\mathbf{B}}+\bar{\mathbf{A}})})}$	1. 
					2. 
					3. 
D-VASim Logic Analyzer	30	$\bar{A}B\bar{C}+\bar{A}BC+AB\bar{C}$	$B(\bar{A}C+\bar{C})$	$\overline{(\bar{\mathbf{B}}+\overline{(\bar{\mathbf{C}}+\bar{\mathbf{A}})})}$	1. 

Figure D.4: Experimental results of *Gene*Tech for the optimization, synthesis and technology mapping of 0x0E circuit.

Appendix E

D-VASim – *Quick Start Guide*

The quick start guide (QSG) of D-VASim latest version is attached as a complete PDF file below.

D-VASim

Dynamic Virtual Analyzer and Simulator

A quick start guide v1.2

October 13 2016

Department of Applied Mathematics and Computer Science
Section of Embedded Systems Engineering
Bio-Design Automation Group
Richard Petersens Plads, Lyngby 3200
Technical University of Denmark

http://bda.compute.dtu.dk

Contents

About D-VASim

D-VASim stands for Dynamic Virtual Analyzer and Simulator. It is developed to analyze and simulate the genetic logic circuit models developed in the Systems Biology Markup Language (SBML) [1]. Practically, D-VASim can be used to simulate any bio-model available in SBML L3v1 (Level 3 version 1) core; however, it is solely designed for the simulations of genetic logic circuits, where user can apply external signals on the model during runtime.

D-VASim is developed on graphical programming language platform called LabVIEW (Laboratory Virtual Instrument Engineering Workbench) [2]. Besides graphical programming, textual programming language, like JAVA, is also used to integrate JSBML [3] library in it.

In this short document, you will learn how to use D-VASim (v1.2) to perform virtual laboratory experiments. This requires you to have pre-synthesized bio-model, in the form of SBML file, you want to test. This SBML can be obtained by creating a bio-model using any other bio-tool and generating its SBML L3v1 format.

Stochastic simulation algorithm (SSA) has been implemented to perform stochastic simulations of SBML models. Furthermore, D-VASim is also capable of simulating the deterministic behavior of a bio model by solving ordinary differential equations.

Latest version of D-VASim can be downloaded from http://bda.compute.dtu.dk/downloads/. The sample SBML models, included in the download package, are developed at the University of Utah by Myer's group [4]. The video demo of D-VASim can also be seen at http://bda.compute.dtu.dk/user-manuals/, which demonstrates the analysis of upper threshold value only. In D-VASim v1.2, the ability to analyze the upper and lower threshold values is included.

Conventions

The following conventions appear in this document:

💡	This icon denotes a tip, which notifies you to advisory information.
⚠	This icon denotes an alert, which notifies you to important information.
bold	Bold text denotes items that you must select or click or enter the value in the software, such as open file option or running the simulation button or entering the value of simulation speed etc.
italic	Italic text denotes the name of a folder or a file path.
bold and italic	Bold and italic text denotes the name of a file.

1. Analyzing the SBML model using D-VASim

In this section, you will learn how to import and analyze the SBML model of a genetic circuit in D-VASim. In this document, we will use the SBML model of a genetic AND gate enclosed in the *Sample_SBML_Models* folder in the download package.

1.1. Launching D-VASim

The front interface of D-VASim, shown in Figure 1, will appear when you double-click on ***D-VASim.app*** (for MAC OS) or ***D-VASim.exe*** (for Windows OS).

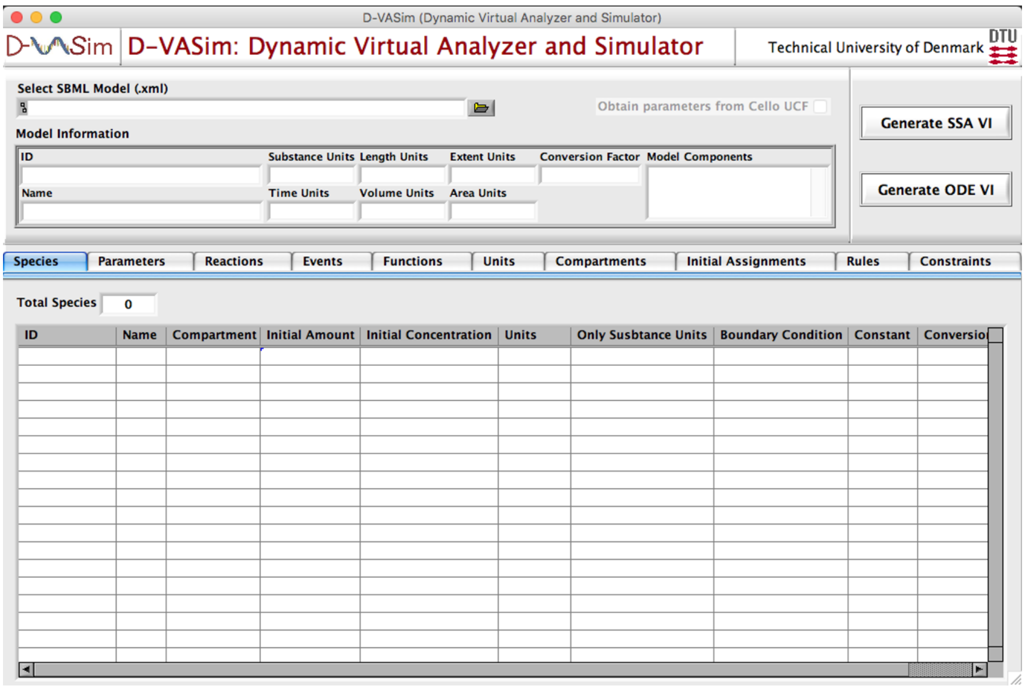
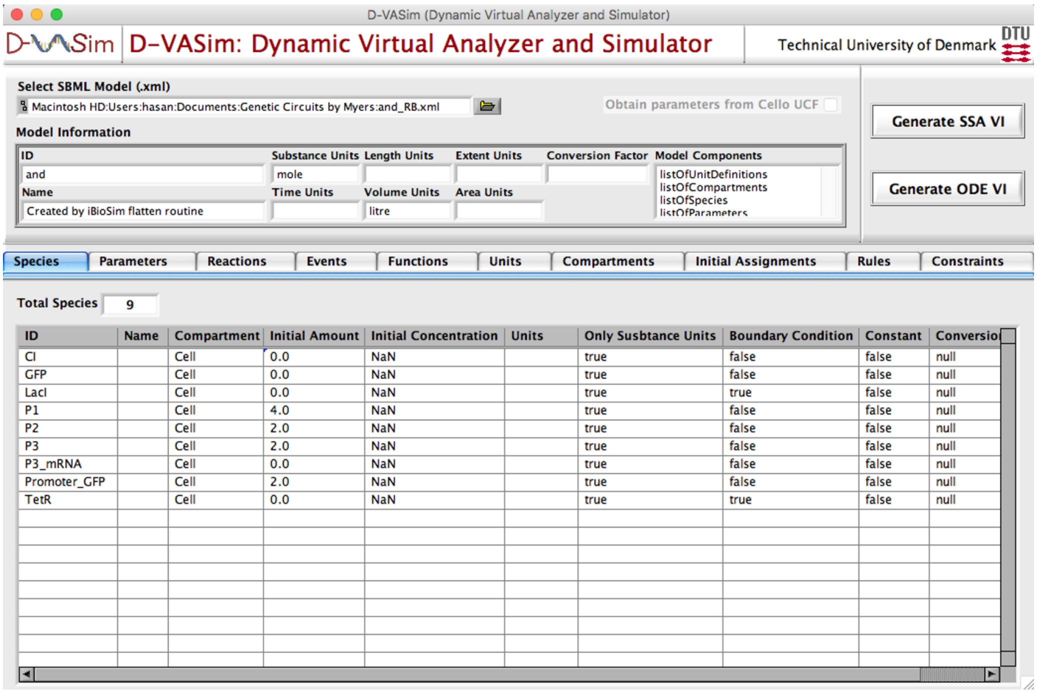


Figure 1. Front interface of D-VASim.

Complete the following steps to import the SBML model of a genetic AND gate circuit.

1. Click **open file** button on the option "Select SBML Model (.xml)".
2. Navigate to the file ***and_RB.xml*** placed under the application directory*../D-VASim/Sample_SBML_Models/*.

This will run the JSBML library and display the SBML components as shown in Figure 2.


D-VASim (Dynamic Virtual Analyzer and Simulator)
D-VASim
D-VASim: Dynamic Virtual Analyzer and Simulator
Technical University of Denmark
DTU
Generate SSA VI
Generate ODE VI
Obtain parameters from Cello UCF
Species
Parameters
Reactions
Events
Functions
Units
Compartments
Initial Assignments
Rules
Constraints

Select SBML Model (.xml)
Macintosh HD:Users:hasan:Documents:Genetic Circuits by Myers:and_RB.xml

Model Information

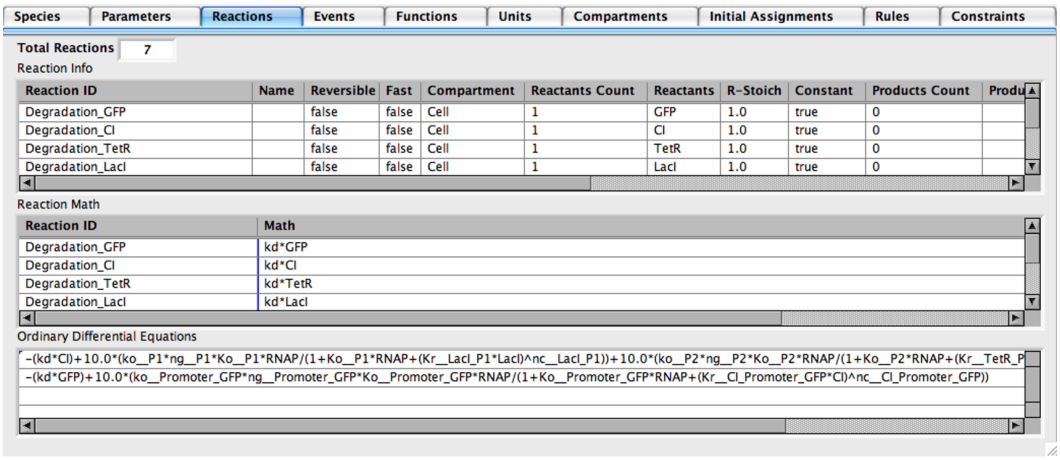
ID	Substance Units	Length Units	Extent Units	Conversion Factor	Model Components
and	mole				listOfUnitDefinitions listOfCompartments listOfSpecies listOfParameters
Name	**Time Units**	**Volume Units**	**Area Units**		
Created by iBioSim flatten routine		litre			

Total Species 9

ID	Name	Compartment	Initial Amount	Initial Concentration	Units	Only Susbtance Units	Boundary Condition	Constant	Conversio
CI		Cell	0.0	NaN		true	false	false	null
GFP		Cell	0.0	NaN		true	false	false	null
LacI		Cell	0.0	NaN		true	true	false	null
P1		Cell	4.0	NaN		true	false	false	null
P2		Cell	2.0	NaN		true	false	false	null
P3		Cell	2.0	NaN		true	false	false	null
P3_mRNA		Cell	0.0	NaN		true	false	false	null
Promoter_GFP		Cell	2.0	NaN		true	false	false	null
TetR		Cell	0.0	NaN		true	true	false	null

Figure 2. D-VASim showing the SBML components of a genetic AND gate model.

All the SBML components of a model can be analyzed here in a clean tabular form. For example, The **Specie** tab shown in Figure 2 depicts the total number of specie used in the genetic AND gate model along with their corresponding details. Similarly, the **Reactions** tab shown in Figure 3 contains the information related to the reaction kinetics of a model. D-VASim also generates the ordinary differential equations of a model, which can be seen in Figure 3. These equations are used to simulate the deterministic behavior of SBML model.



Total Reactions 7

Reaction Info

Reaction ID	Name	Reversible	Fast	Compartment	Reactants Count	Reactants	R-Stoich	Constant	Products Count	Produ
Degradation_GFP		false	false	Cell	1	GFP	1.0	true	0	
Degradation_CI		false	false	Cell	1	CI	1.0	true	0	
Degradation_TetR		false	false	Cell	1	TetR	1.0	true	0	
Degradation_LacI		false	false	Cell	1	LacI	1.0	true	0	

Reaction Math

Reaction ID	Math
Degradation_GFP	kd*GFP
Degradation_CI	kd*CI
Degradation_TetR	kd*TetR
Degradation_LacI	kd*LacI

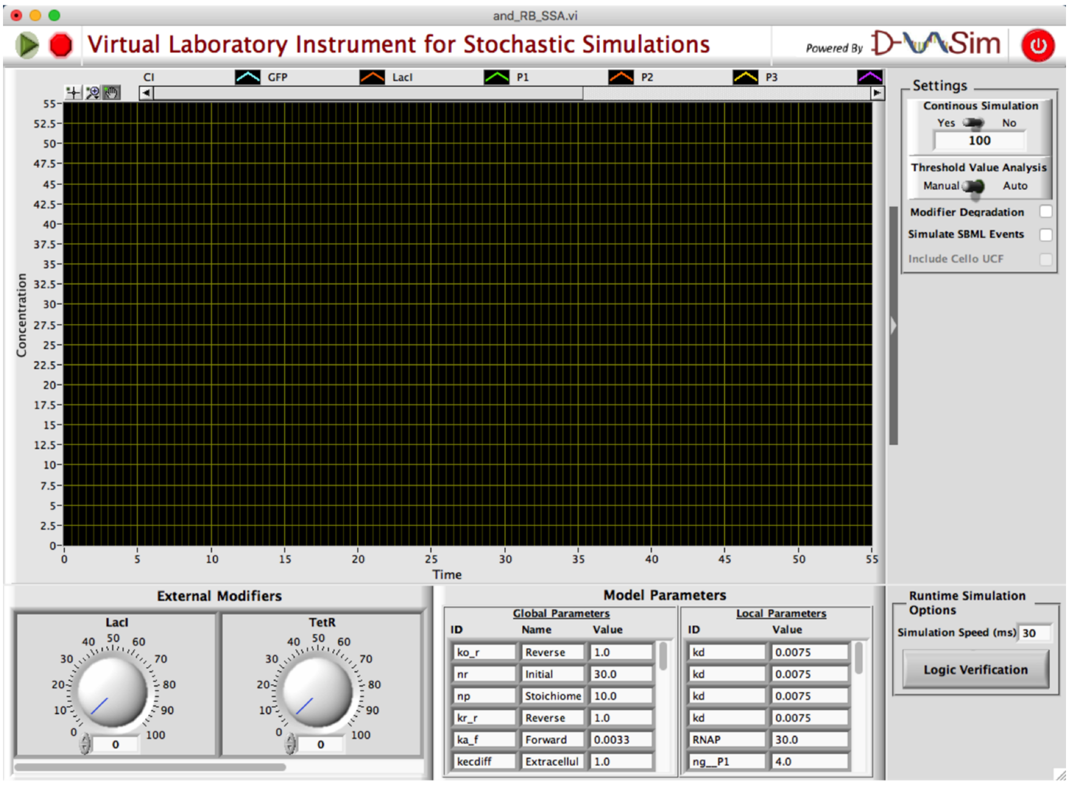
Ordinary Differential Equations

```
-(kd*CI)+10.0*(ko__P1*ng__P1*Ko__P1*RNAP/(1+Ko__P1*RNAP+(Kr__LacI_P1*LacI)^nc__LacI_P1))+10.0*(ko__P2*ng__P2*Ko__P2*RNAP/(1+Ko__P2*RNAP+(Kr__TetR_P
-(kd*GFP)+10.0*(ko__Promoter_GFP*ng__Promoter_GFP*Ko__Promoter_GFP*RNAP/(1+Ko__Promoter_GFP*RNAP+(Kr__CI_Promoter_GFP*CI)^nc__CI_Promoter_GFP))
```

Figure 3. Reactions tab showing the reaction kinetics of a genetic AND gate model.

1.2. Generating a Virtual Laboratory Instrument

Once the components of SBML model are analyzed, a virtual environment for the interactive simulation can be generated simply by clicking on any of the two options, **Generate SSA VI** or **Generate ODE VI** shown at the top right corner of a tool. Now click **Generate SSA VI** button to generate a virtual instrument (VI) for stochastic simulations of a genetic AND gate model. This will bring up a virtual instrument shown in Figure 4.



Model Parameters

Global Parameters

ID	Name	Value
ko_r	Reverse	1.0
nr	Initial	30.0
np	Stoichiome	10.0
kr_r	Reverse	1.0
ka_f	Forward	0.0033
kecdiff	Extracellul	1.0

Local Parameters

ID	Value
kd	0.0075
kd	0.0075
kd	0.0075
kd	0.0075
RNAP	30.0
ng__P1	4.0

Figure 4. Virtual Instrument specifically for a genetic AND gate model.

Any instrument for stochastic simulation generated by D-VASim for the first time does neither reveal the names of graph legend nor the modifier controls, but only the parameters information. Pressing the **RUN** button (▶ shown at the top left corner) starts the simulation and makes the graph legends and modifiers visible.

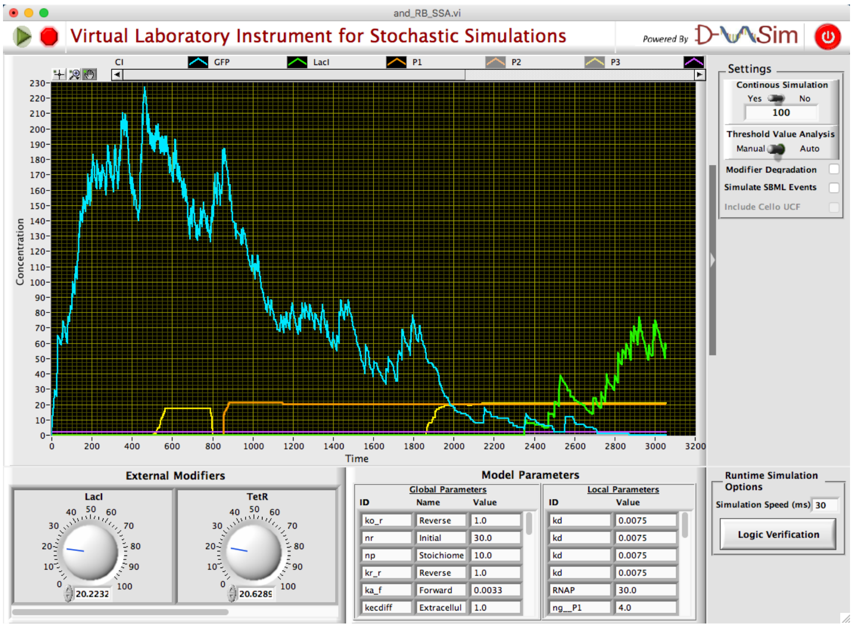
2. Virtual Experimentation for Stochastic Simulations

The virtual instrument (shown in Figure 4) looks similar to a physical instrument, which can be used to interact with the model by tuning the input concentrations with the control knobs and observing the effects graphically.

Once the simulation is started after pressing the ▶ button, user can increase or decrease the concentration of input (or external modifier) specie and observe its effects on the model during run time. This runtime interaction with the model gives user an insight of being in the lab performing wet-lab experiments.

Figure 5 shows the screenshot of the stochastic simulation of a genetic AND gate taken randomly after ~3050 time units has lapsed. This figure shows that unlike SBML events, which are predefined in the SBML file, user can interact with the model and change the concentration of input specie to any level and at any instant of time. In case if the concentration of input specie are required to be triggered to any specific level instantly, the numeric displays located beneath the control knobs can be used. As same as the user interact with the model by varying the input concentration using control knobs, the parameters can also be varied and their effects can be observed graphically during runtime.

When a **STOP** button (⬣ located besides the run button at the top left corner) is pressed, screenshot of a graphical window is captured and saved inside *Sim_Results* folder located in the default application directory.



Settings
Continous Simulation: Yes / No — 100
Threshold Value Analysis: Manual / Auto
Modifier Degradation ☐
Simulate SBML Events ☐
Include Cello UCF ☐

LacI: 20.2232
TetR: 20.6289

Model Parameters

Global Parameters

ID	Name	Value
ko_r	Reverse	1.0
nr	Initial	30.0
np	Stoichiome	10.0
kr_r	Reverse	1.0
ka_f	Forward	0.0033
kecdiff	Extracellul	1.0

Local Parameters

ID	Value
kd	0.0075
kd	0.0075
kd	0.0075
kd	0.0075
RNAP	30.0
ng__P1	4.0

Runtime Simulation Options
Simulation Speed (ms) 30
Logic Verification

Figure 5. Screen capture of the stochastic simulation of a genetic AND gate.

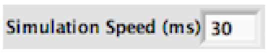
Each image is saved by the following naming convention; “<model-name>_SSA_Screenshot.bmp”, where <model-name> is the name of a model. For example, the screenshot of a graphical window of a genetic AND gate model is saved as ***and_RB_SSA_Screenshot.bmp***. The simulation traces of the entire session are also saved in the *Sim_Results* folder with the convention “<model-name>_SSADatalog.xls”. Pressing the **RUN** button again starts the simulation from the beginning.

Alert – When the **STOP** button is pressed, the captured screenshot and the data log file overwrites any previously saved files of the same model.

Tip – The properties of graphical plots, including colors, can be changed by a left mouse click on the color of graph legend and selecting the desired menu.

2.1. Runtime Simulation Options

There are two runtime simulation options, available in D-VASim v1.2, shown at the bottom right corner of a VI in Figure 5. These two options are described below:


Simulation Speed (ms) 30

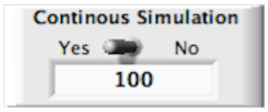
1. **Simulation Speed**: This option allows you to speed up or slow down the simulation by entering the numerical value in milliseconds.
2. **Logic Verification**: Obtain the Boolean logic exist in the model. Clicking this button prompts the user to input the threshold values of input and output specie, as shown in Figure 8. When automatic threshold value analysis is performed, this box is auto-populated with the estimated threshold value. This option is disabled when the simulation is not running.

Alert – Make sure to apply all the possible input combinations each after the correct propagation delays. The results of boolean logic analysis will otherwise be incorrect.

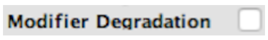
Alert – Though, D-VASim v1.2 supports the logic verification of n-input genetic logic circuits, but the logic verification algorithm is tested for up to 3-input genetic circuits only. It is because the models for more complex circuits are not available.

2.2. Simulation Settings

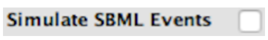
The simulation settings are shown in the right sidebar, which contains the following five simulation options.


Continous Simulation
Yes No
100

1. **Continuous Simulation:** User may specify here if they want to perform continuous simulation or for a specific interval of time. If switched to “No”, the box below can be used to specify the simulation time for which user wants to run the simulation. The default value of this switch is “Yes” meaning that the continuous simulation is always performed unless specified.

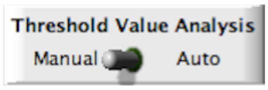

Modifier Degradation ☐

2. **Modifier Degradation**: When checked, perform more realistic simulation by allowing the input concentrations to degrade over time.

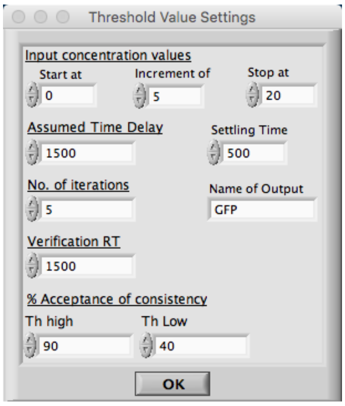

Simulate SBML Events ☐

3. **Simulate SBML Event**: When checked, simulate the events described in SBML file.

Tip – The triggering time and the assignment value of SBML events can be changed during run time until the time of event occurs. For example, if the triggering time of event is 1000 time units, the trigger time and the assignment value can be changed before 1000 time units has lapsed.


Threshold Value Analysis
Manual Auto

4. **Threshold Value Analysis**:
   Manual – In this mode, VI assume that the threshold value of input specie and the propagation delays are already known.
   Auto – When this mode is selected, VI estimates the threshold value based on the setting tab shown in Figure 6 below.
   - **Start at** – Specify the initial concentration of input specie from which the analysis is required to be started. It is recommended to keep this value 0 for correct results.
   - **Increment of** – This control allow you to set up the concentration value, which the tool increments during each iteration for checking threshold concentration.
     For example: If you set
     Start at: 0
     Increment of: 5
     Stop at: 15
     Tool will check the threshold concentration for 4 different levels of concentration. i.e. 0 --> 5 --> 10 --> 15, for all possible input combinations.


Threshold Value Settings

Input concentration values
Start at: 0
Increment of: 5
Stop at: 20

Assumed Time Delay: 1500
Settling Time: 500

No. of iterations: 5
Name of Output: GFP

Verification RT: 1500

% Acceptance of consistency
Th high: 90
Th Low: 40

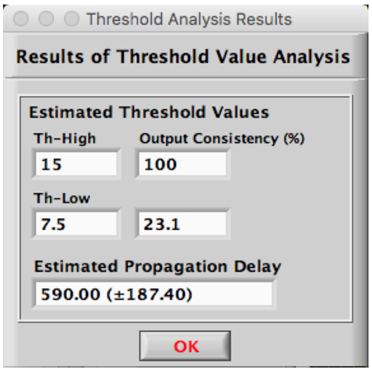
Figure 6. Settings dialogue for automatic threshold value and propagation delay analyses.

- **Stop at** – Specify the concentration of inputs at which the threshold analysis should be stopped.
- **Assumed Time Delay** – Specify initially assumed time delay.

- **No. of iterations** – For how many times a user wants to verify if the estimated threshold value is consistent.
- **Verification RT** – For how long a user wants to verify the estimated threshold value for each no. of iterations mentioned above.
- **Settling Time** – It is also an assumed value, which specifies the time required by a model to get stable. During verification process, D-VASim triggers the concentration of inputs to the estimated threshold level once this time value has elapsed. Thus the verification starts once the input is triggered to its estimated threshold value and therefore it should run for enough amount of time to measure the propagation delay as well as the correctness of an estimated threshold value. Depending on the complexity of input-output stage, the value of "Verification RT" is suggested to be at least 3-4 times higher than the value of *Settling Time.*
- **Name of Output** – D-VASim now allows user to perform the threshold value and propagation delay analysis either for an entire circuit or for the intermediate circuit components. In this field, user can specify the name of output specie. The timing analysis is performed between the input specie, selected as control knobs (see section 2.2) and the output specie specified in this field.
- **% Acceptance of consistency** – Specify the minimum percentage of threshold value consistency a user wants.
  Th High: If the chosen value is 90 for the AND gate genetic circuit, the tool give the results only when the average output is found to be 90% greater than the estimated threshold value.
  Th Low: If the chosen value is 40 for the AND gate genetic circuit, the tool estimates the threshold value to be lower level when the size of an average output data above the estimated threshold value is found to be less than 40%.

Alert – It is an indication of wrong parameter selection if either tool takes too long to estimate the results or produce incorrect results. In this case user should consider increasing the value of "Verification RT" further higher (beyond 3-4 times) than the value of "Settling Time". Though assuming large values of these parameters, including "Assumed Time Delay", may increase the estimation time, but gives the better estimation.

When the tool finish estimating the threshold value with the desired minimum percentage of consistency, the results are displayed in the dialogue box as shown in Figure 7. The value of estimated propagation delay is the average value over all "No. of iterations" specified in the settings dialogue (Figure 6). The value enclosed in braces after the average estimated propagation delay specifies the standard deviation. In this figure, the output consistency for upper threshold level is found to be 100 % and that of lower threshold level is 23.1%.


Threshold Analysis Results

Results of Threshold Value Analysis

Estimated Threshold Values

Th-High	Output Consistency (%)
15	100

Th-Low	
7.5	23.1

Estimated Propagation Delay

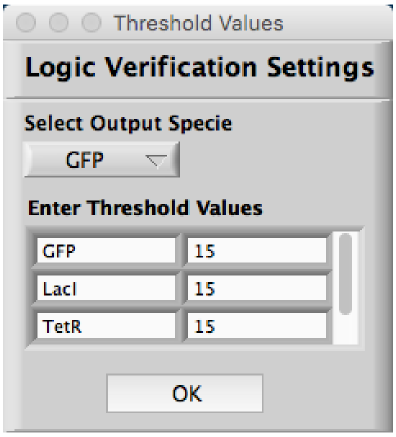
590.00 (±187.40)

Figure 7. Dialogue box indicating the estimated results of threshold value and propagation delay analysis.

The results are stored in the current application directory with the following naming convention; "<model-name>_SSA-TP-Analysis.txt", where <model-name> is the name of a model. For the case of genetic AND gate, the results will be saved as ***and_RB_SSA-TP-Analysis.txt***.

User can now perform the runtime interactive simulations and perform logic analysis, by pressing the *Logic Verification* button shown in section 2.1. It will pop up a window (Figure 8) where user can specify the upper threshold values of a circuit. As mentioned before, if the automatic threshold value analysis is performed, the list of threshold values is auto populated as shown in Figure 8. When all the threshold values are entered, a new window (Figure 9) pops up indicating the logic function, in terms of a Boolean expression, extracted from the genetic logic circuit model. This window also contains the entire simulation data as analog data points and their corresponding digital data according to the defined threshold values. If all the input combinations are applied correctly during the simulation, a message in green text (Figure 9 (a)) will appear showing the percentage fitness of the boolean expression in the acquired simulation data. If all the possible input combinations are not applied, a boolean expression may appear with the message in red text (Figure 9 (b)) indicating that all input combinations are not applied correctly. Boolean expression box may also appear blank when all the input combinations are not applied correctly. A correct set of input combinations is the one in which all the possible input combinations are applied each after the minimum propagation delay.

D-VASim also indicates the time, in seconds, required to estimate the boolean logic from the analog simulation data. This estimation time depends on the size of the data as well as on the number of inputs of a circuit.



Threshold Values

Logic Verification Settings

Select Output Specie

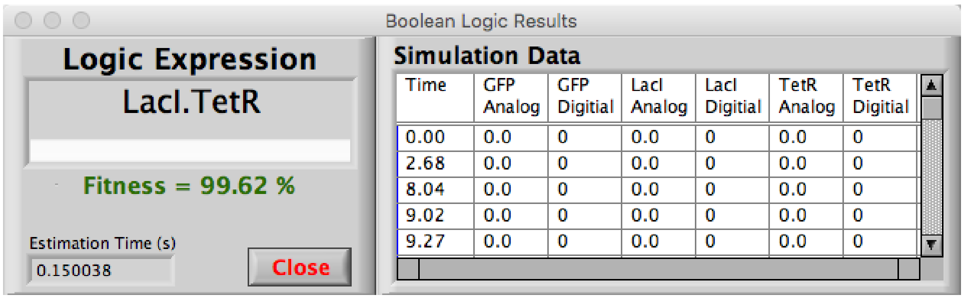
GFP

Enter Threshold Values

GFP	15
LacI	15
TetR	15

OK

Figure 8. Dialogue box to set up the threshold values for Boolean logic analysis.



Boolean Logic Results

Logic Expression

LacI.TetR

Fitness = 99.62 %

Estimation Time (s)

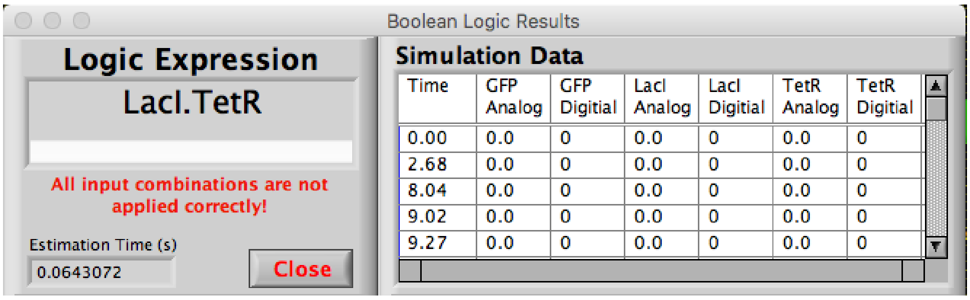
0.150038

Close

Simulation Data

Time	GFP Analog	GFP Digitial	LacI Analog	LacI Digitial	TetR Analog	TetR Digitial
0.00	0.0	0	0.0	0	0.0	0
2.68	0.0	0	0.0	0	0.0	0
8.04	0.0	0	0.0	0	0.0	0
9.02	0.0	0	0.0	0	0.0	0
9.27	0.0	0	0.0	0	0.0	0

(a)



Boolean Logic Results

Logic Expression

LacI.TetR

All input combinations are not applied correctly!

Estimation Time (s)

0.0643072

Close

Simulation Data

Time	GFP Analog	GFP Digitial	LacI Analog	LacI Digitial	TetR Analog	TetR Digitial
0.00	0.0	0	0.0	0	0.0	0
2.68	0.0	0	0.0	0	0.0	0
8.04	0.0	0	0.0	0	0.0	0
9.02	0.0	0	0.0	0	0.0	0
9.27	0.0	0	0.0	0	0.0	0

(b)

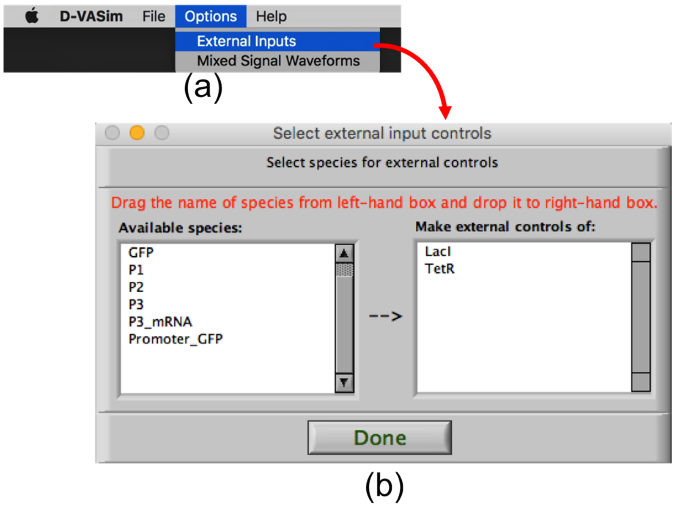
Figure 9. Boolean logic extracted from the analog simulation data along with its corresponding digital data.
(a) Percentage fitness appears when all the input combinations are properly applied.
(b) Error message appears when all the input combinations are not applied correctly before trying to estimate the boolean logic of a circuit.

2.3. Additional features of Stochastic Simulation in D-VASim

In D-VASim v1.1, two new features have been introduced, which allows user to:

1) Create control knobs of specie manually. This not only helps user to vary the concentration of selected specie during runtime, but also perform the timing analysis between these specie and the specified output.
2) View the mixed signal waveforms (digital and analog) for better timing analysis.

The control knobs for the input specie can be created by dropping down the **Options** menu and selecting **External Inputs** as shown in Figure 10(a). When **External Inputs** is selected, a pop-up window is appeared as shown in Figure 10(b). The user can drag the specie name from left-hand box to right-hand box in order to create its control knob. In Figure 10(b), LacI and TetR specie are selected whose control knobs are created as shown in Figure 4. These control knobs can only be created when the simulation is not running.



Select external input controls

Select species for external controls

Drag the name of species from left-hand box and drop it to right-hand box.

Available species:
GFP
P1
P2
P3
P3_mRNA
Promoter_GFP

Make external controls of:
LacI
TetR

Figure 10. Process of creating control knobs for specie. (a) Select **External Inputs** from **Options** dropdown menu. (b) Drag the name of specie from left-hand to right-hand box to create its control knob.

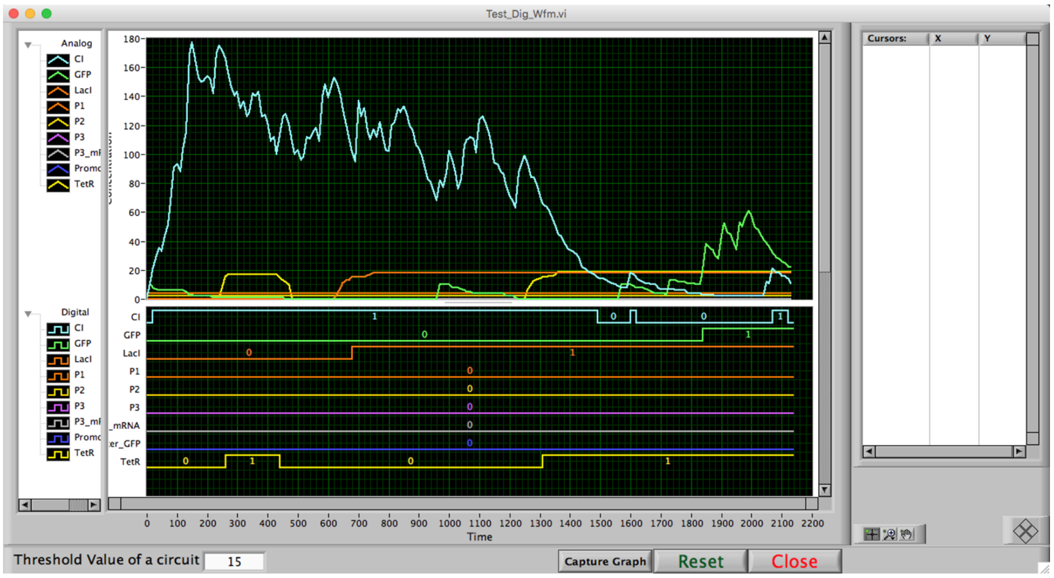


Figure 11. Mixed signal waveforms of genetic AND gate.

Similarly, after performing interactive stochastic simulation of a model in the D-VASim virtual laboratory environment, the mixed signal waveforms for timing analysis can be viewed by selecting **Mixed Signal Waveforms** option from the **Options** dropdown menu. Figure 11 shows the window of mixed signal waveforms generated for the simulation traces of genetic AND gate shown in Figure 5. This figure shows the analog and digital traces, in parallel, for genetic AND gate. The digital curves are generated based on the estimated threshold value shown in Figure 7. This value is automatically extracted if the auto threshold value analysis is performed, otherwise user can enter the threshold value and **Reset** the curves. **Capture Graph** button takes the screen-shot of waveforms and save it inside *Sim_Results* folder under the default application directory. Each image is saved by the file name ***MixedSignalWaveforms.bmp***. The blank vertical box shown at the right-hand side in Figure 11 allows user to create cursors (see section 4 for more details about how to create cursors).

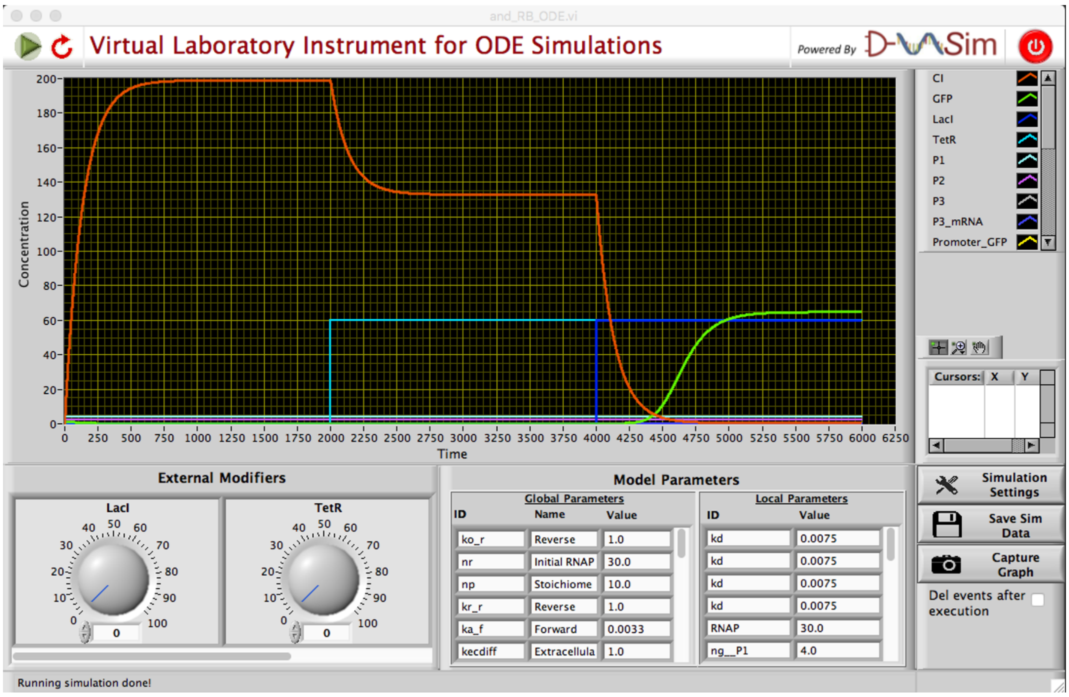
Alert – When the **Capture Graph** button is pressed, the captured screenshot overwrites any previously saved files of the same model.

Tip – The properties of graphical plots, including colors, can be changed by a left mouse click on the color of graph legend and selecting the desired menu.

3. Virtual Experimentation for ODE Simulations

Similar to generating virtual instrument for stochastic simulations, the virtual instrument for deterministic simulations can also be generated by clicking on **Generate ODE VI** button depicting in Figure 2.

When the ▶ button is pressed, the deterministic simulation results of a genetic AND gate for 6000 time units are appeared as shown in Figure 12. As same as in the case of stochastic simulation, user can change the concentration of external input specie using control knobs for deterministic simulation also. However, every single change in the input concentration or parameters results in the execution of selected simulation algorithm for the defined interval of time and plots the new set of curves.



Model Parameters

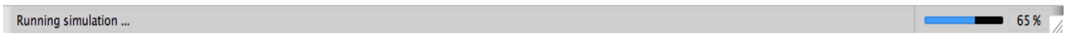
Global Parameters ID	Name	Value
ko_r	Reverse	1.0
nr	Initial RNAP	30.0
np	Stoichiome	10.0
kr_r	Reverse	1.0
ka_f	Forward	0.0033
kecdiff	Extracellula	1.0

Local Parameters ID	Value
kd	0.0075
kd	0.0075
kd	0.0075
kd	0.0075
RNAP	30.0
ng__P1	4.0

Running simulation done!

Figure 12. Screen capture of the deterministic simulation of a genetic AND gate.

The horizontal *status bar* is shown at the bottom in Figure 12. It shows the status of the actions being performed in this VI for ODE simulations. For example, pressing the run button shows the status of simulation as shown in Figure 13 below. The progress bar shown at right-hand side depicts the percentage of results currently being obtained. The plot is visible once the progress bar reaches to 100 %.



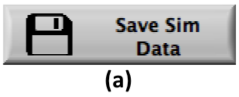
Running simulation ...

65 %

Figure 13. Screen capture of the status bar showing that the VI is running simulation. The progress bar shown at right-hand side indicates that the 65% of simulation data is obtained.

When a **RESET** button (↻) button is pressed, screenshot of a graph window and the simulation data is captured and saved under *Sim_Results* folder located in the default application directory. Each image is saved by the following naming convention; "<model-name>_ODE_Screenshot.bmp", where <model-name> is the name of a model. For example, the screenshot of a graphical window of a genetic AND gate model is saved as ***and_RB_ODE_Screenshot.bmp***. The simulation traces of the entire session are saved with the naming convention; "<model-name>_ODEDatalog.xls". Pressing the **RUN** button again starts the simulation from the beginning.

Alternatively, the simulation data and the graphical plots can be stored by pressing the buttons shown in Figure 14(a) and 14(b) respectively.



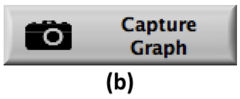


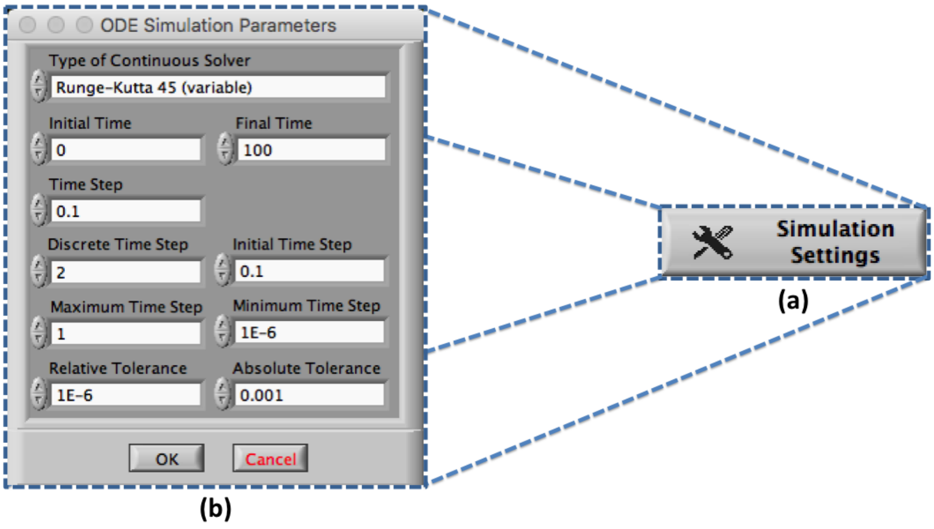
Figure 14. Available options in ODE VI to (a) Save simulation data. (b) Capture graph window.

Alert – When the **RESET** or **Save Sim Data** or **Capture Graph** button is pressed, the captured screenshot and the data log file overwrites any previously saved files of the same model.

Tip – The properties of graphical plots, including colors, can be changed by a left mouse click on the color of graph legend and selecting the desired menu.

3.1. Simulation Settings

The **Simulation Settings** button (shown in Figure 15(a)) helps user to set the desired simulation parameters. A settings window, shown in Figure 15(b), is popped-up when user clicks on the **Simulation Settings** button.



ODE Simulation Parameters

Type of Continuous Solver: Runge-Kutta 45 (variable)

Initial Time: 0

Final Time: 100

Time Step: 0.1

Discrete Time Step: 2

Initial Time Step: 0.1

Maximum Time Step: 1

Minimum Time Step: 1E-6

Relative Tolerance: 1E-6

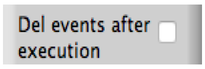
Absolute Tolerance: 0.001

Figure 15. ODE simulation properties (a) Simulation settings option. (b) Window of simulation settings.

D-VASim VIs for ODE simulation supports the following ten continuous solvers:

1. Runge-Kutta 1 (Euler)
2. Runge-Kutta 2
3. Runge-Kutta 3
4. Runge-Kutta 4
5. Runge-Kutta 23 (variable)
6. Runge-Kutta 45 (variable)
7. BDF (variable)
8. Adams-Moulton (variable)
9. Rosenbrock (variable)
10. Discreet States Only

The parameters shown in Figure 15(b) are set as the default parameters. There is one other option available at the right bottom of ODE VI, which is described below



Del events after ☐
execution

Del events after execution: When marked checked, the events are deleted internally after executing once. This helps increasing the simulation speed by avoid checking the executed events repeatedly.

Tip – If the desired simulation results are not obtained, try simulating the model again by checking or unchecking the option "Del events after execution".

4. General Software Options and Tips

Following are some general tips and options for both stochastic and ODE virtual instruments generated by D-VASim.

Tip – Hover mouse over simulation options to pop up the tip strip with a short description.

Tip – Pressing keys CMD+SHIFT+H (on MAC OS) and CTRL+H (on Windows OS) open the context help.

Tip – User can right click on the graph window to get the runtime shortcut menu shown in Figure 16 below.

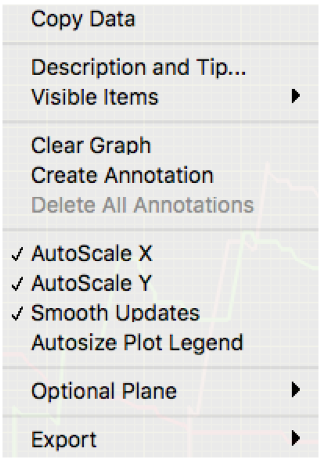
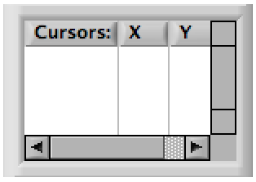
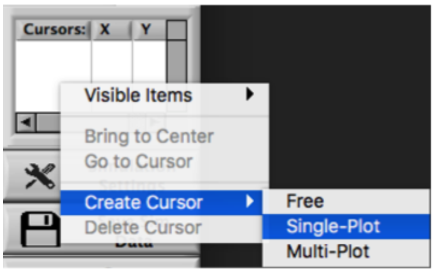


Figure 16. Runtime shortcut menu of graph window.

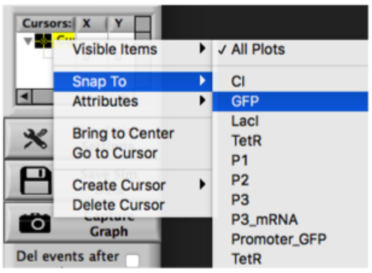
Graph Palette: This option can be used to zoom-in, zoom-out, expand graph etc.



Cursor Legend: With the help of this box, cursors can be set to track the values of desired specie. For example, single-plot cursor can be set to monitor GFP specie by **right clicking** in the Cursors window and then selecting **Single-Plot** from **Create Cursors** option, as depicted in Figure 17(a). Once the cursor is created, it can be snap to GFP specie by **right clicking** on the created cursor and then go to **Snap To** option and select **GFP**. These steps are shown in Figure 17(b).



(a)



(b)

Figure 17. Steps of creating cursors on specie.

Finally, similar to any physical instrument, virtual instruments powered by D-VASim also have emergency shut down button (⏻) shown at the top right corner of a virtual instrument. This button can be used to stop and shut down the instrument instantly.

References

[1]. Systems Biology Markup Language, http://sbml.org/Main_Page.

[2]. National Instruments LabVIEW, http://www.ni.com/labview/.

[3]. Dräger A, Rodriguez N, Dumousseau M, Dörr A, Wrzodek C, Le Novère N, Zell A, and Hucka M. **JSBML: a flexible Java library for working with SBML**. *Bioinformatics* (2011), 27(15): 2167–2168.

[4]. Chris J. Myers. Engineering Genetic Circuits. Chapman & Hall/CRC Press, July 2009.