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 toward 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. To name a few, 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], or the deadly pandemic virus, COVID-19.

### 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 Sect. 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.

SBML allows 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 Sect. 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 crosstalk with the neighboring circuit components. Also, the electronic logic gates are composed of transistors which have well defined and uniform threshold voltage levels that categorize 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 remain 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 Scope of This Book

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 book. 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 Sect. 2.3 for more details). For example,
events (shown as green-boxes in Fig. 2.6b) 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 run-time, 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 allow 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 assist 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 behavior 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 Book Organization

This book 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 also describes the standards in more detail and also give information about existing GDA tools.

**Chapter 3** gives an overview of Virtual Experimentation using DVASim. It briefly describes the whole simulation flow with the help of an example circuit model. The details of each subsequent step in this flow are discussed in separate chapters.

In **Chap. 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 **Chap. 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.

**Chapter 7** introduces the audience to a graphical (G) programming language platform. After completing this chapter, you will have enough knowledge and confidence to start developing programs using G language.

**Chapter 8** is a hands-on project which teaches you how to implement a very simple form of stochastic simulation algorithm using graphical programming language. The chapter is followed by a challenge to enhance the development skills further.

**Chapter 9** is another hands-on project which gives you a brief introduction of SBML and teaches you how to develop an SBML parser using graphical programming language. This chapter also contains a challenge to further improve the development skills.

Figure 1.1 shows a very high-level diagram which describes how the methods and tools discussed in this book can be used. Having the SBML model of a genetic circuit in DVASim, 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, DVASim 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. GeneTech generates the output in the form of standard SBOL
Fig. 1.1  The abstract diagram showing how the methods and tools discussed in this book can be used.

Data files as well as pictorial representation. The dotted line between GeneTech and the DVASim 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 SBOL models of genetic circuit can be converted into SBML form using any SBOL-to-SBML conversion tool (like iBioSim [15]) and then can be analyzed back in DVASim again.

1.6 Resources

DVASim latest released version and the corresponding quick start guide (QSG) are available to download from http://bda.compute.dtu.dk/downloads/d-vasim/.

The latest released version of GeneTech and the QSG can be downloaded from http://bda.compute.dtu.dk/downloads/genetech/.

Both DVASim and GeneTech are open source tools and their source codes can be accessed at the following links:

DVASim: https://github.com/hasanbaig/D-VASim.
GeneTech: https://github.com/hasanbaig/GeneTech.

Section III resources are provided as a supplementary material with this book. Solutions for all exercises and project challenges are available separately for instructors.

1.7 Conventions Used in This Book

The following conventions appear in this book:
This icon denotes a tip or hint or an idea, which notifies you to advisory information or further clarification.

This icon denotes an alert, which notifies you to important information.

**Bold**

Bold text is used to indicate the steps to follow; option selection; built-in LabVIEW function/structure, procedure names, and definitions.

**Bold Italic**

Bold italic text is used to denote the file names and VI names.

**Italic**

Italic text is used to emphasize something or to denote the names of program variables.

References

1. A. Arkin, Setting the standard in synthetic biology. Nat. Biotechnol. **26**(7), 771–774 (2008)
2. J.J. Collins, T.S. Gardner, C.R. Cantor, Construction of a genetic toggle switch in Escherichia coli. Nature **403**(6767), 339–342 (2000)
3. R.W. Basu, S. Basu, The device physics of cellular logic gates, in *NSC-1: The First Workshop of Non-Silicon Computing*, vol. 158 (2002), pp. 39–41
4. J.C. Anderson, E.J. Clarke, A.P. Arkin, C.A. Voigt, Environmentally controlled invasion of cancer cells by engineered bacteria. J. Mol. Biol. **355**(4), 619–627 (2006)
5. S. Atsumi, J.C. Liao, Metabolic engineering for advanced biofuels production from Escherichia coli. Curr. Opin. Biotechnol. **19**(5), 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. Euchas, T.S. Ham, J. Kirby, M.C.Y. Chang, S.T. Withers, Y. Shiba, R. Sarpong, J.D. Keasling, Production of the antimalarial drug precursor artemisinic acid in engineered yeast. Nature **440**(7086), 940–943 (2006)
7. M.A. Marchisio, J. Stelling, Computational design tools for synthetic biology. Curr. Opin. Biotechnol. **20**(4), 479–485 (2009)
8. H. de Jong, Modeling and simulation of genetic regulatory systems: a literature review. J. Comput. Biol. **9**(1), 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 Novére, 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, J. Wang, The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics **19**(4), 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, W. Wipat, H. Sauro, Synthetic biology open language (SBOL) Version 2.0.0. J. Integr. Bioinform. **12**(2), 272 (2015)
11. H. McAdams, L. Shapiro, Circuit simulation of genetic networks. Science **269**(5224), 650–656 (1995)
12. D. Bernardi, J.T. Dejong, B.M. Montoya, B.C. Martinez, Bio-bricks: biologically cemented sandstone bricks. Constr. Build. Mater. **55**, 462–469 (2014)
13. **SBML Software Matrix** (2010). [http://sbml.org/SBML_Software_Guide/SBML_Software_Matrix](http://sbml.org/SBML_Software_Guide/SBML_Software_Matrix)
14. A.A.K. Nielsen, B.S. Der, J. Shin, P. Vaidyanathan, V. Paralanov, E.A. Strychalski, D. Ross, D. Densmore, C.A. Voigt, Genetic circuit design automation. Science **352**(6281), aacl7341–aac17341 (2016)
15. C.J. Myers, N. Barker, K. Jones, H. Kuwahara, C. Madsen, N.P.D. Nguyen, iBioSim: a tool for the analysis and design of genetic circuits. Bioinformatics **25**(21), 2848–2849 (2009)