Decoding Genetic Circuit Failures:
Analyzing Static and Dynamic Failures in Genetic Circuitry

by

L. S. M. Buecherl

M.Sc., University of Colorado Boulder, 2022

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Biomedical Engineering

2024

Committee Members:

Chris Myers, Chair

Mirela Alistar, Member

Brian DeDecker, Member

Fabio Somenzi, Member

Zhen Zhang, Member

Buecherl, L. S. M. (Ph.D., Biomedical Engineering)

Decoding Genetic Circuit Failures:

Analyzing Static and Dynamic Failures in Genetic Circuitry

Thesis directed by Prof. Chris J. Myers

Synthetic biology resides at the nexus of engineering and biology, employing diverse approaches to engineer biological systems. These systems can be as simple as DNA sequences, biochemical reactions, or more abstracted through control theory or digital logic, among other ways. Similar to other engineering disciplines, for real-world applications, the designed systems must exhibit robustness and adaptability to environmental changes beyond controlled laboratory settings. This dissertation focuses on genetic constructs viewed specifically as digital logic genetic circuits, examining their implementation and failure behavior. It aims to elucidate and analyze various failure modes and proposes analytical methods to enhance genetic circuit robustness. This work defines genetic circuit failure, where deviations from expected output are deemed as unexpected and faulty. Such deviations may stem from failures at the cellular level or from flaws in the circuit's logic implementation or Boolean function. Subsequently, this dissertation develops computational methods to predict circuit behavior, employing diverse analysis techniques such as ordinary differential equation analysis, stochastic simulation algorithms, and stochastic model verification. These methodologies enable the prediction of the likelihood of failure occurrence. Furthermore, this dissertation compares different computational modeling techniques to assess the effort required for genetic circuit analysis. Finally, experimental validation is provided for a predicted circuit failure, demonstrating the practical application of the proposed methodologies.

Dedication

Dedicated to *Europe*, whatever that is.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to the following individuals and groups whose support and contributions have been instrumental in the completion of this dissertation. My appreciation goes to my advisor, Chris J. Myers, for his invaluable guidance, mentorship, and expertise throughout this journey. While I consider myself passionate you were always able to redirect myself into a productive direction. Your insights and encouragement have been shaping the direction of my research. Of course, I am also immensely grateful to the esteemed members of my dissertation committee: Mirela Alistar, Brian DeDecker, Fabio Somenzi, and Zhen Zhang. Their constructive feedback and suggestions have significantly enhanced the quality of this work.

Obviously, I am deeply indebted to the members of our Genetic Logic Lab, the Biocompute Lab lead by Thomas Gorowochski, and the FLUENT Verification Project, whose collaborative, intellectual, and personal environment has allowed me to innovate and grow. Specifically, Pedro Fontanarrosa has been a close collaborator on genetic circuit analysis. Special acknowledgment goes to Shivang Joshi, whose weeks of mentoring made it possible for me to proficiently use the "grown-up," good pipettes. I also want to give a shout-out to Jazz Ghataora for teaching me the importance of not disinfecting a bottle of ethanol in a bunsen burner (it's called *common sense*). Furthermore, I want to acknowledge Mohammad Ahmadi from the University of South Florida for his support throughout this work.

I extend my gratitude to the developers of the Synthetic Biology Open Language, including but not limited to Georgie Hau Sørensen and Gonzalo Vidal, as well as the developers of other essential tools that have laid the foundation for my research. Your dedication to advancing the field of synthetic biology has been invaluable. I am also grateful for the financial support provided by the National Science Foundation and the Interdisciplinary Quantitative Biology program, which enabled me to conduct this research and contribute to the scientific community.

Finally, I extend my deepest appreciation to my family and friends. Your unwavering support, encouragement, and understanding have been my guiding light throughout this academic journey. I owe immense gratitude to my "boys," Felix, Mathias, and Karl, whose presence made this journey possible. A heartfelt thank you is also due to my best friend, Patrick, whom I admittedly did not call enough but whose unwavering support remains invaluable. I am forever grateful for his presence in my life. In addition, I am thankful for the cherished friendships I have made along the way, including but not limited to Andrea, Micha, Pierre-Aurelien, Tejasvi, and Thea (and her ability to drink endless amounts of coffee with me) in the UK, and Ange, Alex, Cassidy, Clair, Cooper, Daniel, Emily, Hayley, Jared, Kidus, Rosie, Thomas, Zach, and many others in Boulder. Each of you has played a significant role in my journey, offering support and camaraderie that I deeply appreciate. Special mention goes to my partner Emma, whose unwavering support, companionship, and moments of respite have been my rock and inspiration during this intense academic pursuit. Her presence, along with that of our beloved Moxie, has provided a sense of stability and joy amidst the challenges.

To each of you, and all my friends I could not list, my heartfelt appreciation for being an integral part of this journey and for contributing to the successful completion of this dissertation.

Contents

List of Tables

List of Figures

"It's the job that's never started as takes longest to finish, as my old gaffer used to say."

- Samwise Gamgee

1 Introduction

Synthetic biology offers potential solutions to a wide range of contemporary challenges. These challenges span from pressing environmental issues, such as climate change and pollution, to incurable diseases and pathogenic viruses, to everyday concerns like corrosion [34] (see Figure 1.1). This potential is harnessed by applying engineering principles to create innovative biological systems capable of responding to environmental signals, such as changes in temperature, the presence of pathogens, or exposure to toxic substances [40, 117, 229]. In one illustrative example, researchers have engineered the bacterium *Escherichia coli* (E. coli) to emit an optical signal when it detects the presence of a landmine in the nearby soil [22], showcasing the practical application of synthetic biology in addressing real-world challenges.

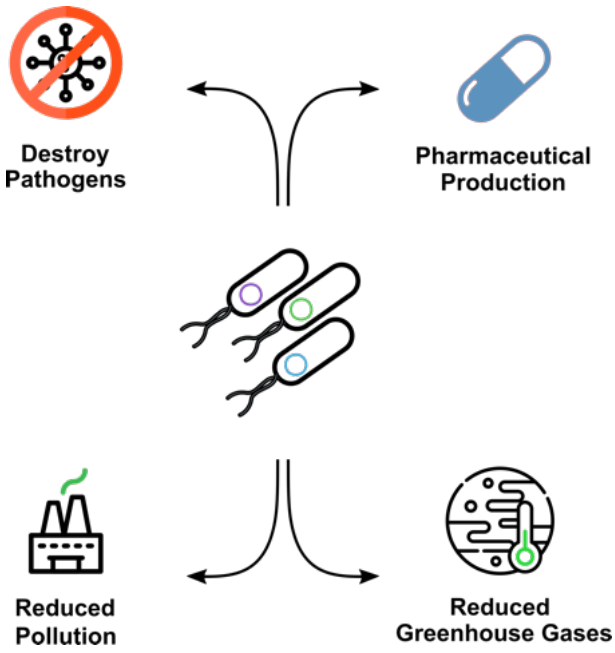


Figure 1.1: Practical applications of synthetic biology. Synthetic biology holds promise for addressing urgent challenges, such as pathogen destruction, pharmaceutical production, pollution reduction, and greenhouse gas capture.

Synthetic biology is commonly characterized as engineering biology. Benner et al. categorize synthetic biology into two main approaches: (1) the reproduction of behaviors from natural biology using unnatural molecules and (2) the assembly of components from natural biology into artificial systems [23]. According to Heinemann et al., synthetic biology involves the "engineering-driven building of increasingly complex biological entities for novel applications" [100]. Khalil et al. offer a similar definition, describing it as the collaborative effort between engineers and biologists to conceive and construct innovative biomolecular components, networks, and pathways for rewiring and reprogramming organisms [117]. Finally, in a more recent definition from 2016, Tsuda adds engineering principles like standardization to define synthetic biology as "the design and construction of biological components (e.g., enzymes, gene circuits, and whole cells) from scratch or from standardized parts" [218].

Breaking it down further, synthetic biology fuses the term "synthetic," which refers to the creation of substances through synthesis, to mimic natural products, with "biology," the study of living organisms. In essence, synthetic biology is the exploration of the synthetic creation of living organisms. This interdisciplinary field unites biologists and engineers to devise scientific approaches for the purposeful design and construction of novel synthetic biological systems tailored for specific applications. The innovative and purposeful design from the ground up distinguishes synthetic

biology from disciplines such as systems biology and genetic engineering.

While often perceived as an emerging field, "synthetic biology" as a term, was first coined in 1980 by Barbara Hobom to describe genetically engineered bacteria [218]. A little over two decades ago, a new era of synthetic biology dawned with the introduction of two genetically engineered constructs, the genetic toggle switch [85] and a genetic oscillator named the repressilator [71]. These synthesized constructs have the capacity to regulate cellular functions and the cell's behavior.

Today, synthetic biology has found applications in diverse domains, including protein engineering, metabolic engineering, artificial cell engineering [132], vaccine development, molecular diagnostics, cell-based therapeutics [214], and the bio-manufacturing of various products [202]. In this new wave of the field, it can be imagined that plants can be employed as sensors for pollution detection, bacteria can be used in the treatment of conditions like cancer or diabetes or for the synthesis of pharmaceuticals or biofuels. Moreover, utilizing cells for computing not only enhances our comprehension of biological processes, but also broadens the horizons of biotechnology's capabilities [16].

1.1 Synthetic Biology's Promise

A significant aspect of the field is devoted to designing innovative engineered genetic constructs in cells, commonly referred to as *genetic circuits*. In the early 2000s, the process of genetic circuit design primarily relied on manual and experimental methods conducted in laboratory settings [85, 71]. However, given the field's interdisciplinary nature, synthetic biology aims for a more systematic, model-based approach to designing genetic circuits, with a focus on automated construction [5, 166, 133, 207, 192]. For example, similar to electronic design, researchers have aimed to develop well-defined, characterized, standardized, and reusable biological components (*parts*) [222, 103, 197, 131]. Nevertheless, synthetic biology encounters unique challenges within the realm of engineering disciplines. Synthetic biology holds the promise of adapting and leveraging engineering principles, such as the reuse of parts and design information (*standards*), the analysis of behavior at a higher level (*abstraction*), and the separation of the design process from the

construction process (*decoupling*) [117], as illustrated in Figure 1.2.

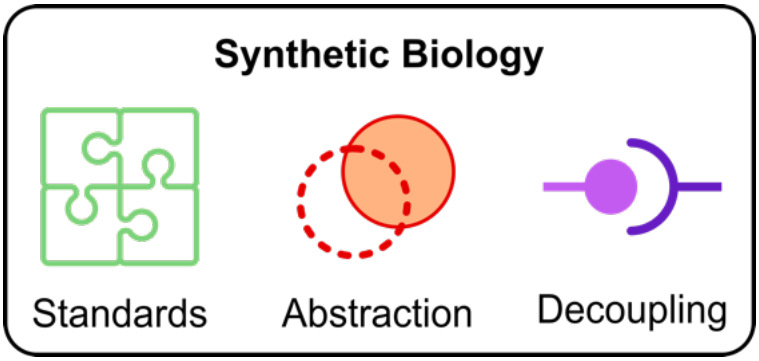


Figure 1.2: Engineering principles in synthetic biology. Reuse of parts and design information (*standards)*, analysis of behavior at a higher level (*abstraction*), and separation of the design process from the construction process (*decoupling*).

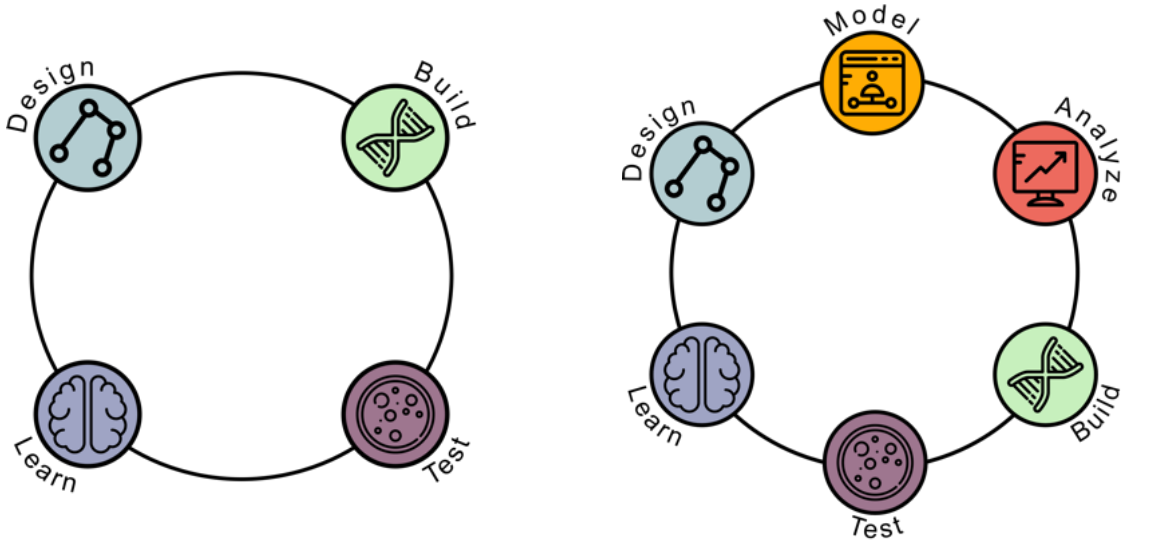
The introduction of engineering principles is essential for the bottom-up construction of genetic circuits. Standards play an indispensable role in technology and engineering, and they are equally vital in synthetic biology. They serve as the bedrock of every engineering discipline, with applications spanning architecture, electronics, mechanical design, and chemical synthesis [19]. In synthetic biology, standards become essential for precise property descriptions, reproducibility, and predictability. These are achieved through the establishment of clear definitions, descriptions, and characterizations of modular and reusable genetic parts [76, 121]. Moreover, standards foster effective communication among diverse research groups and streamline the compatibility of various software tools. Abstraction within the design process empowers researchers to work with higher-level descriptions, focusing on parts rather than intricate sequences. In the realm of modeling, abstraction simplifies mathematical models, thus expanding the scope of systems that can be effectively analyzed and simulated [192]. Finally, decoupling refers to the process of breaking down a complex problem into smaller, modular components that can be addressed independently. In the realm of synthetic biology, this entails separating the design and modeling aspects from the intricacies of the construction process. Designers can articulate specifications using high-level formalisms or languages, which are then translated into physical realizations. Essentially, this means that designers do not require detailed knowledge of the physical layer involved in the construction process.

1.2 Synthetic Biology's Workflow

Synthetic biology adheres to an iterative *design-build-test-learn* (DBTL) cycle, illustrated in Figure 1.3 (a). The cycle commences with an initial user-specified genetic circuit design, which is then taken to the laboratory for building. Following this, the genetic circuit undergoes testing within the specified environment. Typically, the initial attempt does not yield the desired results, prompting users to draw conclusions, learn from errors, and subsequently refine the design. Drawing inspiration from the field of *electronic design automation* (EDA), *genetic design automation* (GDA) methods have been developed to enhance the DBTL workflow [169]. Software tools have been created to support researchers in the *in silico* design, modeling, and analysis of genetic circuits. GDA encompasses the aforementioned key engineering principles, including standardization for part reuse and design information sharing, abstraction for higher-level analysis, and decoupling to separate the design phase from the construction process [117]. The primary aim of GDA is to streamline the iterative process by automating specific steps and guiding users in transitioning from functional descriptions to genetic design descriptions [172, 173, 175, 52], and eventually to mathematical models [80] for detailed analysis. Additional software tools have been developed to emulate the construction process, assist in testing, and facilitate data analysis during the learning phase.

Currently, a gap persists between computational and laboratory-based synthetic biology practices, impeding the advancement of the field. Although synthetic biology software tools are advancing, their integration into laboratory workflows remains limited. As a result, the field primarily attracts researchers with expertise in related disciplines. Given the increasing complexity and critical nature of genetic circuits, the long-term goal of this work is to encourage researchers, including those without computational backgrounds, to integrate GDA into their workflows. This work proposes a modification to the conventional DBTL cycle, emphasizing *modeling* and *analysis* following the design phase, as illustrated in Figure 1.3 (b), transforming the cycle into a model-based GDA workflow. This revised cycle entails the identification of potential failure modes in genetic designs, followed by computational analysis and evaluation, leveraging this insight to explore alternative

designs of the genetic circuit. Furthermore, to validate its findings, this work compares computational outcomes with real-world experimental implementations, thus underscoring and evaluating the efficacy of model-based GDA.



(a) Design-Build-Test-Learn cycle

(b) Design-Model-Analyze-Build-Test-Learn cycle

Figure 1.3: The iterative synthetic biology workflow. The traditional process (a), begins with the design of a genetic circuit, followed by building in the laboratory, testing, and utilizing any issues or undesired behaviors for learning and circuit redesign. In contrast, the updated, proposed workflow (b) introduces an additional phase involving modeling and computational analysis after the initial design step. This phase aims to enhance the circuit's design before progressing to construction, testing, and further learning, with the goal of reducing the total iterations required to achieve a robust design.

However, introducing modeling and analysis into the workflow also brings forth new challenges. Like engineers in all disciplines, synthetic biologists must prioritize robustness when designing genetic circuits. Robustness refers to a system's ability to function effectively despite external and internal disturbances. Nevertheless, modeling and analysis remains a challenge, primarily due to the inherent noisy and unpredictable nature of biological systems [72]. Closely linked to this issue is the accurate predictability of genetic circuits. Predictability refers to a model's capacity to accurately predict the likelihood of erroneous behavior through computational analysis. In the context of computational analysis, enhancing the predictability of a model becomes essential to precisely anticipate the likelihood of circuit failure.

Incorporating model-based GDA into the workflow facilitates the scaling up of genetic designs, their application, and their ability to control cellular behavior. Consequently, this expedites the design process and bolsters the predictability and robustness of designs, thereby increasing confidence in the final product. Enhancing computational modeling and analysis through model-based GDA conserves resources by enabling the prediction of a circuit's behavior *in silico* before physical construction in the lab.

The **objective** of this dissertation is to foster trust in synthetic biology software tools by enhancing the predictability of genetic designs using model-based GDA tools. The central **hypothesis** asserts that by emphasizing the advantages of fully utilizing model-based GDA tools, more researchers will adopt them, thus broadening the field's accessibility to a diverse, interdisciplinary, and non-expert community.

1.3 Contributions

The contributions of this work center on genetic circuit design, modeling, and analysis. In particular, this work focuses on three main aspects: the computational prediction of genetic circuit behavior, identifying the appropriate level of abstraction that balances accuracy and computational cost, and experimentally validating predicted erroneous behavior. The primary contributions are as follows:

- Specification of mathematical properties to describe erroneous behavior of genetic circuits
- Framework for computational analysis of genetic circuits using different analysis methods
- Evaluation of various modeling techniques of genetic circuits, considering intrinsic and extrinsic noise as well as characterized and default parameters
- Experimental validation of genetic circuit's erroneous behavior

As part of this work, properties for the effective testing of genetic circuits have been developed, allowing the computational analysis of their stochastic behavior. Testing defined properties

constitutes a fundamental component in various engineering disciplines. For instance, mechanical engineers rely on computational analysis for material testing [237]. In electrical engineering, asynchronous circuits undergo testing for delay faults due to their lack of clock control, unlike their synchronous counterparts [118]. Additionally, they are also tested for manufacturing faults commonly found in transistor-based circuits [187]. Similarly, in the realm of synthetic biology, analysis methods are applied to examine various properties, enabling the identification of unexpected behaviors. By enabling effective testing, these developed properties facilitate a more comprehensive analysis of genetic circuits, leading to more reliable computational assessments. They form the basis for precise computational predictions of a genetic circuit's robustness.

Reviewing software tools developed for synthetic biology reveals that most applications rely on *ordinary differential equation* (ODE) and *stochastic simulation algorithms* (SSAs) for computational analysis. In this work, to utilize the developed properties, an automatic model converter that includes an extensive collection of case studies has been implemented to translate ODE models into stochastic models. The stochastic models, together with the properties, can be analyzed using stochastic model checking. Stochastic model checking does not simulate the model and estimate the probability of the state of interest but calculates its actual true probability. Furthermore, stochastic model checking allows to return information on how the failure was reached. The included case studies aid the community in developing and improving software tools for model analysis.

Delving into the design phase, various modeling techniques were developed to compare different influences on the genetic circuit as well as the effect of different levels of abstraction. The same genetic circuit was modeled under varying conditions, including intrinsic and extrinsic noise effects, the utilization of laboratory-characterized parameters, and the use of default parameters obtained from literature. The comparison of results from these different models serves to assess the level of effort required to achieve accurate predictions. For instance, it helps determine whether extensive laboratory-based part characterization is necessary or whether using default parameters from existing literature sources is acceptable. Additionally, this analysis illuminates the influence of different sources of noise on a genetic circuit's behavior. Finally, the results indicate the level

of abstraction that can be used to compromise between the needed accuracy and computational feasibility.

Finally, it is essential to validate computational results with experimental findings to underscore the importance of *in silico* analysis. As previously mentioned, ensuring robustness is crucial to guarantee proper circuit functionality. Experimental results revealed that dynamic failures could impact steady-state behavior. Through *in silico* analysis, erroneous behavior was identified, and the underlying causes of failure were pinpointed. Subsequent computational analysis identified a dynamic failure, which was later confirmed through experimental validation via a modified experiment.

Overall, this work deepened the understanding of circuit failure modes and improved their analysis. Circuit failure has not yet been in the spotlight of the synthetic biology community. Given the increasing complexity of genetic designs and the progression towards more vital systems such as applications in human medicine, the necessity for robust and reliable circuits becomes paramount. Achieving this level of design integrity is contingent upon employing model-based GDA methodologies. Ultimately, this dissertation aims to bridge the gap between computational and laboratory synthetic biology and highlight the advantage of utilizing the full extent of genetic design, modeling, and analysis software in a laboratory workflow.

1.4 Dissertation Outline

This dissertation comprises seven chapters, each building upon the foundation established in the preceding ones. Chapter 2 lays the groundwork for the subsequent chapters by introducing essential context and key concepts. It begins with an exploration of fundamental biological principles, before diving deeper into the definition of genetic circuits. It is followed by attention to the computational aspects of this work, encompassing standards, part libraries, and the software tools used. By the end of the chapter, readers will possess a solid understanding of the foundational principles essential for this dissertation.

Chapter 3 outlines failure modes observed in genetic circuits. It begins by defining incorrect

genetic circuit behavior and proceeds to identify potential failures at the cellular level, as well as those arising from the logic implementation or the function of the circuit. This chapter further elaborates on various properties that characterize these failures. Finally, it concludes by elucidating the selection of parameters utilized for specifying these properties.

Chapter 4 explains the modeling and computational analysis of genetic circuits. The chapter first lays the groundwork by explaining the difference between ODE, SSA, and continuous time Markov chain analysis. Next, it introduces circuit 0x8E, first published by Nielsen et al. [175], that was used as a case-study throughout the presented work in this chapter. The main contribution is the multifaceted analysis of circuit 0x8E and two of its deviations, analyzing the probability of failing behavior for input transitions with function hazards, input transitions without function hazards, and its steady-state behavior. Finally, the chapter scores the three different implementations to see if one of the designs outperforms the others.

Chapter 5 extends the analysis conducted in Chapter 4 by repeating the analysis on circuit 0x8E model. This time, four additional computational models are employed, each utilizing different techniques. The results from Chapter 4 are then compared to the outcomes of the analysis conducted using these four additional computational models, each distinct from the others. The five models used are: an intrinsic noise model, an extrinsic noise model, a default parameter model, a characterized parameter model, and a stoichiometry amplified abstracted model. After comparison of the analysis results, the chapters contribution is the argument that even if different modeling techniques are used, the overall trend of the behavior remains, increasing the confidence in the obtained results.

Chapter 6 demonstrates the practical application of the analysis methodology discussed in previous chapters. It introduces a new circuit, circuit 0xF6 by Nielsen et al. [175], and outlines the laboratory-based analysis conducted on this circuit. The chapter then compares the results of this experimental analysis with the computational analysis performed using the methodologies presented earlier. Based on the findings from both laboratory and computational analyses, the chapter proposes a hypothesis suggesting that a circuit's steady-state behavior may be influenced by

its dynamic behavior over time. Finally, this chapter aims to experimentally validate this hypothesis by presenting the outcomes of additional experimental analysis conducted on circuit 0xF6.

In Chapter 7, the work is summarized, discussing the results in detail and presenting potential future research directions. These directions include refining failure definitions, advancing computational progress for model-based GDA by updating standards and model fitting techniques, enhancing methods for part characterization, and furthering laboratory automation.

"But do not despise the lore that has come down from distant years; for oft it may chance that old wives keep in memory word of things that once were needful for the wise to know"

- Celeborn, Lord of Lothlórien

2

Background

This chapter explains relevant concepts to provide a better understanding of the topics covered in this dissertation. Section 2.1 establishes the necessary background in biochemistry. Section 2.1.1 introduces the central dogma of biology [59], while Section 2.1.2 acquaints readers with genetic parts, their applications, and setup. Based off genetic parts, Section 2.2 delves into the engineering of genetic circuits, exploring various design paradigms for their implementation. This section also covers asynchronous digital logic, which serves as an abstraction to simplify DNA sequence design. Section 2.3 connects the biological foundation to computational aspects, introducing mathematical modeling techniques for genetic circuits. This includes the *classical chemical kinetic* (CCK) model [167] in Section 2.3.1, the *stochastic chemical kinetic* (SCK) model [167] in Section 2.3.2, and

seady-state and *dynamic modeling* in Sections 2.3.3 and 2.3.4.

Transitioning to the computational background, Section 2.4 delves into various standards employed throughout this work, encompassing Sections 2.4.1 to 2.4.4. Section 2.5 introduces part libraries that house a registry of genetic components, including those used in this work. Lastly, Section 2.6 presents the software tools used or developed as part of this research, which includes iBioSim [226] (Section 2.6.1), Cello [175] (Section 2.6.2), and STAMINA [188] (Section 2.6.3).

2.1 Fundamentals of Biochemistry

While many concepts in this work originate from engineering and have applications in both electrical/computer and biological engineering contexts, a foundational knowledge of biochemistry is necessary for understanding this research. This section serves as an introduction to the central dogma of biology and the concept of genetic parts, which are the building blocks used in the construction of genetic circuits.

2.1.1 Central Dogma of Biology

Cells can be categorized into two types based on the presence or absence of a nucleus. *Eukaryotic* cells contain nuclei, which are membrane-enclosed organelles that store genetic information in the form of *deoxyribonucleic acid* (DNA). While *prokaryotic* cells also utilize DNA, it is freely distributed within the cell and not enclosed by a membrane.

The central dogma of biology outlines the flow of genetic information in both eukaryotic and prokaryotic cells. According to this principle, genetic information is initially *transcribed* into *messenger RNA* (mRNA) and subsequently *translated* into proteins that play a vital role in constructing and maintaining the organism's life functions. The central dogma is visually represented in Figure 2.1.

The central dogma involves other key actors, including *RNA polymerase* (RNAP) and *ribosomes*. RNAP, an enzyme, facilitates the transcription process by binding to DNA, separating its strands, and transcribing one of them into mRNA. During translation, mRNA associates with

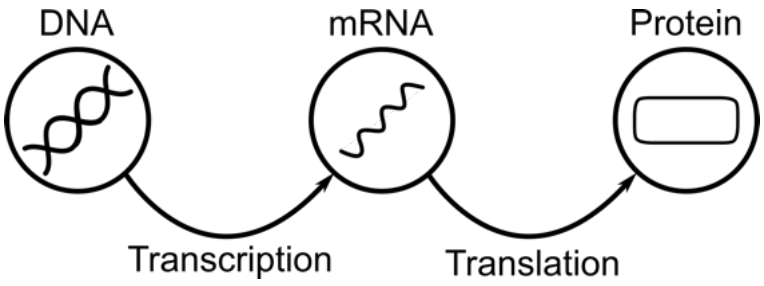


Figure 2.1: The central dogma of biology. A cell's DNA is transcribed into mRNA and subsequently translated into proteins. DNA is characterized by its double-stranded structure, while mRNA is most commonly single-stranded.

an available ribosome, which reads the information on the mRNA and assembles amino acids to construct a protein, following the provided instructions.

Segments of the DNA sequence that carry instructions for protein synthesis are commonly referred to as *genes*. The entire collection of genes within a cell is known as the *genome*. Cells regulate their gene expression in response to signals and environmental cues, enabling them to communicate with other cells and adapt to changing conditions. Researchers have the capability to manipulate a cell's DNA, harnessing its capabilities for computational purposes [94]. These modified DNA sequences, manipulating the central dogma, can be engineered to express specific proteins or to guide the cell's behavior in line with the designer's objectives, leading to the creation of constructs known as genetic circuits.

2.1.2 Genetic Parts

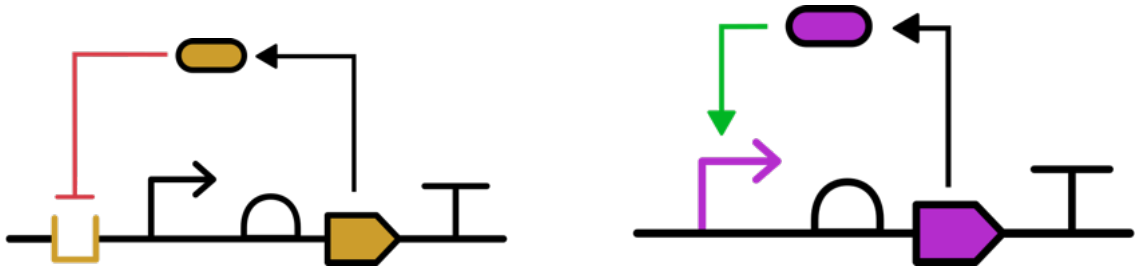
Inspired by engineering principles, genetic circuit components in synthetic biology are abstracted to enable the creation and modification of desired functions without direct DNA sequence manipulation. Synthetic biology achieves this by creating and reusing well-defined genetic parts, serving as modular building blocks [227], such as BioBricks [227, 145, 201]. These genetic parts commonly include components like *operators*, *promoters*, *ribosome binding sites* (RBS), *coding sequences* (CDS), and *terminators*.

Promoters define the DNA sequences where transcription begins, while operators provide sites for proteins to bind. Following the promoter, RBSs serve as binding sites for ribosomes on the mRNA, initiating translation from mRNA to the final protein. The CDS represents the DNA section encoding a protein using the genetic code, while terminators signal the end of transcription, acting as roadblocks for RNAP.

Figure 2.2 shows a visualization of the genetic parts mentioned. Often, the combination of these parts, as shown in the figure, is referred to as a *transcriptional unit* (TU). TUs are typically integrated into a plasmid vector, also known as a backbone, which is a small, circular DNA fragment often containing antibiotic resistance genes. This inclusion in the backbone ensures the stability of the entire DNA sequence. The interactions between the components in the figure are indicated by arrows. An arrow connecting the CDS to the protein represents the process of protein production. In Figure 2.2 (a), the red, flat-headed arrow denotes repression, indicating that the protein binds to the operator, thereby inhibiting transcription. This results in a negative feedback loop, where the protein counteracts its own production. In many cases, operators lie within the promoter. Therefore, instead of depicting operators and promoters separately, interactions are directly connected to the promoter, as shown in the example in Figure 2.2 (b). Here, the green arrow represents activation of the promoter, ultimately causing a positive feedback loop.

The proteins depicted in Figure 2.2, responsible for regulating transcription, are known as *transcription factors* (TFs). These factors play a pivotal role in controlling gene expression through activation or repression mechanisms by binding to operators situated either adjacent to or within promoters [35]. Their binding can either activate DNA transcription by facilitating the recruitment of RNAP or repress transcription by obstructing RNAP from binding to the DNA. Notably, transcriptionally regulated genetic circuits were employed in the original genetic toggle switch [85] and the repressilator [71].

Genetic parts exhibit varying behaviors within different genetic contexts [173, 219, 35, 42]. Combining multiple genetic parts can result in undesired interactions at the intersection of parts [231]. Nevertheless, standardization and a clear understanding of the expected behavior of genetic parts



(a) Repression (b) Activation

Figure 2.2: Common genetic parts. In (a), the DNA strand is represented as a horizontal line linking various genetic parts. Moving from left to right, these parts include: operator, promoter, ribosome binding site, coding sequence, and terminator. The oval shape represents a protein, with an arrow indicating protein production from the coding sequence. A red, flat-headed arrow denotes protein binding to the operator, inhibiting RNA polymerase attachment to the promoter, leading to the repression of protein production and creating a negative feedback loop. In (b), the genetic parts are depicted without the operator site, since the operator is often considered part of the promoter. The green arrow represents activation pointing from the protein directly to the promoter, resulting in a positive feedback loop. This abstracted view remains independent of activation or repression mechanisms.

are essential for accurate modeling and analysis of their function. The field has made substantial efforts in characterizing and standardizing parts [74], building techniques, gate conformation, and gate parametrization [121, 122, 163, 182, 201, 53].

To address the challenge of different genetic contexts, the inclusion of insulator sequences proves effective in preserving the performance of genetic parts in a wide range of genetic applications [175]. For characterization, one method involves the use of *relative promoter units* (RPUs) to measure promoter activity [116]. Promoters are characterized by comparing them to *in vivo* reference standard promoters, and the results are quantified in RPUs. The availability of a standardized kit for experiments simplifies the measurement of RPUs and encourages their adoption by various laboratories. Nevertheless, it's important to note that, despite the availability of these test kits, their effective use still requires expertise and effort.

2.2 Genetic Circuits

Genetic circuits, also known as *genetic regulatory networks* (GRNs) [110], involve multiple molecular regulators interacting to collectively control a cell's gene expression and shape its behavior. Genetic circuits are purposefully designed synthetic GRNs. Over time, a multitude of genetic circuits have been created for diverse applications, across various organisms, including bacteria, yeast, and fungi [68].

In synthetic biology, genetic circuits are built in a bottom-up approach. This process entails the selection and combination of different genetic parts to engineer desired functions. In contrast, genetic engineering primarily revolves around the modification or inactivation of existing genes within an organism's genome. This distinction, along with the engineering principles of standards, abstraction, and decoupling, highlights the fundamental difference between synthetic biology's approach to circuit construction and genetic engineering's practice of altering pre-existing genetic systems [66].

2.2.1 Genetic Circuit Design Paradigms

Different methods can be employed to implement genetic circuits within cells. Among these methods, cells can be programmed at the transcriptional level through gene expression regulation, at the post-transcriptional level using metabolic perceptrons [176], or protein interactions [155]. In the context of this work, the focus is on designing genetic circuits at the transcriptional level, involving the activation and repression of gene expression. However, a brief overview of three additional design paradigms is provided.

Apart from transcriptional circuit design based on DNA-binding proteins, these paradigms encompass adapted RNA-IN/RNA-OUT systems, *clustered regularly interspaced short palindromic repeats interference* (CRISPRi) regulators, and recombinase systems [35]. For a more comprehensive understanding of both transcriptional and post-transcriptional circuits, readers can refer to [75]. Ultimately, irrespective of the chosen design paradigm for genetic circuits, the growing complexity of larger designs mandates a model-driven approach, as manual consideration of all potential

interactions becomes impractical for researchers.

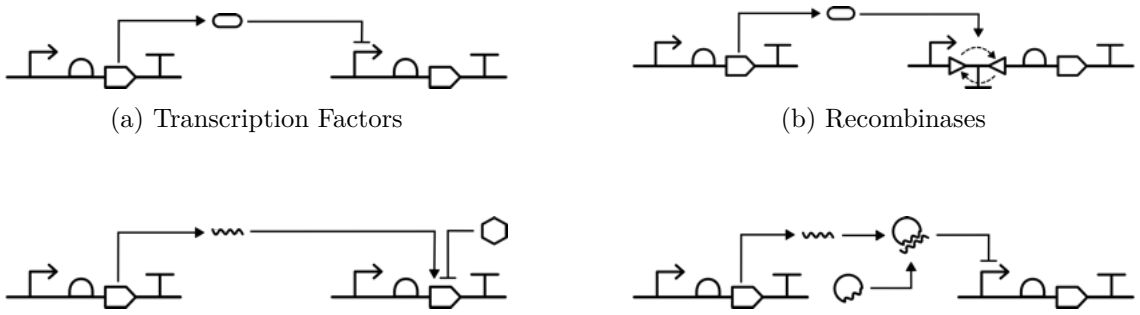
For completion it is noteworthy that synthetic biology circuits can extend beyond cellular implementation, with a specialized field known as cell-free synthetic biology [102]. A review of current cell-free synthetic biology platforms can be found in[108].

The genetic circuits in this work are constructed using TFs, as previously introduced. Figure 2.3 (a) provides a visual representation of how TF regulated genetic circuits operate. In the left TU, a protein is produced, as indicated by the arrow going from the CDS to the protein. This same protein then represses the promoter of the second TU, as shown by the arrow with the flat top. The protein binds to an operator within the promoter, preventing RNAP from accessing it and thus inhibiting transcription.

Flipping specific DNA sequences, such as promoters or terminators, provides an alternative method for controlling gene expression [205]. This process involves the use of recombinases, which are specialized proteins capable of flipping DNA sequences between predetermined binding sites [35]. For instance, recombinases can be employed to activate or deactivate terminators, thereby regulating the transcription of downstream coding sequences. Moreover, they can also be used to invert promoters, coding sequences, or any other sequence situated between these specified binding sites.

Figure 2.1 (b) illustrates this system. The recombinase binding sites are indicated by the two arrow heads on the DNA strand surrounding the terminator of the right TU. The left TU produces a recombinase protein responsible for flipping the direction of a terminator associated with the right TU. If the terminator is deactivated or flipped, as depicted in the figure, transcription of the protein of interest can proceed. Conversely, when the terminator is activated by flipping it in the opposite direction, RNAP will be unable to continue transcribing the protein of interest.

Gene expression can also be regulated using short RNAs, referred to as *non-coding RNAs*, which do not encode proteins like mRNAs [35]. This system was initially adapted from E. coli, where short RNA sequences bind to mRNA, impeding translation [134]. However, Liu et al. further modified this system to repress transcription [129]. In this system, the mRNA of the gene of interest (RNA-IN) is equipped with an additional binding site for a protein called *rho-factor*. When a specific



(a) Transcription Factors

(b) Recombinases

(c) RNA-IN/OUT

(d) CRISPRi

Figure 2.3: Illustration of four different genetic circuit design paradigms. (a) Shows a TF binding to a promoter of a downstream gene (right), preventing RNAP from binding and initiating transcription. In (b), a TU produces a recombinase that flips a terminator in the downstream gene, enabling RNAP to bind to its promoter and transcribe the CDS. (c) Illustrates the RNA-IN/OUT system, where an upstream (left) TU produces a short non-coding RNA that binds to an actively transcribed mRNA of a downstream gene. Through this binding, a rho-factor binding site is exposed, allowing the rho-factor protein to bind to the mRNA and halt the ongoing transcription. Finally, (d) demonstrates the CRISPRi system, in which a deactivated Cas protein is guided by a guide RNA encoded by an upstream TU to a downstream TU, where it binds to a target sequence, preventing RNAP from transcribing the unit.

short non-coding RNA (RNA-OUT) is transcribed, it leads to the binding of the rho-factor to the mRNA of interest, displacing the RNAP, and consequently repressing transcription elongation.

Figure 2.3 (c) provides a visual representation of the RNA-IN/RNA-OUT system. In this diagram, the left TU produces a short non-coding RNA, while the right TU transcribes its DNA into mRNA. The short non-coding RNA subsequently binds to the mRNA of the protein of interest, facilitating the binding of the rho-factor to the mRNA. This interaction displaces the RNAP and effectively terminates the transcription of the respective TU.

A fourth system to regulate gene expression is CRISPRi system based on modified Cas proteins [35]. Cas proteins are part of the CRISPR system, which functions as an adaptive immune system in bacteria [210]. CRISPR/Cas9 serves as a genetic scissor by guiding Cas (CRISPR-associated) nucleases to specific DNA sequences using guide RNAs [195]. The nuclease then cleaves the DNA at the specified location. Building upon CRISPR, CRISPRi employs guide RNAs and *deactivated Cas* (dCas) nucleases to obstruct specific DNA locations, impeding transcription by RNAP [104, 194, 29]. This system can also function as an activator by fusing an RNAP recruiting domain to the dCas protein, facilitating transcription instead [29].

Figure 2.3 (d) illustrates the CRISPRi system. The left TU encodes a guide RNA that binds to a dCAS protein and guides it to the TU of interest. There, it attaches to the promoter of the target TU, preventing RNAP from binding to the promoter and initiating transcription.

2.2.2 Asynchronous Digital Logic

Designing genetic circuits at the sequence level by manipulating individual base pairs quickly becomes unfeasible when dealing with larger circuits. Therefore, to assist researchers, various abstractions can be applied to conceptualize the behavior of genetic circuits. One such abstraction involves the application of *digital logic theory* to DNA. By employing digital logic abstraction commonly used in electronic circuit design, genetic circuits can be visualized as combinations of well-known logic elements, such as NOT, AND, or OR gates [233, 60, 219, 31].

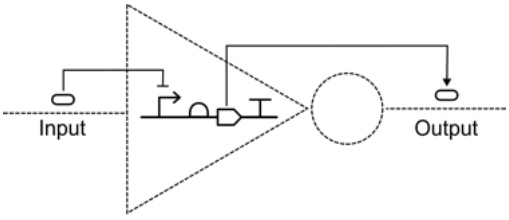
Logic gates determine their output based on input signals. Digital logic theory is a funda-

mental discipline in electrical engineering, foundational in the design of digital electronic systems, including computers and smartphones. Digital logic circuits rely on binary states, with each signal wire being in either a *high* (*on*) or *low* (*off*) state. These circuits calculate the states of their outputs based on the states of their inputs, employing different logic gates to perform various mathematical logic operations. For example, a NOT gate inverts the input signal, meaning that if the input is high, the output is low, and vice versa. An AND gate yields a high output only when both input signals are high, while an OR gate produces a high output when either or both of its input signals are high.

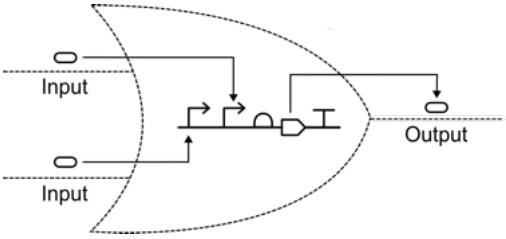
Figure 2.4 depicts the genetic implementation of a NOT gate (i.e., inverter) and an OR gate. The NOT gate in Figure 2.4 (a) comprises a single TU with a repressible promoter. When the input molecule is available, the signal is considered high or on. This high signal represses the promoter, preventing transcription and thereby inhibiting the production of the output, resulting in a low or off output signal. Thus, when the input signal is high, the output is low, and vice versa, which is characteristic of a NOT gate.

Figure 2.4 (b) illustrates a genetic OR gate. An OR gate is characterized by producing a high output signal if one or both of the input signals are high. The gate consists of a TU with two activatable promoters. When the upper input molecule is available, the right promoter is activated, leading to output production. Similarly, when the lower input molecule is available, the left promoter is activated, resulting in transcription and production of the output molecule. If both inputs are available, both promoters are active, leading to output production. However, if no input molecule is available, no promoter is active, and no output production occurs.

Similar to logic design in electrical engineering, digital circuits can follow the *synchronous* or *asynchronous* design paradigm. The synchronous design principle utilizes a global clock to update the states of the system. It remains difficult to implement a precise clock in biological systems. However, researchers put effort and progress towards successfully designing synchronous genetic circuits [141]. The asynchronous design paradigm does not rely on a clock to update its states and is, therefore, more applicable for genetic circuit design [173]. Asynchronous circuits can further



(a) Genetic NOT gate



(b) Genetic OR gate

Figure 2.4: Genetic logic gates. a) Illustrates a genetic NOT gate (i.e. inverter). When the input molecule is present (i.e. high or on), it represses the production of the output protein, thus the signal is considered low or off. b) Demonstrates a genetic OR gate. The TU producing the output is controlled by two activatable promoters. One promoter is activated by one input, and the other promoter is activated by the other input. Therefore, if either of the inputs or both are available, the TU is activated, resulting in the production of the output.

be divided into *sequential* or *combinational* circuits. The output of combinational circuits is state-independent, meaning the sequence of their inputs does not influence the output of the circuit. Sequential circuits, however, have feedback loops or a memory unit, making the output dependent on previous states and therefore depending on the sequence of their input changes [3]. Furthermore, it is important to note that genetic circuits can alternatively to digital abstraction be viewed as control systems as shown in [181, 65] or as analog circuits [62].

2.3 Genetic Circuit Modeling

For computational analysis, which is a key aspect of this work, genetic circuits must be effectively modeled to enable *in silico* analysis. These models provide researchers with the ability to predict the system's behavior, potentially uncover unknown or unintended properties, and even guide the overall design process. Mathematically, models empower researchers to construct the circuit in the laboratory based on existing knowledge of the system, replacing the trial-and-error approach [46].

While various modeling techniques have been developed, this work primarily utilizes *dynamic modeling*, a well-established approach in the field [199]. Dynamic models are particularly useful for describing the behavior of the various species within a genetic circuit, taking into account transient states and responses to changing external stimuli or inputs. The dynamic modeling of the genetic circuits in this work is accomplished using the *law of mass action*. GRNs, including genetic circuits, comprise chemical reactions that describe the interactions among various molecular species and DNA binding events [46, 146]. By applying the law of mass action, these chemical reactions can be transformed into a set of ODEs, providing a mathematical representation of the system's behavior. The aggregation of these ODE models is collectively referred to as a *kinetic-based* model.

2.3.1 Classical Chemical Kinetic Model

The law of mass action is used to model the chemical reactions within biochemical systems as CCK models. These models track chemical species' concentrations within cells over time, with each

model consisting of individual ODEs known as *reaction rate equations*. For a more in-depth understanding of CCK models, including mathematical definitions, reference to [167] is recommended. Consider the chemical reactions shown in Equation 2.1 as an example.

$$\begin{aligned} \text{pLac} &\xrightarrow{\text{k}_\text{o}} \text{np} \cdot \text{LacI} + \text{pLac} \\ \text{pLac} + \text{nc} \cdot \text{LacI} &\xleftrightarrow{\text{K}_\text{r}} \text{pLacLacI} \\ \text{LacI} &\xrightarrow{\text{k}_\text{d}} \emptyset \end{aligned} \tag{2.1}$$

This simple network comprises three species: the promoter *pLac*, the protein *LacI*, and the complex formed when the protein binds to the promoter, denoted as *pLacLacI*. These species interact through three distinct chemical reactions. The first reaction in the system involves the production of *np LacI* molecules through the promoter *pLac*. The second reaction describes the binding of *nc LacI* molecules to the promoter *pLac*, forming the *pLacLacI* complex and thereby repressing the production of *LacI*. The final reaction is the degradation of *LacI*. The rate constants for these reactions are denoted as k_o, $K_r = \frac{k_f}{k_r}$, and k_d, respectively, and are indicated above the arrows representing the direction of each reaction. According to the law of mass action, the rate of a reaction, which signifies the change in species concentration over time, is determined by the rate constant and the concentrations of the reactants raised to the power of their stoichiometry. By applying this law, the reaction system depicted in Equation 2.1 can be transformed into an ODE model, as presented in Equation 2.2.

$$\begin{aligned} \frac{\text{d[pLac]}}{\text{dt}} &= -k_f[pLac][LacI]^{nc} + k_r[pLacLacI] \\ \frac{\text{d[LacI]}}{\text{dt}} &= np \cdot k_0[pLac] - nc \cdot k_f[pLac][LacI]^{nc} + nc \cdot k_r[pLacLacI] - k_d[LacI] \\ \frac{\text{d[pLacLacI]}}{\text{dt}} &= k_f[pLac][LacI]^{nc} - k_r[pLacLacI] \end{aligned} \tag{2.2}$$

Analyzing the ODEs, it becomes evident that the availability of *pLac* increases through the reverse reaction when *pLacLacI* dissociates into *pLac* and *LacI*. Conversely, it decreases through the forward reaction when these two species bind to form *pLacLacI*. The concentration of *LacI* behaves similarly due to this reaction: it decreases when *LacI* and *pLac* bind and increases when

they dissociate. Additionally, $LacI$ increases proportionally with $pLac$ due to its production and decreases through degradation. Finally, the concentration of $pLacLacI$ is solely dependent on the binding of $pLac$ and $LacI$ through the second reaction.

CCK models offer valuable insights, even though their reaction rate equations are often challenging to solve. Researchers commonly turn to numerical simulations to analyze these models. However, CCK models come with specific assumptions. Firstly, they assume that the entire system is well-mixed, enabling the disregard of spatial considerations [146]. To account for spatial aspects, one must employ *partial differential equations* (PDEs) instead of ODEs. Secondly, the CCK model assumes that all reactions occur deterministically and continuously [167]. While this assumption is valid when a large amount of each species is involved, it may not hold true for GRNs. In GRN reactions, certain species, such as mRNAs, are not abundantly available, making noise and stochasticity dominant factors [150, 6]. Consequently, a stochastic modeling approach is required to accurately describe their behavior.

2.3.2 Stochastic Chemical Kinetic Models

The assumption that a GRN behaves deterministically and continuously is often violated, primarily due to the low number of molecules, typically on the order of tens or hundreds [167]. Addressing this issue, alternatively, chemical reactions can be viewed as stochastic processes. One way to address this stochastic behavior is by transitioning from CCK models to SCK models. In this framework, the conventional reaction rate constant k_μ is replaced by a specific probability rate constant c_μ, representing the probability of a reaction occurring through the collision of a randomly chosen combination of reactant species.

This model also operates under the assumption that reactions occur within a well-stirred medium to mitigate spatial effects. Moreover, while the system does not necessarily imply *chemical equilibrium* (i.e. reactant and product concentrations do not change), it assumes *thermal equilibrium* (i.e. no temperature change) to calculate the relative velocity of the involved species. For a more detailed explanation, please refer to the "Engineering Genetic Circuit" textbook [167].

Another method frequently employed in this dissertation to model stochastic processes are *continuous-time Markov chain* (CTMC) models. CTMCs are mathematical models designed to elucidate the dynamics of a stochastic system continuously over time [123]. Importantly, these systems can transition between various states as time progresses. In the context of a GRN it is represented by a finite or infinite set of states. Each transition carries a specific rate, signifying the probability of transitioning from one state to another. Consider the example shown in Equations 2.3:

$$R_1 : \; P \xrightarrow{k_p} P + M, \qquad R_2 : \; M \xrightarrow{k_d} \emptyset \tag{2.3}$$

with

$$\mathrm{k}_p = 1 \qquad \mathrm{P}(\mathrm{t}_0) = 1$$
$$\mathrm{k}_\mathrm{d} = 0.025 \qquad \mathrm{M}(\mathrm{t}_0) = 40$$

The system consists of two reactions R_1 and R_2 and a promoter P expressing the molecule M. R_1 is the production reaction, producing one molecule through the transcription of a TU including the promoter. The second reaction, R_2, specifies the degradation reaction of M. In the initial state, there are 40 molecules and one promoter. The reaction constants for production and degradation are given as k_p and k_d. An example of a CTMC state space is shown in Figure 2.5, illustrating the reaction system based on the reactions in 2.3.

The states within the CTMC represent the current quantity of molecules in the system. Subsequent states indicate whether reaction R_1 or R_2 has occurred, leading to an increase or decrease in the count of M, respectively. The reaction rate is determined by the number of reactants and reaction constants. Without a bounding condition (e.g., M cannot exceed 250), there is no restriction on the number of molecules that can be produced. Consequently, the model's state space is infinite, encompassing an unbounded number of states.

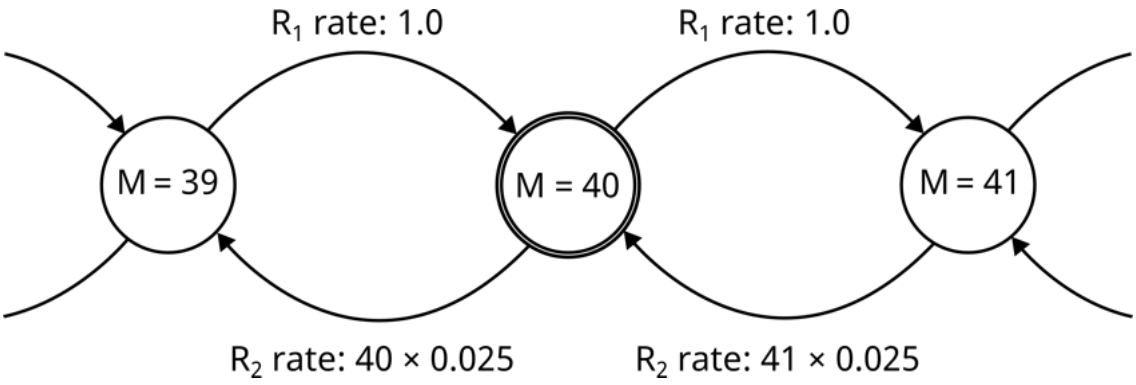


Figure 2.5: CTMC illustrating the reaction system described in Equations 2.3. The initial state of the system includes one promoter ($P = 1$) and 40 molecules ($M = 40$). Reaction R_1 involves the production of molecule M, while reaction R_2 represents the degradation of M.

2.3.3 Steady-State Modeling

The concept of *steady state* occurs when the production and degradation rates of a species reach chemical equilibrium, leading to no net change in the species over time. In models describing steady states, it is assumed that all chemical reactions have reached equilibrium. This assumption, referred to as the *steady-state assumption*, allows for the creation of steady-state models. These models are derived by solving the set of algebraic equations resulting from setting all the first derivatives in a kinetic-based model to zero [30]. Steady-state models simplify the mathematical description of a genetic regulatory network, focusing on its long-term behavior. Research has demonstrated the efficacy of this technique in genetic regulatory network modeling [146].

2.3.4 Dynamic Modeling

A different approach is necessary when there is interest in capturing dynamic behavior and the steady-state model fails. Instead of assuming that all reactions are at equilibrium, it is assumed that only fast reactions reach equilibrium. This *quasi-steady-state assumption is often applied to protein and enzyme reactions since they proceed much more rapidly than transcription and translation processes* [30, 183]. Protein dynamics reach equilibrium faster, enabling a more precise study of species dynamics that are not at steady state. However, a considerable difference in timescales between the dynamics of species that reach equilibrium rapidly and those that do not is required

to safely make the quasi-steady-state assumption [183].

2.4 Standards

The effective utilization of the mathematical models presented relies significantly on the use of standards. Standards are an essential component of technology and engineering and, consequently, are vital in synthetic biology. They play a crucial role in ensuring the reproducibility of experiments, connecting various GDA software tools, facilitating data sharing among collaborators, and enabling the submission of research results for publications. It is worth noting that standardization is not limited to laboratory procedures; it also encompasses computational file formats, information storage, and, as demonstrated, mathematical models. Standardization is indispensable for fostering seamless and efficient communication across diverse research workgroups due to the growing array of computational tools, prompting the community to establish these standards [38].

While some standards have been adapted from related fields like systems biology, synthetic biology has also established its own set of standards. Bioinformatics standards, such as GenBank [24] and FASTA [178], are often employed for representing DNA sequences. These formats are text-based and display the DNA sequence of a given design, with GenBank providing limited annotations regarding the functionality of parts within the sequence. In the laboratory, standards like BioBricks [201] and MoClo [107] have been developed for DNA assembly and cloning protocols, while SEVA [204] is used for representing basic components of plasmid vectors.

To represent hierarchical genetic designs that encompass DNA as well as other types of components (such as RNA, proteins, small molecules, etc.), the synthetic biology community has introduced the *synthetic biology open language* (SBOL) [37, 151, 138, 189, 83]. Additionally, the community has developed SBOL Visual, which provides a standardized set of glyphs for visualizing genetic designs [11, 185]. For modeling purposes, the community relies on the *systems biology markup language* (SBML) [115, 105]. To encode simulations conducted on these models, the *simulation experiment description markup language* (SED-ML) [208] is used, while the *COmputational Modeling in BIology NEtwork* (COMBINE) archives [25] are employed to package design and mod-

eling information together.

For the research presented in this dissertation, four key standards are of importance: SBOL, SBML, SED-ML, and the COMBINE archive. Their logos and role in the workflow are shown in Figure 2.6.

2.4.1 Synthetic Biology Open Language (SBOL)

SBOL is a community-driven, open-source standard for exchanging data related to genetic designs, based on a machine-readable xml-based format. Libraries for programming languages are available for accessing SBOL in Python [159], Java [236], JavaScript [153], and C++ [15]. Additionally, software tools that utilize this data format are available and covered later in this section. The SBOL community encompasses both academic and industrial groups. To ensure standardization and interoperability, it relies on ontologies such as the *sequence ontology* (SO) and the *systems biology ontology* [189]. Ontologies are like dictionaries, formally specifying terms and their relationships to another.

A key aspect of SBOL is that, unlike DNA sequence standards in other fields such as FASTA and GenBank, it goes beyond representing the DNA sequence alone. SBOL is primarily focused on capturing the composition and abstraction of genetic designs [170]. This means it has the capability to encode various components, including the biological chassis, proteins, metabolites, and their interactions. This feature is particularly valuable because designers are not necessarily required to have detailed knowledge of the specific design sequence; they can effectively represent the interactions within the system. Additionally, SBOL supports hierarchical designs, enabling the combination of separate designs into more complex systems. Lastly, SBOL offers the flexibility to add attachments, which could include experimental data or images related to a design.

Another aspect of SBOL is SBOL Visual, a standard designed to facilitate the graphical visualization of genetic designs. Currently, SBOL Visual includes a collection of glyphs that enable users to visually represent their designs. Users can easily download svg files containing these glyphs for use in graphic editors or can draw these glyphs manually. Alternatively, tools like DNAplotlib [67],

a Python package, or SBOLCanvas [216], a cloud-based tool, can assist users in creating designs more interactively. It is worth noting that the glyphs featured in Figures 2.2, 2.3, and 2.4 have been sourced from the SBOL Visual standard. All designs and visualizations in this work adhere to the SBOL or SBOL Visual standards.

2.4.2 Systems Biology Markup Language (SBML)

The SBOL standard is complemented by the SBML format. While SBOL stores design information related to genetic designs, including their sequence, structure, and interactions, SBML encodes the mathematical models of such designs. Therefore, SBML plays a crucial role in modeling genetic circuits. Models, such as CCK models as depicted in Equations 2.2, are typically stored as SBML files. SBML is a machine-readable format used for representing networks of biochemical reactions, encompassing various processes, such as gene regulation within genetic circuits, cell signaling pathways, metabolic pathways, and biochemical reactions.

SBML is encoded using xml, similar to SBOL, facilitating interoperability between different software tools and enabling reproducible science and collaboration. To work with SBML, there are tools and libraries available for various programming languages, such as libSBML [33] and jSBML [70].

SBOL and SBML complement each other significantly: one describes the system qualitatively, while the other provides a quantitative description. These standards are widely used within the community, and converters between them have been developed [174, 191, 80]. In this work, the mathematical models of all genetic circuits are stored in the SBML format.

2.4.3 Simulation Experiment Description Language (SED-ML)

The third standard used in this work, SED-ML, serves another crucial purpose in ensuring reproducibility. Similar to SBOL and SBML, SED-ML is a machine-readable xml-formatted standard. In a figurative sense, while the aforementioned standards provide the blueprints of a system, SED-ML encodes the instructions on how to utilize it. SED-ML includes information about the analysis

method, specifying the simulation algorithm and its parameters, which model is used, any modifications applied to the model, as well as the output and visualization of the results [224]. For this work, the models and designs are published alongside a SED-ML file, ensuring the reproducibility of results in the future and upholding good scientific practices.

2.4.4 Computational Modeling in Biology Network (COMBINE) Archive

COMBINE is a global community that coordinates the development of various standards and formats for computational models [168]. SBOL, SBML, SED-ML, and others are part of the initiatives led by this community. The COMBINE archive is often referred to as "the one file to share them all" since it serves as a means to bundle all the necessary information for reproducing a simulation study. This includes, but is not limited to, SBOL, SBML, and SED-ML files.

In order to enable future researchers to reproduce the studies conducted in this thesis, the COMBINE archive is used to gather and share all the pertinent information. This sharing of results is crucial for this thesis as well as for collaboration with others in the field. The wider success and scalability of this research field rely on the adoption and acceptance of such standards within the community [149]. The standards mentioned herein represent a significant step in the right direction.

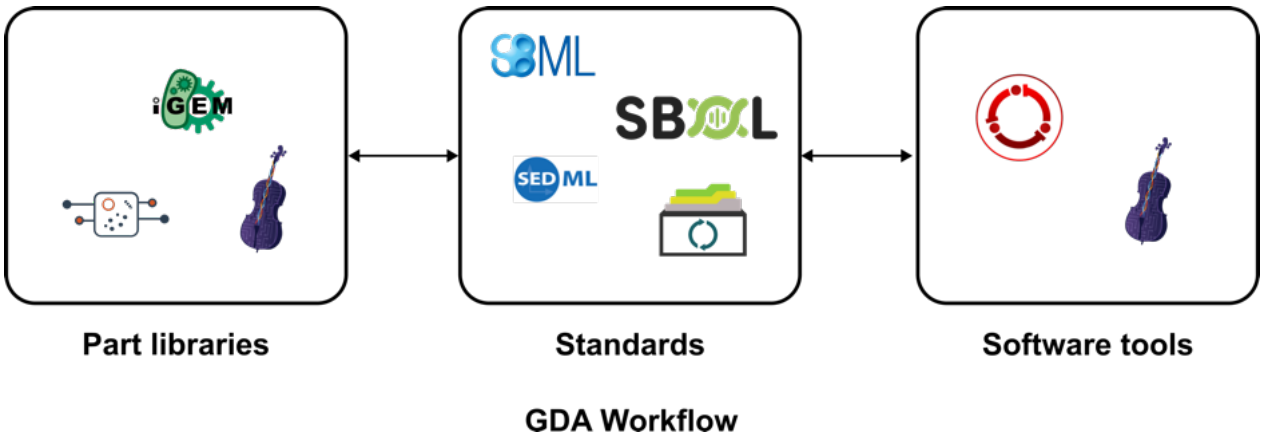


Figure 2.6: GDA workflow. Part libraries, standards, and software tools play pivotal roles in the GDA workflow. Part libraries contain comprehensive genetic part information, formatted in standardized formats, which can be readily accessed and integrated into various software tools. Within these tools, genetic components are assembled into designs, subjected to modeling and analysis, and the outcomes are then uploaded to repositories using the same established standards.

2.5 Part Libraries

Part libraries enable a reproducible and modular engineering approach to design. Community-developed part repositories store characterized genetic parts, facilitating the exchange of genetic design information. Accessing information about previously used parts allows researchers to efficiently design and build new genetic circuits. These part libraries can then be shared using data repositories, such as the *International Genetically Engineered Machines* (iGEM) registry of standard biological parts [221, 209] or the SynBioHub data repository [152]. Sequences can also be found in public sequence databases, such as those available from the *National Center for Biotechnology Information* (NCBI) [196], or in plasmid repositories, such as *Addgene* [111]. Examples of part libraries include the *Standard European Vector Architecture* (SEVA) database [204] for plasmid vectors and the *Joint Bio-Energy Institute Inventory of Composable Elements* (JBEI-ICE) [96].

The iGEM part library is an important resource used by the iGEM competition [36], in which interdisciplinary student teams design, build, and test genetic circuit designs. These teams use existing and contribute new registry parts. iGEM parts have been used, for example, to build an arsenic biosensor [14] and to control bacterial cellulose production [79]. However, curation of such a database remains challenging [144, 145].

Another important repository is SynBioHub [152]. SynBioHub is a design repository that allows users to upload their biological designs using standards like SBOL. SynBioHub enables the creation of collections that can be private or public and offers the option to create share links for sharing collections of designs with collaborators. Parts can be stored individually within the collection, including their DNA sequence, or as entire constructs. SynBioHub also allows users to visualize the design using SBOL Visual, highlighting the different components, sequence annotations, and sequence constraints. Furthermore, SynBioHub facilitates the upload of attachments such as experimental data or simulation results. Another aspect of SynBioHub is its database of genetic parts that be queried for research designs. Finally, SynBioHub, based on standards, can share the stored information with other software tools to seamlessly integrate it into the synthetic biology workflow. Some of these software tools are highlighted in the next section. Both, the iGEM

and SynBioHub logo and their role in the workflow are shown in Figure 2.6.

2.6 Software Tools

Software tools enable the design of genetic circuits through graphical user interfaces, making the field accessible to researchers who may not have expertise in laboratory work or extensive computational knowledge. These tools automate the design process by utilizing the aforementioned standards to access part libraries containing characterized genetic parts. Similar to electronic design, an in-depth understanding of the individual properties of each part is not required, as the software guides the user in creating a genetic circuit by combining standardized genetic components.

As published by the author in [38], Table 2.1, provides a non-comprehensive overview of software tools for GDA. In this context, GDA tools refer to those specifically developed for designing and modeling genetic circuits and their sequences. The table includes information about their application domains, support for standards and part libraries, and the availability of support and documentation. The first group of tools is developed by academic researchers, while the second group is developed by commercial vendors. Additionally, software tools can be categorized into sequence editors, which require users to work directly on the DNA sequence, and *high-level* (HL) design and modeling tools that introduce abstraction into the design process, allowing researchers to work with genetic parts rather than focusing on the sequence itself.

There are two high-level design tools that support the entire workflow by utilizing standards and connecting to part library repositories. First, iBioSim [226, 164, 190], is an actively developed, open-source academic tool for the model-based design of genetic circuits. Researchers can create and edit hierarchical genetic designs represented using SBOL and visualize them with SBOL Visual. iBioSim can automatically generate computational models represented in SBML for simulation [81, 158], and it can fetch parts and store design information in SynBioHub. It is used in the community; for example, Xiang et al. used it to design and analyze a genetic circuit for the detection of early biological markers of Lung Cancer [228].

Second, Cello [109, 52, 175], specifies genetic circuit designs using the *hardware description*

Table 2.1: Non-comprehensive list of software tools specifically developed for genetic circuit design. The table is categorized into two groups: academic and industrial. The academic tools originate from research institutions, while the industrial ones are developed by commercial entities.

Tool	Application	Standards	Part Library Repositories	Supported / Documented
ApE [63]	Sequence Editor	GenBank	-	Yes / Yes
Cello [109, 52, 175]	HL Design	SBOL	SynBioHub	Yes / Yes
Device Editor [50]	Sequence Editor	Genbank, FASTA SBOL, SBOL Visual	JBEI-ICE	Yes / Yes
Eugene [86]	Sequence Editor	FASTA, GenBank, SBOL	-	No / Yes
GeneTech [13]	HL Design	SBOL, SBOL Visual	-	Yes / Yes
GenoCAD [61]	Sequence Editor	GenBank, FASTA, SBML	-	No / Yes
iBioSim [226, 164, 190]	HL Design & Modeling	SBOL, SBOL Visual, SBML, SED-ML, COMBINE Archive, GenBank, FASTA	SynBioHub	Yes / Yes
j5 [101]	Sequence Editor	SBOL, SBOL Visual, FASTA, GenBank	JBEI-ICE	Yes / Yes
Mosec [156]	Modeling	SBML, CellML, GenBank, SBOL	-	No / No
Proto BioCompiler [20]	HL Design & Modeling	SBOL, SBML, CellML	-	No / No
SBOLCanvas [216]	Sequence Editor	SBOL, SBOL Visual	SynBioHub	Yes/ Yes
SynBioSuite [200]	Sequence Editor	SBOL, SBOL Visual	SynBioHub	Yes/ Yes
SBROME [106]	HL Design	-	-	No / Yes
Tellurium [154, 54]	Modeling	COMBINE archive, SBML, SED-ML, CellML, SBOL	-	Yes/ Yes
Tinkercell [48, 49, 47]	HL Design & Modeling	SBML, SBOL, SBOL Visual	-	No / Yes
Benchling *https://www.benchling.com*	Sequence Editor	GenBank, FASTA	AddGene, iGEM, NCBI JBEI-ICE	Yes / Yes
Doulix *https://getstarted.doulix.com*	Sequence Editor	SBOL, SBOL Visual GenBank, FASTA	Doulix	Yes / Yes
Geneious *https://www.geneious.com*	Sequence Editor	GenBank, FASTA	NCBI	Yes / Yes
Genetic Constructor [17]	Sequence Editor	SBOL, SBOL Visual, GenBank	iGEM, NCBI	No / Yes
Visual GEC [234]	HL Design & Modeling	SBML	-	No / Yes
Genome Compiler *https://designer.genomecompiler.com*	Sequence Editor	GenBank, FASTA	Addgene iGEM, NCBI	No / No
OpenVectorEditor *https://github.com/TeselaGen/openVectorEditor*	Sequence Editor	GenBank, FASTA	-	Yes / Yes
Snapgene *https://www.snapgene.com*	Sequence Editor	GenBank, FASTA	NCBI	Yes / Yes

language (HDL) Verilog, and maps them to parts from the Cello gate library. The Cello tool has been tested with 60 genetic circuits in E.Coli [175]. Example circuits designed using Cello include a drug delivery circuit for *Bacteroides thetaiotaomicron* [213] and genetic logic gates for *Pseudomonas putida* [215]. iBioSim and Cello are both important GDA tools used in this work. Their logos and connection to the workflow can be seen in Figure 2.6.

2.6.1 Intelligent Biological Simulator (iBioSim)

iBioSim is an actively developed open-source tool for the modeling, analysis, and design of genetic circuits. Its primary objective is to promote model-based design of genetic circuits using community-developed data standards such as SBOL [83, 189, 151, 138], SBML [105], and SED-ML [224]. Additionally, iBioSim facilitates the sharing of designs, models, and analysis results via the SynBioHub data repository [152].

iBioSim offers a comprehensive suite of tools for genetic circuit design, including the integrated SBOLDesigner software [235]. SBOLDesigner enables the creation and editing of hierarchical genetic designs, represented in compliance with the SBOL data standard and visualized using the SBOL Visual standard [185, 21, 11]. Genetic part information can be accessed directly from SynBioHub, and designs can be exported in the SBOL format. Furthermore, iBioSim supports the automatic generation of models from SBOL to SBML, leveraging the *Virtual Parts Repository* (VPR) model generator [157, 56]. VPR enhances the SBOL representation with non-DNA components and their interactions. This includes the addition of proteins produced by genetic circuits, specifying interactions between these proteins and the promoters within the design, and incorporating small molecules used as inputs and their interactions. The computational model is then created using the integrated SBOL to SBML converter [191, 80]. This converter translates structural and functional information in SBOL into a quantitative model expressed in SBML, using either generic or user-defined parameters. As part of this work, iBioSim was enhanced with a converter allowing to translate SBML models to CTMCs encoded in the PRISM language [124].

Finally, iBioSim includes various simulation methods for analyzing SBML models including,

but not limited to, ODE and stochastic simulations. The simulations are encoded using SED-ML, facilitating compatibility with other simulators. iBioSim was extensively utilized for numerous analyses, circuit construction, and modeling in this work.

2.6.2 Cello

In 2016, Nielsen et al. introduced Cello [175], a GDA tool that applies principles from EDA to genetic circuit design, streamlining and expediting the genetic design process. Users specify their desired circuit function using the Verilog HDL, and Cello automatically translates this into a corresponding DNA sequence. Cello was instrumental in automating the design of 60 combinational genetic circuits, all of which were tested in E. coli.

A pivotal component of this project is the Cello library, which consists of 12 orthogonal repressors. These repressors drive individual *expression cassettes*, each of which includes a stability element, RBS, CDS, and terminator. Additionally, for each expression cassette, there exists a corresponding repressible *sensor* unit, comprised of a spacer and a promoter. Figure 2.7 illustrates the F1 AmeR expression cassette and its associated sensor unit. The stability element, positioned upstream of the RBS, serves as an insulator for the expression cassette, preventing interference from other parts, such as promoters, added upstream. The sensor unit also incorporates an isolator, built out of a 15 DNA base pair spacer with a randomly generated DNA sequence.

The expression cassettes and sensor units within the Cello project are well-characterized and fine-tuned to ensure predictable behavior. This predictability enables the seamless interconnection of these components to construct genetic circuits with known and reliable outcomes. This work leverages genetic circuits designed by Cello and assembled using components sourced from the Cello library.

2.6.3 STochastic Approximate Model-Checker for INfinite-State Analysis (STAMINA)

Stochastic model checking has proven to be a highly effective method for studying biochemical systems, including GRNs and genetic circuits [140, 39]. A genetic circuit's stochastic model's state

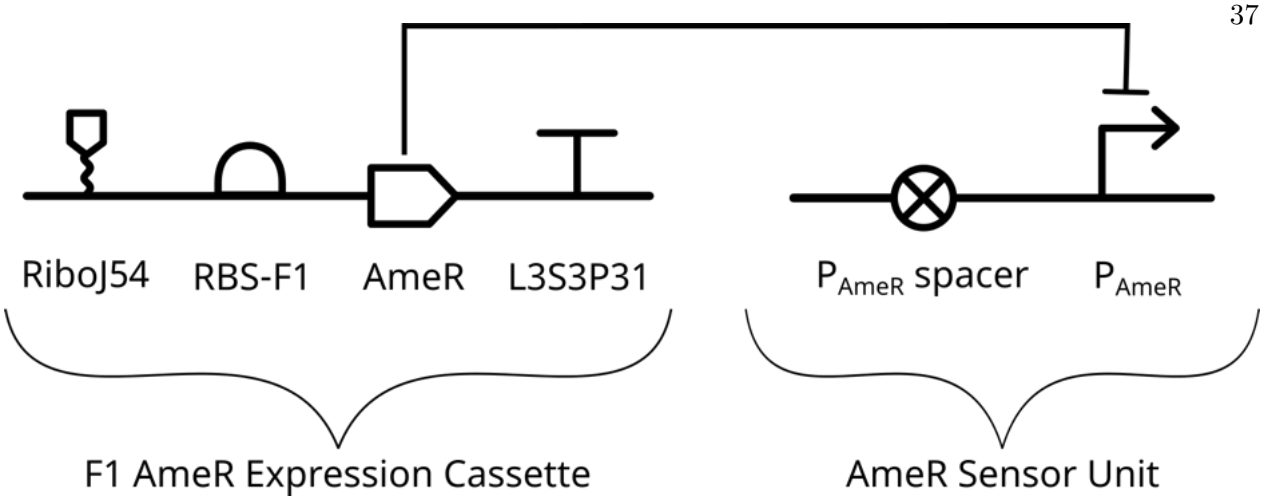


Figure 2.7: Cello expression cassette and sensor unit. The F1 AmeR expression cassette comprises a stability element (*RiboJ54*), a RBS (*RBS-F1*), a CDS (*AmeR*) encoding the AmeR protein, and a terminator (*L3S3P31*). The incorporated stability element functions as an insulator, safeguarding the cassette from upstream genetic parts. The sensor unit is composed of a 15 DNA base pair spacer (P_{AmeR} *spacer*) and a repressible promoter (P_{AmeR}) that corresponds to the cassette.

space represents all possible states that the genetic circuit can reach. Stochastic model checking, in turn, calculates the probability of a specific event occurring during the system's execution. This specific event, often referred to as *property*, is user-defined and encoded in probabilistic temporal logic [125]. Probabilistic temporal logic defines the likelihood of events and their temporal relationships with other events [123]. This approach differs from conventional model checking as it focuses on the probability of failure rather than just its possibility. Instead of determining whether an event can be reached, it quantifies the likelihood of that event being reached.

Numerous algorithms have been developed for stochastic model checking, with additional information available in [123, 10, 9, 58, 97, 220]. Currently available tools include, but are not limited to, PRISM [124], STORM [64], STAMINA [171, 188], INFAMY [95], and STAR [162]. Nevertheless, a challenge remains when applying these tools to GRNs.

PRISM and STORM are effective tools for analyzing stochastic models, but they are mostly confined to models with finite state spaces. In contrast, synthetic genetic circuit models do not have finite state spaces; they possess countably infinite state spaces. To overcome this limitation, alternative approaches have been developed. Tools like STAMINA, INFAMY, and STAR offer

methods to transform infinite state space models into finite representations, enabling their analysis using tools such as PRISM or STORM.

In this study, STAMINA is used to obtain a finite representation of the infinite state spaces of the genetic circuit models. Subsequently, PRISM and STORM, integrated in STAMINA, are utilized for stochastic model checking throughout this research. The next chapter, in conjunction with standards, part libraries, and tools, establishes properties of interest in genetic circuit design. These properties serve as the basis for analyzing and predicting the behavior of various genetic circuits, employing both simulation and model checking approaches.

"If we fail, we fall. If we succeed - then we will face the next task."

- Gandalf the White

3

Failures of Genetic Circuit Behavior

Chapter 2 introduced the concept of standardized and predictable genetic circuit parts. However, in reality, genetic parts are considerably less predictable and standardized than their electronic counterparts [35, 42]. Despite ongoing efforts to develop perfectly predictable and reliable genetic parts, ideally with data sheets akin to those in electronic design, it is highly improbable that they will ever attain the level of predictability and standardization observed in electronic components [41]. Genetic circuits, owing to their biological nature, depend on protein-protein and protein-DNA/RNA interactions rather than physical wiring. Consequently, unlike electronic circuits, genetic circuits are much more susceptible to the influence of noise. Researchers frequently report faulty behavior in their genetic designs at the transcriptional, translational, and cellular levels [35, 175, 120].

This chapter initially outlines incorrect behavior in genetic circuits by introducing two circuits: *circuit 0xF6* and *circuit 0x8E*, both designed computationally using the software tool Cello [175] (Section 3.1). These circuits were constructed using well-characterized Cello parts but are of particular interest due to the detection of faulty behaviors during laboratory testing. The examples presented will guide the reader through the chapter.

Following the discussion of the empirical failures observed, the subsequent Section 3.2 systematically defines potential failure reasons. The section begins by addressing reasons for circuit failure with a focus on biological aspects (Section 3.2.1). Commonly reported failure modes of this nature include read-through [35, 53], roadblocking [173, 175], crosstalk [57, 173, 219], and signal mismatch [173, 219, 35, 230]. Additionally, the low molecular count within the cell renders the genetic design susceptible to stochastic and noisy behavior [193, 127, 186, 72, 217].

Despite the biological reasons of failure, abstractions such as digital logic and control theory can provide researchers with a different perspective on understanding circuit failures. Section 3.2.2 delves into failures stemming from the logic implementation, referred to as *logic hazards* of a circuit. The last failure mode elucidated in Section 3.2.3 pertains to failures resulting from the function of the genetic circuit itself, referred to as *function hazards*.

Section 3.3 subsequently introduces the properties needed for the analysis of failures in the context of abstractions. The section begins with the mathematical definition of these properties (Section 3.3.1) and concludes by outlining the parameter evaluation, justifying the specific parameters used in the analysis of the circuit in the following chapters (Section 3.3.2).

3.1 Definition of Incorrect Genetic Circuit Behavior

To investigate genetic circuit failures, it is necessary to establish criteria for determining when a circuit is malfunctioning rather than operating correctly. There are various ways in which a circuit can exhibit incorrect behavior. One evident failure occurs when a circuit does not produce the expected output, manifesting unexpected and sustained behavior that does not self-correct. These failures pertain to the circuit's *steady-state behavior*. The state of a circuit is defined by the

configuration of its internal molecules, representing the status of signals governing the circuit as either on or off. A steady-state failure signifies a situation in which the circuit's output deviates from the expected output for a given state. In other words, the output stays off when it should be on, or vice versa. Such failures can occur when the circuit lacks the capability to effectively discriminate between on and off signals.

Another failure arises when circuits manifest undesirable behavior during transitions between states. When the circuit transitions from one state to another, the internal signals must adapt to the new state. However, during this transition, the circuit may exhibit an unexpected output before reaching the intended final state. Since this failure is observable only during the transition and not at the final steady state, it is categorized as *transient behavior.*

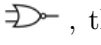
The prevention of these failures is of paramount importance, especially when the circuit is deployed in safety-critical applications. For example, in scenarios where a circuit is tasked with producing pharmaceuticals within a biological system, it is crucial to ensure the correct operation of the circuit. Failures in steady-state behavior can lead to issues such as the drug not being released when needed or being dispensed in inadequate concentrations to achieve the desired effect. Similarly, failures in transient behavior can have serious consequences, for instance, when the drug is unintentionally released prematurely, contrary to the intended timing.

3.1.1 Steady-State Failure of Circuit 0xF6

As mentioned in Chapter 2, genetic circuits can be regarded as asynchronous digital circuits. The behavior of a digital circuit is defined by its *Boolean function.* Boolean functions calculate an output state of a system to be either on or off based on the input states of the circuit. Instead of mathematical operators like addition, subtraction, and multiplication, Boolean functions rely on logic operators such as NOT, OR, and AND gates. Genetic implementations of these logic operators are illustrated in Chapter 2.

A Boolean function can be depicted as a *truth table*, illustrating columns for the input signals and output signal. Typically, an on signal is denoted by a 1, and an off signal is represented by a 0.

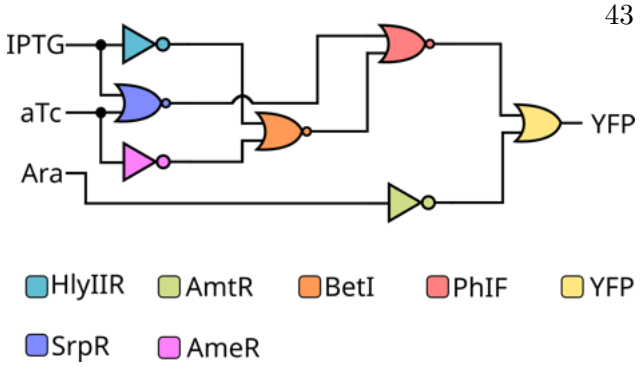
Figure 3.1 (a) displays the truth table of circuit 0xF6, originally introduced by Nielsen et al [175]. The circuit is named according to its outputs, *11110110*, which corresponds to "F6" in hexadecimal notation. This circuit is non-sequential as it lacks a memory unit or state-holding gate, rendering it combinational. The circuit responds to the presence of three inducer molecules (i.e., input signals): *Arabinose* (Ara, ChEBI=17535), *Isopropyl-beta-D-thiogalactopyranoside* (IPTG, ChEBI=61448), and *Acetylcholine* (aTc, ChEBI=15355), producing *yellow fluorescent protein* (YFP) as an output. YFP is a fluorescence protein absorbing light at 513 nm (green/blue light) and emitting yellow light at 530 nm. For instance, in the absence of any inducer, as shown in the first row of the truth table, the circuit yields a high output, indicated by YFP production. Conversely, when all inducers are present, as demonstrated in the last row, no YFP is produced, and the output signal is off. For this work, the output reporter YFP used by Cello was replaced with *superfolder green fluorescence protein* (sfGFP) [180], which is more stable with visible fluorescence, making it easier to detect. sfGFP absorbs light at 488 nm (green/blue light) and emits green light at 510 nm.

The combination of logic gates, often referred to as the circuit's logic, is illustrated in Figure 3.1 (b). The circuit comprises three NOR gates, denoted by  , three NOT gates  , and one OR gate indicated by  , all adhering to the IEEE Std 91/91a-1991 standard. In digital logic, a NOT gate inverts the input, meaning a high input leads to a low output and vice versa. OR gates produce a high output if either or both of the inputs are on. NOR gates are essentially OR gates with an additional NOT gate, inverting the output. The color coding represents the expression cassettes employed for gate implementation. For instance, the blue gate is based on the pSrpR expression cassette, producing the SrpR protein. The two promoters steering this expression cassette are repressed by the inducers IPTG and aTc.

Circuit 0xF6 is noteworthy for its steady-state behavior. In the context of this dissertation, experimental flow cytometry analysis, as conducted in the Cello paper [175], was replicated in a laboratory setting. For preparation, three cultures of bacteria, referred to as *bacterial replicates*, containing the circuit were cultivated in *Luria-Bertani* (LB) broth—a nutrient-rich medium conducive to optimal bacterial growth. Subsequently, these populations were transferred to three

Inputs			Output
IPTG	aTc	Ara	YFP
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	0

(a) Truth table of circuit 0xF6



(b) Logic of circuit 0xF6

Figure 3.1: Circuit 0xF6, as presented by Nielsen et al. [175]. (a) The circuit's function is illustrated through a truth table. The original circuit comprises three inputs —IPTG, aTc, and Ara— and one output, YFP. (b) Depicts the circuit's logic, consisting of three NOT gates, three NOR gates, and one OR gate.

distinct rows of a 96-well plate. Each well in a row held a different concentration of the three inducer molecules, representing all eight possible input combinations as depicted in the Truth Table 3.1 (a). After incubating the cells for three hours, their current state was arrested using a *stop solution*. The stop solution comprises a mixture of approximately 2.5 mg/mL chloramphenicol solution and *phosphate-buffered saline* (PBS). PBS is included to help maintain osmotic pressure in the cells. The purpose of the stop solution is to rapidly and completely halt cell growth. Finally, the 96-well plate was subjected to analysis through flow cytometry.

Flow cytometry uses a laser to assess the fluorescence of individual cells by pulling them through a tube individually. Besides measuring the fluorescence, this method also enables cell counting. The results of this analysis are depicted in Figure 3.2. This figure illustrates the outcomes for the three biological replicates highlighted in different shadings, along with the results of three sfGFP-expressing bacteria employed for normalization and calculating RPUs. Each column in the plot represents the fluorescence of a state in logarithmic *relative fluorescence units* (RFU) against the number of cells on the x-axis. For instance, the initial column of biological replicates one, two, and three showcases the state IPTG, aTc, and Ara = (0, 0, 0), denoting an on state. The average fluorescence for this state exceeds 10^3. The measurement forms a normal distribution, with over

500 cells contributing to the average fluorescence.

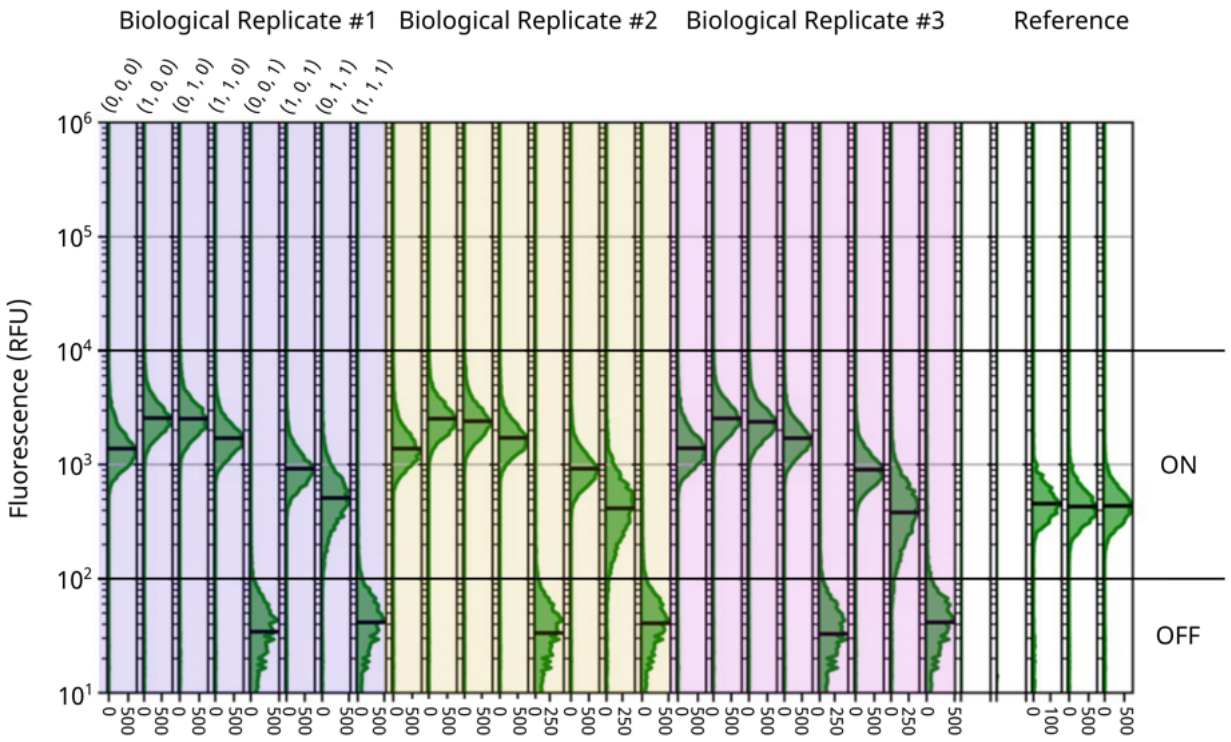


Figure 3.2: Flow cytometry analysis of circuit 0xF6. The analysis was conducted on three biological replicates indicated in different shadings, along with three RFU expressing bacteria for normalization. Each column in one biological replicate represents one of the eight possible states, such as the first column indicating state IPTG, aTc, Ara = (0, 0, 0). The results are presented as a cell's fluorescence in RFU over the number of cells, resulting in a normal distribution. Six out of the eight state reach a high output with two states remaining low.

The outcomes indicate variations in fluorescence among different states. Despite having six distinct states with high output, each state does not exhibit equally high signals. While it is feasible to distinguish the six on states from the two off states, it is evident that some on states do not display the same level of activation as others. This undesired behavior constitutes a steady-state failure.

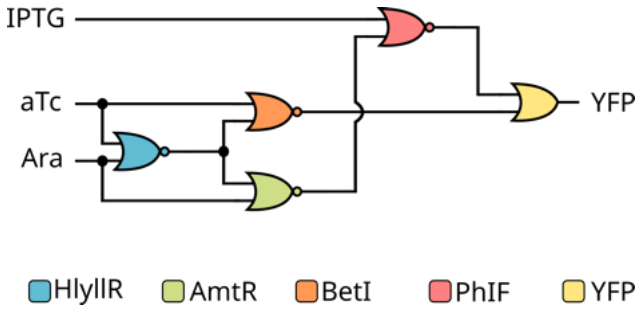
3.1.2 Transient Failure Behavior of Circuit 0x8E

Similar to circuit 0xF6, circuit 0x8E also exhibited a failure in the laboratory, albeit of a different nature. Failures do not only occur during steady state, but also during state transitions.

These failures only occur during the transition phase and correct themselves over time when steady state is reached. Therefore, they are called *transient failures*. Figure 3.3 (a) presents the truth table for circuit 0x8E, as initially introduced by Nielsen et al. [175]. It is named after the hexadecimal conversion of the binary number *10001110*, the circuit's output, which corresponds to "8E". Similar to circuit 0xF6, this circuit is non-sequential, lacking a memory unit or state-holding gate. It responds to the presence of the same three inducer molecules—IPTG, aTc, and Ara and also generates YFP as its output.

Inputs			Output
IPTG	aTc	Ara	YFP
0	0	0	1
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	0

(a) Truth table of circuit 0x8E



(b) Logic of circuit 0x8E

Figure 3.3: Circuit 0x8E as published by Nielsen et al. [175]. (a) Shows the function of the circuit as a truth table. The circuit has three inputs, IPTG, aTc, and Ara, as well as one output YFP. (b) Shows the logic of the circuit. The circuit consists of four NOR gates and one OR gate.

The circuit comprises four NOR gates, denoted by  , and one OR gate, represented by  , adhering to the IEEE Std 91/91a-1991 standard. The color coding again corresponds to the expression cassettes employed for gate implementation. For instance, the blue gate utilizes the HlyIIR expression cassette, producing the HlyIIR protein. The two promoters steering this expression cassette are repressed by the inducers aTc and Ara.

This circuit is particularly interesting due to an observed transient behavior failure in experimental results [175]. In an experiment, the circuit was initially set to the states Ara, aTc, IPTG = (0, 0, 0). After three hours, the different inputs were set to the other eight states. Figure 3.4 displays the results of a time course experiment for circuit 0x8E. The x-axis represents time in hours,

and the y-axis, in a logarithmic scale, represents RPUs. During the experiment, flow cytometry analysis was conducted every 30 minutes over a 5-hour period.

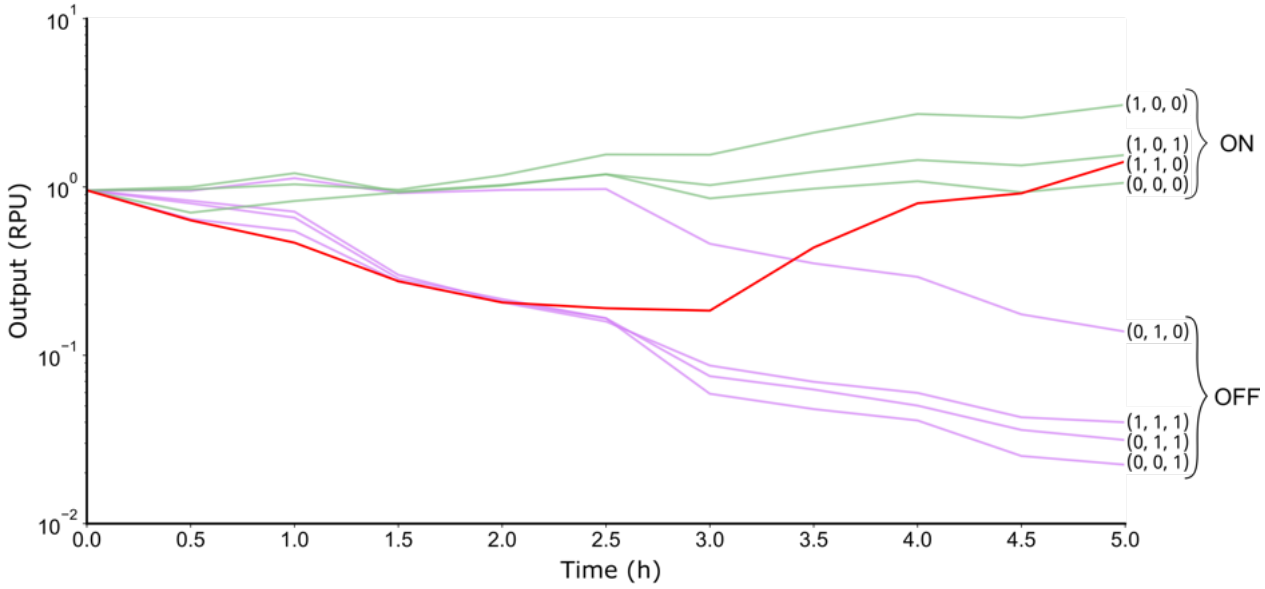


Figure 3.4: Time course analysis of circuit 0x8E using flow cytometry. The circuit was initially incubated in state Ara, IPTG, aTc = (0, 0, 0). After reaching a steady state, inducers were added to transition to all other states. Throughout the transition, flow cytometry was conducted every 30 minutes for 5 hours. At the five-hour mark, the states were grouped into their respective on (shown in green) and off (shown in purple) states. While all states reach their expected steady state, the transition from (0, 0, 0) to (1, 1, 0) (shown in red) briefly turns off instead of staying on as expected. Courtesy of [81].

The circuit initially starts in an on state with inducers set to (0, 0, 0). The plot illustrates eight transitions to all other states. On the right-hand side, the states are grouped based on whether they end up in an on (green) or off (purple) state. Comparing these states with the truth table reveals that the circuit correctly reaches the intended steady state, producing the expected output. However, special attention is given to the transition from (0, 0, 0) to (1, 1, 0), highlighted in red. Both the initial and final states produce a high output. However, during this transition, the circuit briefly turns off before returning to its final on state, indicating a transient failure.

3.2 Genetic Circuit Failures

As illustrated, real-world failures have been observed, presenting as undesired or unpredictable switching behavior in the circuit's output. These manifestations of faulty behavior take different forms in practice. Initially, during the transition from one state to another, the output briefly turns off, contrary to the expected maintenance of the on state. This transient failure, observed in circuit 0x8E as mentioned earlier, is identified as a *static 1→1 glitch* and is depicted in Figure 3.5 (a). The anticipated behavior, where the output remains high, is represented by the black dashed line. However, it briefly turns off after the input transition (dashed blue line).

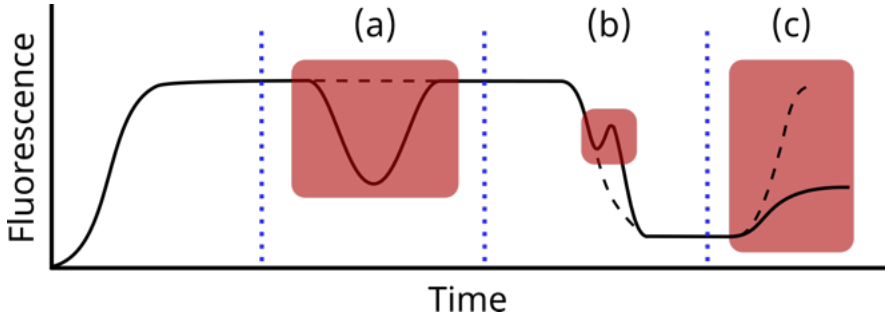


Figure 3.5: Potential failures of a genetic circuit's output are illustrated in the graph depicting the circuit's fluorescence over time. Dashed blue lines signify input transitions. (a) Unexpected switching occurs after the input transition, where the output is expected to stay high both before and after the change. However, the signal briefly turns off before recovering and returning to the on state. (b) A different unwanted switching behavior occurs after a second input transition, where the circuit is intended to turn off. During the turn-off process, the circuit briefly turns on again before ultimately switching off. The behaviors in both (a) and (b) self-correct over time, categorizing them as transient. Lastly, (c) displays a steady-state failure. Despite being expected to turn on after the final input change, it settles in the incorrect steady state, between on and off.

Another transient failure is a *dynamic 1→0 glitch*, illustrated in Figure 3.5 (b). Following the second input transition, indicated by the blue dashed line, the circuit's output is expected to transition from on to off. However, during this transition, the output briefly switches back on before ultimately settling in the off state.

The transient behavior depicted in Figure 3.5 (a) and (b) is attributed to *hazards* within the circuit. A hazard denotes the potential occurrence of an undesirable effect, whether stemming from the circuit design or external influences. While a hazard only signifies the possibility of such

a failure, a *glitch* characterizes the actual occurrence of the failure. It is crucial to recognize that glitches signify transient failures only, implying that the failure corrects itself over time. However, if the output is irreversible, the circuit's function can be compromised.

Finally, the steady-state failure observed in circuit 0xF6 is depicted in Figure 3.5 (c). Following the last input transition in this case, the output is expected to switch on. Nevertheless, it remains in an incorrect steady state, staying below the expected on state but above the off state.

It is crucial to recognize that each failure has an opposite counterpart. A *static 1→1 glitch* refers to a momentary turning off of the circuit when it should remain on. In contrast, a *static 0→0 glitch* involves the circuit, expected to stay off, briefly turning on. Similarly, for the *dynamic 1→0 glitch*, there is a *dynamic 0→1 glitch*. The former describes a transition from an on to an off state with a brief relapse to output production, while the latter involves a transition from off to on, with the output briefly turning off again. Ultimately, the circuit can also erroneously settle in an incorrect steady state by remaining high when it is supposed to be off.

Various approaches can explain the occurrence of these failures. Firstly, they may arise from cellular-level issues, as discussed in the next section. Secondly, failures can stem from the logic of a circuit, similar to circuits in electronic design. Thirdly, failures can also be introduced due to the function of the circuit itself. It is crucial to note that, unlike electronic circuits, glitching behavior in genetic circuits can be reduced but not completely eliminated. Input transitions, even without hazards, still have a low probability of glitching, primarily due to the underlying noisy behavior of biological systems.

Hazards are crucial considerations in digital circuit design to ensure correct and reliable operation. Techniques like hazard detection and hazard removal play a significant role, involving the introduction of additional logic elements or changes in the circuit design to eliminate or mitigate the effects of hazards. Proper handling of hazards is paramount in critical digital systems, particularly in applications with safety-critical requirements and high-speed designs.

3.2.1 Genetic Circuit Failures on the Cellular Level

Genetic circuits require fine-tuning of their regulation to initiate the desired response. The placing of the circuit in the host genome, its genetic context, as well as the environment, and its growing conditions all influence the circuit's behavior. As mentioned in Chapter 2, genetic circuits are built by connecting standardized genetic parts. However, the characterization of these parts happens in isolation and a specified environment. Introducing those parts in a new environmental, cellular, and genetic context results in variation of the parts performance [173, 219, 35, 42]. For example, the behavior of a specified component varies depending on its adjacent components [77, 163, 131]. Additionally, combining parts can result in unintended functional sequences at the part-junction creating new functional parts that interfere with gene expression [231]. Therefore, the same circuit acts differently depending on the order of the genetic parts, the selected host, or the used environment.

Furthermore, considering the unpredictability of biological interactions, additional problems arise. Common failure modes include transcriptional read-through. Transcriptional read-through, shown in Figure 3.6 (a), happens if a terminator is not strong enough to knock off the RNAP and therefore fails to end transcription [35, 53]. In that case, downstream sequences that should be regulated otherwise get transcribed and translated manipulating the behavior of the circuit.

Combining two promoters to a tandem promoter can result in another phenomenon called roadblocking [173, 175] shown in Figure 3.6 (b). Roadblocking occurs when the downstream promoter blocks the transcription of the upstream promoter decreasing the upstream promoter's efficiency.

Signal mismatch occurs if the selected genetic part does not produce the required output to activate or repress the next genetic part [173, 219, 230]. Signal mismatch, indicated by the protein with the dotted repression arrow in Figure 3.6 (c) can result in a decreased dynamic range of the circuit or even in the loss of its functionality [35].

Designing circuits gets further complicated since genetic circuits contrary to electronic circuits do not have physical wiring of the components allowing the signal carriers of genetic circuits to move

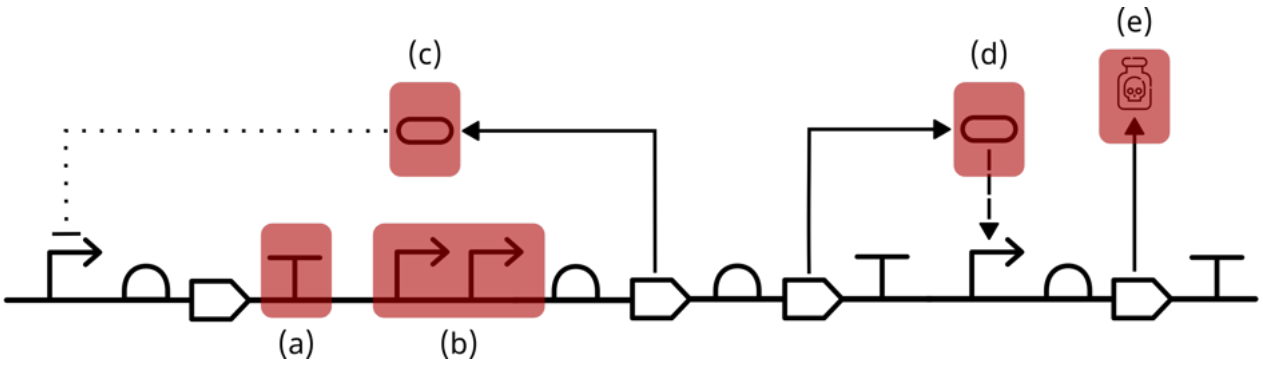


Figure 3.6: Overview of genetic circuit failure modes. (a) Shows transcriptional read-through. If a terminator is not strong enough to end transcription, the downstream TU is also transcript. (b) Shows roadblocking of tandem promoters. Here, the second promoter can block the binding site of the upstream promoter decreasing its efficiency. (c) Shows signal mismatch. Here, the output of one CDS does not produce enough molecules to repress the activity of another promoter. (d) Shows missing orthogonally. The expressed protein should not interact with the genetic sequence, yet it activates another promoter. Finally, (e) Shows the expression of a protein that in large quantities is toxic to the host.

freely within the cell or even diffuse out of the cell's membrane. Therefore, the output of one genetic part can have an unintended influence on other genetic parts or the host genome due to crosstalk, if there is no orthogonality between parts [57, 173, 219]. Figure 3.6 (d) indicates an unwanted activation of a downstream promoter by the dashed activation arrow. Crosstalk becomes a bigger problem with increasing circuit size, limiting the scalability of genetic design leading to more effort in characterizing unique signal carriers [219, 211, 160].

Finally, considering the circuit as part of a host cell, while single transcription factors can be non-toxic, their combination can result in strong toxicity killing the host cell [35] as shown in Figure 3.6 (e). Additionally, challenges arise due to the interaction of the circuit with the host organism [43, 7]. The proper functioning of a genetic circuit requires the use of host resources, for example, its transcription and translation machinery. Genetic circuits with increasing size can quickly overload a host organism, resulting in a decreased growth rate or even apoptosis [35, 69]. The sharing of resources between the host and the circuit further results in a delay in the circuit's activity [148, 55]. Recent work presents laboratory procedures [93, 45] and computational

models [92] to evaluate the burden on the cell. However, predicting the burden of a genetic circuit on its host is difficult since it varies between different hosts and different environmental conditions. Testing and measuring a circuit's performance faces its own challenges since there is no standardized procedure or protocol that allows a direct comparison between circuits. Lastly, the changes to the genetic circuit due to homologous recombination and mutations caused by natural evolution [44] can render a genetic circuit nonfunctional [35, 53]. Homologous recombination cuts out DNA between repeated sequences and increases with the toxicity of the circuit [35, 206].

Transcriptional read through or unwanted interactions between molecules (orthogonal) and/or functional parts at part junctions can all lead to such a behavior. Systems with wrong steady-state behavior are unusable in real world applications and need improved robustness.

3.2.2 Genetic Circuit Logic-Related Failures

A Boolean function is implemented through a combination of various logic gates. However, constructing a function can occur in multiple ways, employing alternative logic implementations consisting of different gate combinations. The choice of logic gates, however, impacts the signal path's length before reaching the output. For instance, in circuit 0xF6 illustrated in Figure 3.1 (b), the Ara signal only needs to traverse one genetic NOT gate before reaching the final OR gate. On the other hand, the IPTG signal must pass through both a NOT gate and a NOR gate before reaching the final OR gate. This race condition introduces a potential hazard in the circuit. This is particularly relevant for genetic circuits, where each gate comprises different genetic parts with varying reaction speeds, further influencing the time it takes for a signal to traverse the various logic levels.

These hazards are solely dependent on the logic and are therefore termed *logic hazards.* However, adjusting the logic allows the avoidance of such hazards and, consequently, the failures associated with them [165]. Designing a circuit without logic hazards necessitates initially employing hazard-free logic synthesis to achieve a two-level logic design free from hazards. Subsequently, this design can be refined to incorporate only multi-level logic transformations using hazard-free logic

design methods [165]. Modifying the logic of a circuit does not alter its function; the circuit still exhibits the same behavior. However, relying solely on these hazard-free logic design methods may result in a final layout containing more logic gates and redundancy. Although the final circuit will be free of logic hazards, it might respond more slowly or become too large to be managed by a single cell without compromising its host. These aspects must be taken into account before opting for a logic hazard-free design.

3.2.3 Genetic Circuit Function-Related Failures

Computational analysis [81] disclosed that the glitch observed in the laboratory for circuit 0x8E (see Figure 3.4) is attributed to a *function hazard.* Function hazard glitches occur when multiple input changes happen simultaneously, leading to an incorrect temporary output. Unlike logic hazards, function hazards are unavoidable as they arise from the circuit's inherent function.

As mentioned earlier, a circuit's Boolean function can be represented in a truth table. An alternative method is a Karnaugh map [113]. Figure 3.7 displays the Karnaugh map of circuit 0x8E presented in Figure 3.3. In the Karnaugh map, the rows indicate the presence of Ara, and the columns indicate the presence of IPTG and aTc, respectively. The glitch observed in the laboratory, in this input configuration, occurred during the transition from (0, 0, 0) to (1, 1, 0). As shown in the Karnaugh map, there are two ways the transition can occur. The circuit can either detect the input change of Ara first, thus transitioning from (0, 0, 0) to (1, 0, 0) to (1, 1, 0), or it can detect IPTG first. In that case, the circuit transitions from (0, 0, 0) to (0, 1, 0) to (1, 1, 0). If the circuit follows the green arrow, detecting Ara first, the output stays high throughout the transition since the transition state (1, 0, 0) is also high. However, if the circuit detects IPTG first, the red arrow, the glitch occurs since the transition state (0, 1, 0) has a low output.

Since functions hazards are a property of the function being implemented and not the logic implementing it, it is impossible to circumvent this function hazard for the specified input transition without altering the function itself. Figure 3.8 illustrates the simulation results of an ODE model corresponding to the observed input transition in the laboratory. The x-axis represents time, while

Ara \ IPTG aTc	0 0	1 0	0 1	1 1
0	1	0	0	0
1	1	1	1	0

Figure 3.7: Karnaugh map of circuit 0x8E. The rows indicate the presence of Ara, and the columns indicate the presence of IPTG and aTc, respectively. The individual cells represent the output of the circuit, with a high output denoted by 1 and a low output by 0. When the circuit transitions from (0, 0, 0) to (1, 1, 0), it can follow two paths: either through state (1, 0, 0), responding to Ara first (green arrow), or through state (0, 1, 0), responding to IPTG first (red arrow). State (1, 0, 0) yields a high output, preventing a glitch, whereas state (0, 1, 0) results in a low output, causing a glitch as the output briefly turns off.

the y-axis represents the output fluorescence. As depicted in the figure, the ODE model also reproduces the occurrence of the glitch.

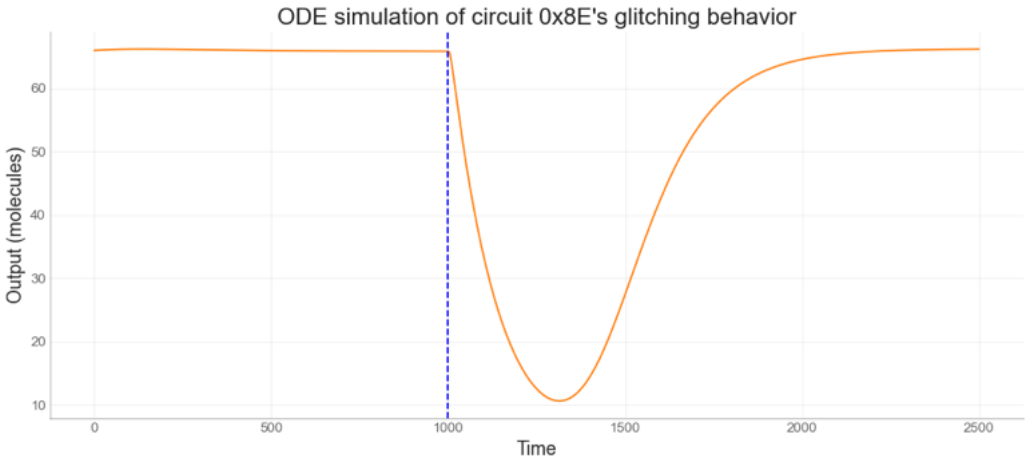


Figure 3.8: ODE simulation of circuit 0x8E's glitch observed in the laboratory. The x-axis shows the time and the y-axis the output fluorescence. The output of the circuit is shown as the orange graph. The blue dashed line indicates the input change from Ara, IPTG, aTc = (0, 0, 0) to (1, 1, 0). As observed in the laboratory, output production briefly turns off before recovering to a high state.

Although it is impossible to completely eliminate the hazards associated with such input transitions, modifications to the circuit can alter the probability of the glitch manifesting. If the circuit has an increased probability of detecting the correct transition state (as indicated by the green

arrow in Figure 3.7), it becomes less prone to exhibiting the undesired behavior. Alternatively, users can manually apply one of the inputs with a delay, compelling the circuit to follow the correct path. However, to assess the likelihood of the circuit correctly detecting the right input first or whether it requires manual intervention to ensure the correct direction, defined properties are essential for standardizing the assessment of failures.

3.3 Properties for Genetic Circuit Analysis

Efficiently testing defined properties constitutes a fundamental aspect of workflows in various engineering disciplines. Mechanical engineers, for instance, employ computational analysis for material testing, as exemplified in studies such as Zohdi et al.'s work [237]. In electrical engineering, asynchronous circuits undergo testing for delay faults, particularly since they lack synchronization with a clock, unlike their synchronous counterparts [118]. Additionally, computational simulation plays a pivotal role in anticipating design and manufacturing faults in transistor-based circuits within the realm of electrical engineering [187].

3.3.1 Mathematical Definition of Properties

In the field of synthetic biology, various analysis methods are employed to assess the properties of diverse biological species over time. Specifically, more intricate genetic circuits undergo scrutiny to ensure their proper functionality, a matter of considerable real-world significance. Property 3.1 serves as an illustrative example, detailing the criterion for a genetic digital logic circuit to maintain a low output during an input transition from one state to another, both characterized by a low output. Any deviation from this behavior, such as the circuit turning on during the transition, signifies a malfunction in its intended operation [175]. The property is written in *continuous stochastic logic* (CSL), a formal language used for specifying temporal properties of continuous-time stochastic systems [8]. CSL contains *temporal operators* like the *future* (F) operator indicating that a property must eventually become true, the *globally* (G) operator specifying that a property must remain true indefinitely, and the *until* (U) operator denoting that one property must remain true until another

property becomes true. Additional insights into this topic can be found in works such as those by Buecherl et al. [39] and Fontanarrosa et al. [81].

$$\mathrm{P} = ?\ [\text{true}\ U\ [0, \text{time}]\ (\text{species} \leq \text{threshold})] \tag{3.1}$$

The property denoted as "P" employs a probabilistic assessment, denoted by the "?" symbol, to determine the likelihood of the property being true. The U operator establishes a condition where the initial state must persist as true until the moment the second condition is met. The interval notation $[0, \text{time}]$ defines a time window during which the condition following the U operator must remain true. Lastly, (species $\leq$ threshold) represents the condition that must eventually be fulfilled within the specified time frame.

Analyzing dynamic glitches, as depicted in Figure 3.5 (b), requires a different approach. The logical condition defined in Equation 3.2 evaluates whether the circuit achieves its expected output (*threshold*) by a given time (t). Such conditions are employed in programming or mathematical modeling to describe system behaviors. A transition without dynamic hazards reaches its steady state more rapidly than one exhibiting a dynamic glitch.

$$(\text{species} \leq \text{threshold})\ ||\ (\text{time} > t) \tag{3.2}$$

Consequently, the constraint examines whether the species stays below or equal to the threshold (or off) before the specified time. In the presence of a dynamic glitch, the circuit will take an extended duration to turn off and will not dip below the threshold by the designated time.

3.3.2 Time and Threshold Parameter Evaluation

Following the definition of properties, the next step involves identifying the parameters used. To determine the thresholds for on and off signals, ODE analysis was conducted on a NOT gate [39]. In genetic design, a NOT gate is implemented using a repressible promoter, a ribosome binding site, a coding sequence, and a terminator. The choice of the NOT gate for analysis was made to determine the threshold at which the input needs to be to repress the output of the downstream promoter, and vice versa.

Figure 3.9 illustrates the results of the ODE analysis of a genetic NOT gate. The x-axis represents the number of input molecules, while the y-axis represents the number of output molecules.

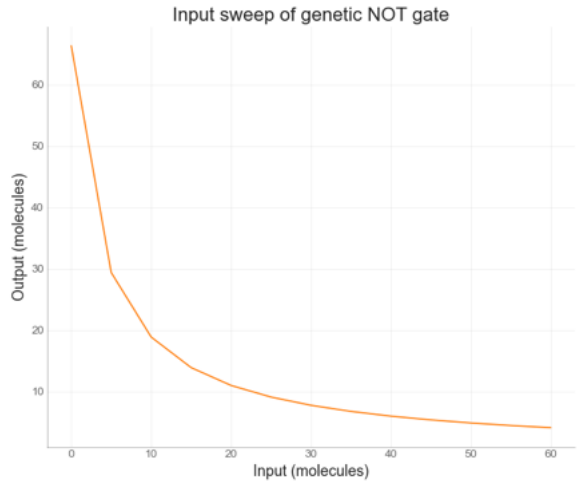


Figure 3.9: Input sweep of genetic NOT gate. The graph depicts the molecular output of a logical NOT gate against varying input molecule counts. It illustrates that an input signal of ten molecules causes the output to decrease by over 40 molecules. In contrast, a high input signal of 30 molecules and above leads to a low output of under ten molecules. Consequently, a high signal is defined as being above 30 molecules, while a low signal is set to be below ten.

Figure 3.9 illustrates that an input of ten molecules leads to a reduction of over 40 molecules in the output. This observation implies that if the output is expected to stay low, it becomes critical if the molecular count briefly exceeds ten. Consequently, the threshold for a low signal was set to ten, as a higher threshold would overly repress a downstream promoter.

Conversely, the threshold for a high signal was established at 30 molecules. Even with a high signal of 30 molecules, the output remains low, with fewer than ten molecules, effectively repressing the promoter. This approach, although simplified compared to the more sophisticated method proposed by Baig et al. in [12], is grounded in the fundamental practice of clearly defining what constitutes a failure.

To determine the necessary time, ODE analysis can be conducted to observe when a system attains steady state. In Figure 3.10, the orange plot illustrates a genetic circuit transitioning to its on state after receiving the input. While the simulation has a time limit set to 2100, the circuit actually reaches steady state at around 1000 time units (indicated by the dashed blue line). This time point is chosen to facilitate efficient simulation and prevent prolonged runtimes.

To analyze dynamic glitches, a non-dynamic hazard transition, illustrated in orange, must be contrasted with a transition featuring a dynamic glitch, shown in purple. The green dashed

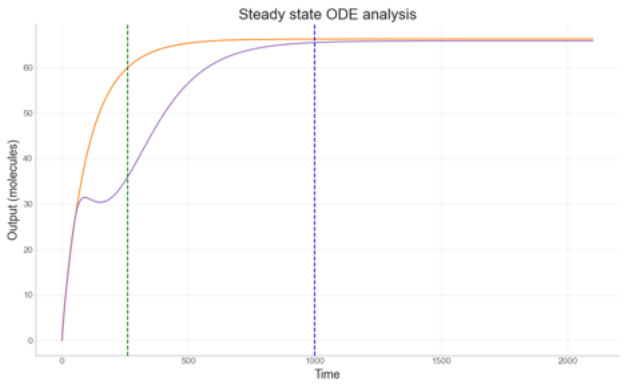


Figure 3.10: Steady-state ODE analysis. The graph depicts the output of a genetic circuit transitioning from the absence of inducers to two distinct on states. The orange transition functions correctly, while the purple transition exhibits a dynamic glitch. The green line represents the threshold for detecting a dynamic glitch, as the correctly behaving transition has already reached 60 molecules, whereas the glitch transition remains below 40 molecules due to the introduced delay. The blue dashed line signifies that the circuit reached its steady state by time point 1000.

line indicates that after 260 time points, the properly functioning state has already reached 60 molecules, whereas the transition with the glitch has not surpassed 40 molecules yet. Therefore, 260 is considered a suitable cutoff for identifying dynamic glitches for this circuit. Since the response time for genetic circuits varies, it needs to be defined for each circuit individually. For subsequent analyses, however, the threshold for the signals is set to either ten or 30, the *simulation time* to 1000, and the threshold time for dynamic glitches is determined based on the speed of a correctly functioning transition of the given circuit.

3.4 Summary

This chapter introduces an examination of steady-state failure using the genetic circuit 0xF6 [175] as an example. The circuit is particularly intriguing due to an anomaly in one of its anticipated high states, which exhibited a comparatively weaker on signal compared to its other high signals. Subsequently, the discussion transitions to transient failures, highlighting circuit 0x8E [175], another circuit realized in practical settings. These circuits assume significance throughout the dissertation as their experimental failures provide valuable case studies for analysis. Following the delineation of steady-state and transient failure behaviors, the chapter delves into potential explanations for these anomalies, spanning cellular-level issues, circuit logic, and function inadequacies. Ultimately,

the chapter concludes by defining mathematical properties that enable computational analysis of genetic circuit models in the subsequent chapters, offering insights into observed failure behaviors in real-life scenarios.

"Faithless is he that says farewell when the road darkens."

- Gimli, son of Glóin

4

Genetic Circuit Modeling and Compuational Analysis

After identifying properties for simulating and verifying genetic circuit models, this chapter delves into the computational methods used to analyze and simulate the identified failure modes, predicting a genetic circuit's behavior *in silico*. Previous studies have employed ODE analysis, SSAs, and *stochastic model checking* [81, 39]. Each analysis method has its own advantages. ODE analysis offers examination of a genetic circuit's average behavior, while stochastic simulation can encompass non-deterministic behavior. Verification through stochastic model checking provides the accurate, true probability of failure but quickly becomes impractical for larger designs [39, 137, 139]. Additionally, in considering different analysis methods, the impact of varying levels of abstraction must be taken into account, with more detailed models allowing for a more accurate prediction at

the expense of increased computational complexity.

While synthetic biologists aspire to engineer genetic circuits analogous to electrical circuits, the inherent unpredictability of biology poses a significant challenge [133, 126]. Computational analysis of genetic circuit models empowers researchers to study their design and predict the behavior of a circuit *in silico*. This approach enables designers to leverage analysis results for circuit debugging before committing to time-intensive laboratory experiments, akin to the process of debugging software prior to application deployment. Undetected, unintentional circuit failures can have drastic effects on the host organism, jeopardizing the overall desired function or purpose of the system. Therefore, the modeling and analysis of highly complex genetic circuits are crucial for designing robust genetic circuits [112, 46]. This analysis serves as a tool to uncover critical errors in the design or model itself.

In Section 4.1, this chapter begins by exploring various analysis methods applied to a model of circuit 0x8E [175]. The employed methods encompass ODE analysis (Section 4.1.1), SSA (Section 4.1.2), and stochastic model verification (Section 4.1.3). Section 4.2 presents two additional design implementations of the previously introduced circuit 0x8E. Coupled with the properties defined in Chapter 3 and the introduced analysis methods, Section 4.3 scrutinizes the three different models concerning their glitching behavior for static input transitions with function hazards (Section 4.3.1), without function hazards (Section 4.3.3), and their probability of steady-state failure (Section 4.3.4).

4.1 Model Analysis Methods

In this study, three different analysis methods have been employed. First, the traditional ODE model analysis is introduced, detailing the *Euler* and *Runge-Kutta* simulation algorithms. Following that, to simulate probability functions, *Gillespie's* SSA is introduced. The final method described is stochastic model verification, presented along with an overview of the STAMINA algorithm [188], which enables the application of the analysis method in this work.

4.1.1 Ordinary Differential Simulation Algorithms

A set of differential equations, as introduced in Chapter 2, is typically challenging to solve and nearly impossible for large, complex models [167]. Therefore, numerical simulation methods are employed to analyze a system's behavior. The goal of simulation is to approximate the function over time, given an initial state of the system. In this work, this initial state describes the number of each different species in the system at time $t = 0$. Simulation then approximates the change in the number of the different species over time. Problems of this kind are called *initial value problems*.

Simulation methods like Euler and Runge-Kutta can be employed to simulate the behavior of a provided ODE system [167]. The forward Euler method determines the rate of change for each species at the initial time step t_0. The value of the species is then updated until the next time step t_1 is reached. At this point, the rate of change is recalculated, and the process is repeated until the time limit is reached. Euler's method only uses information from the current state and is therefore an *explicit method*. In contrast, *implicit methods* do not determine the rate of change based on the current state, but instead by calculating the rate that would have taken you to your current state in a time step Δt. One example of such a method is the backward Euler method.

Moving beyond Euler's method, Runge-Kutta offers a more sophisticated approach to simulating a system's behavior. Unlike Euler's method, which assumes a constant rate of change throughout each time step, Runge-Kutta calculates the rate of change at the midpoint of the time interval Δt between t_0 and t_1. Based on the rates at t_0, the midpoint is calculated, and the rates are then recalculated. The updated rates are used until t_1 is reached. This method is also known as the *midpoint method* or *second-order* Runge-Kutta. Further refinement can be achieved by adding more points and fine-tuning the time interval Δt.

ODEs are an effective method for promptly assessing a system's behavior. However, ODE analysis assumes the system to be deterministic, and values of molecules to be continuous. The same simulation yields consistent results across multiple runs. Unfortunately, genetic circuits are not deterministic, nor is the number of molecules continuous. Therefore, alternative analysis methods can be required.

4.1.2 Gillespie's Stochastic Simulation Algorithm

As an alternative to ODEs, genetic systems can be analyzed using SSAs, given the inherent stochasticity of genetic systems and the discrete nature of molecule counts. Chemical reactions occur when two (or more) molecules collide. Rather than tracking the spatial location and velocity of each molecule, reactions can be viewed as a stochastic process. This is particularly applicable to systems with low molecular counts. Genetic circuits arguably fall into this category, given that, for example, a single RNA typically codes, on average, for ten proteins [167].

The SCK models introduced in Chapter 2 are classified as *jump Markov processes* [167]. In these models, the next state of the system depends solely on the current state and is independent of past states. Additionally, state updates occur in discrete amounts. It is important to note that, in such models, it is impossible to determine the specific next state the system will be in. Instead, it is only possible to calculate the probability of reaching a given state from the current state. Mathematically, the *chemical master equation* defines a function describing the evolution of the state probabilities by considering which reactions can be fired [167].

Built upon an equivalent formulation of the chemical master equation, the SCK relies on selecting a small time step, denoted as Δt, during which the system is updated if a reaction occurs [167]. It is crucial to emphasize that Δt is chosen to ensure that at most one reaction can occur within that time frame. However, opting for a small Δt may lead to numerous time steps where no reactions take place. The Gillespie SSA [88], employed in this study, navigates through these inactive time steps during simulation, enhancing efficiency while computing the trajectory of the system's behavior. The algorithm is rooted in a joint probability density function for two random variables, τ (time to the next reaction) and μ (the index of the next reaction), given the system's state x at time t. Commencing from the initial state and time, the algorithm utilizes two random variables to determine τ and μ, identifying the reaction to be executed in the subsequent time step. Following the update of the system, the process is iterated.

In contrast to ODE analysis, SSA incurs higher computational costs [167]. While a single run of SSA is faster than an ODE run, the former necessitates multiple runs to obtain meaningful

statistics for the system. The analyses conducted in this study involved running simulations up to 100,000 times. Conversely, ODE is deterministic and, as such, requires only a single run.

4.1.3 Stochastic Model Checking

Stochastic model checking is a formal verification technique that calculates the probability of a specified event for a given stochastic system using CTMC analysis methods. Mathematically, CTMCs are defined by a *transition rate* or Q matrix. In contrast to a transition probability matrix, the entries of the Q matrix not only specify the likelihood of each next state but also the time the system takes to transition to that state [167]. This matrix can then be employed to formulate rate equations for steady-state and transient analyses.

As described in Chapter 2, the state space of CTMCs of genetic circuits is infinite and, therefore, enumerating the entire state space is impossible with finite resources. However, the software tool STAMINA, briefly introduced in Chapter 2, transforms the state space of an infinite model into a finite state space. It does so by traversing the state space and pruning states where the reachability value falls below a user-specified threshold κ. The method is illustrated in Figure 4.1, with the threshold set to $\kappa = 0.001$. The illustrated example system comprises four reactions, as depicted in Equations 4.1, with the initial state presented in Equations 4.2.

$$R_1: \; P \xrightarrow{k_p} P + S_1, \qquad R_2: \; S_1 \xrightarrow{k_t} S_2, \qquad R_3: \; S_1 \xrightarrow{k_d} \emptyset, \qquad R_4: \; S_2 \xrightarrow{k_d} \emptyset \tag{4.1}$$

$$\mathrm{S_1(t_0) = 40} \qquad \mathrm{S_2(t_0) = 0} \qquad \mathrm{P(t_0) = 1} \tag{4.2}$$

STAMINA begins its exploration from the initial state $S_1 = 40$ and $S_2 = 0$ (see Figure 4.1 (a)), identifying three reachable states from this initial condition (see Figure 4.1 (b)). These states become accessible through the firing of either reaction R_1, R_2, or R_3. Since all states possess a reachability value greater than κ, they are included in the exploration process. Subsequently,

STAMINA iterates over the newly discovered states, identifying those with a reachability value exceeding κ (see Figure 4.1 (c)). Ultimately, additional states that do not meet the criteria of having a reachability value greater than κ are truncated.

The resulting finite state space can now be used for stochastic model checking. After STAMINA has completed the state space generation and truncation, it passes the finite state space to established tools like PRISM [124] and STORM [64] for Markov chain analysis. However, because certain parts of the state space have been removed, there is a probability leakage. As a result, the system's analysis yields an under-calculated probability. To address this, the probability of transitions from truncated states is redirected to an absorbing state to capture the omitted probability (see Figure 4.1 (d)). Consequently, stochastic model checking is performed on both the state space without and with the absorbing state, producing two probabilities. The true probability of the event lies within the probability window provided by STAMINA.

Stochastic model checking has the advantage of providing the actual probability of an event of interest occurring, unlike simulation results. However, given the size of the models, there is a computational cost associated with running stochastic model checking. Especially larger, more complex models require computational resources that are not always accessible.

4.2 Alternative Designs of Circuit 0x8E

Chapter 3 introduces circuit 0x8E [175]. This circuit lends itself as a great case study since it is built out of well-defined parts and has been proven to exhibit glitching behavior in the laboratory, as shown in Figure 3.4 and its ODE simulation in Figure 3.8. In previous work, Fontanarrosa et al. investigated the cause of this glitching behavior and demonstrated that an in-depth analysis of hazards is necessary to guide the design towards robust genetic circuits [81]. Using dynamic ODE models, their work identified all input transitions that result in glitching behavior of the genetic circuit. Additionally, they presented two modified implementations of the circuit's function to reduce the magnitude of the circuit's typical glitching behavior. Following this work, this chapter evaluates if these two modified versions of the circuit actually reduce the probability of the glitching

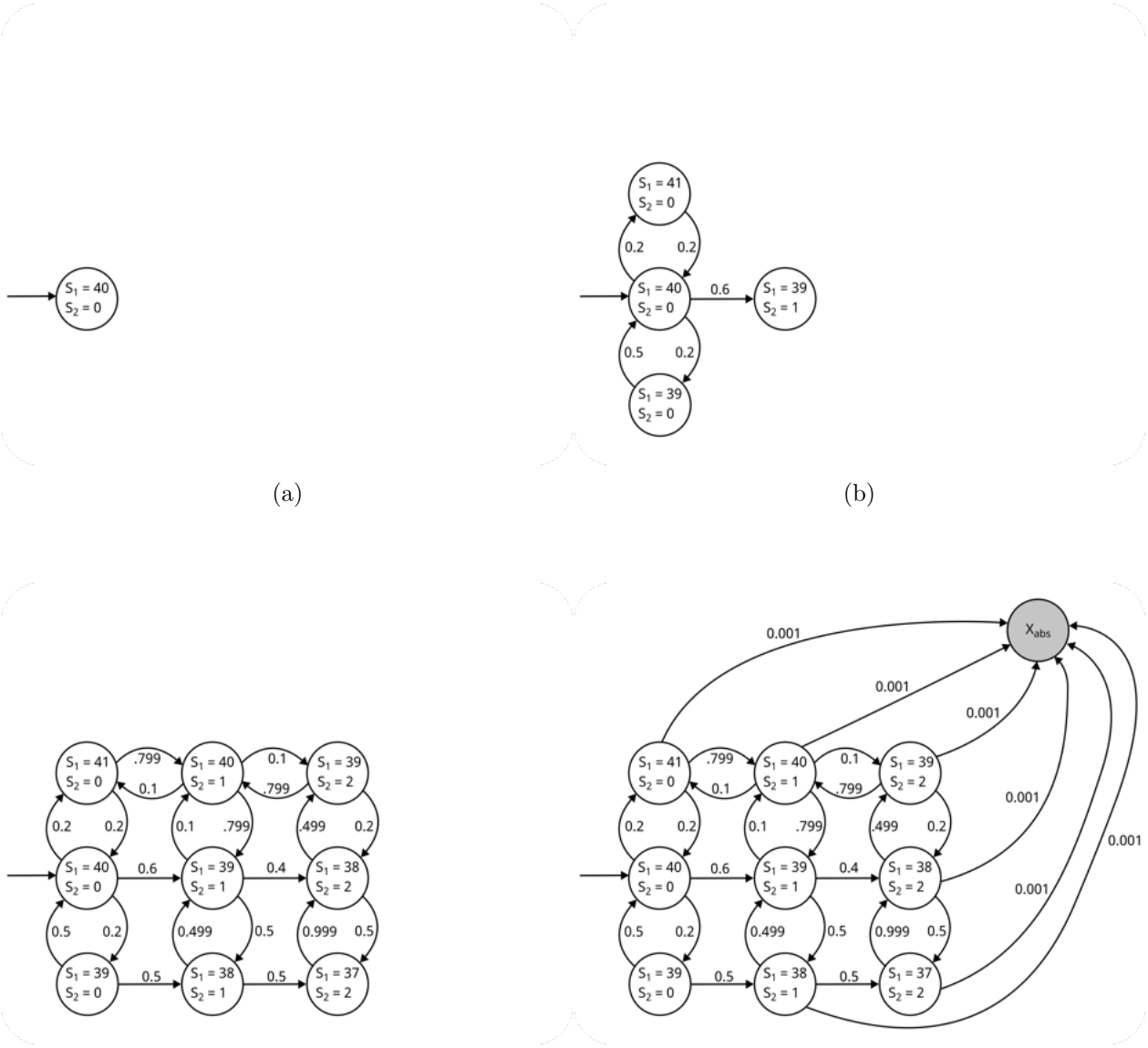


Figure 4.1: STAMINA state space truncation example with a threshold set to $\kappa = 0.001$. (a) STAMINA begins in the initial state $S_1 = 40$ and $S_2 = 0$. Since all outgoing reachability values are greater than κ, it explores the reachable states from the initial state. (b) STAMINA iterates over the newly discovered states, identifying those with a reachability value exceeding κ, continuing to add states to the state space. (c) State space exploration is terminated, as all reachable states not yet part of the state space exhibit reachability values below κ. (d) Probability of states with truncated transitions is redirected to an absorbing state to prevent probability leakage.

behavior occurring.

To answer the stated question, the analysis methods described are run on models of the three implementations presented by Fontanarrosa et al. [81]. Figure 4.2 (a) shows the original implementation of the circuit [175]. Figure 4.2 (b) and (c) show two additional implementations of the circuit presented in [81]. All three circuits implement the same logic function represented by the truth table shown in Figure 3.3 (a), but use different networks of logic gates. This is especially noticeable when looking at the circuit diagram shown in Figure 4.2 (b). Here, the circuit has two added NOT gates that add an extra delay to the IPTG pathway. Two NOT gates flip the signal twice, therefore not altering the signal overall. The added delay is supposed to impact the glitching behavior of the circuit positively, especially the transition that was observed to glitch experimentally.

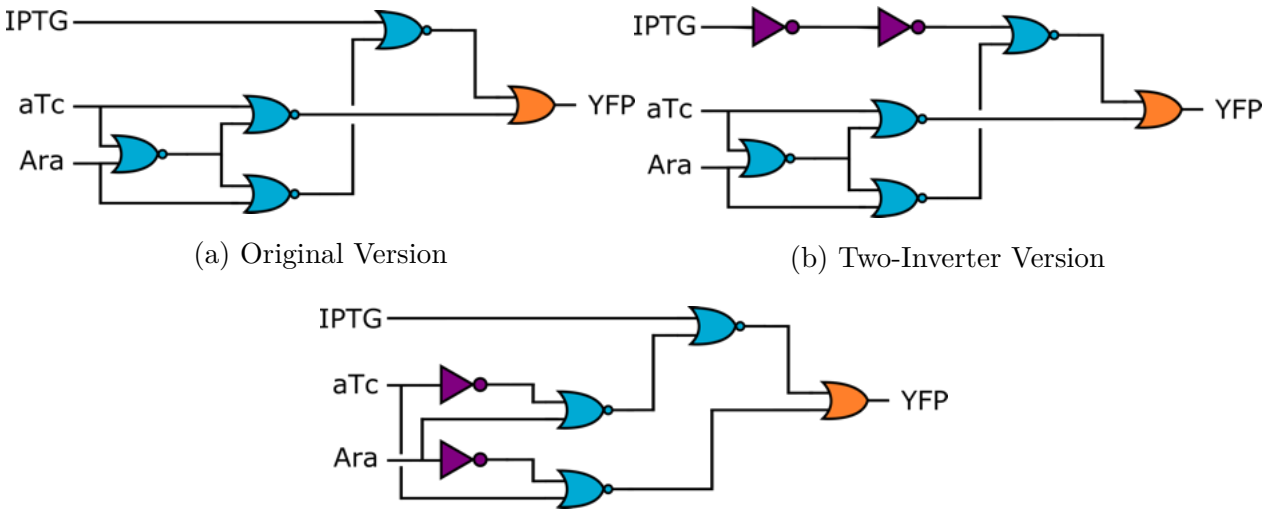
The third iteration of the circuit, presented in Figure 4.2 (c), represents a logic-hazard-free version. As explained in Section 3.2.2, logic hazards stem from the circuit's logic rather than its function. Unlike function hazards, logic hazards can be circumvented through hazard-free logic synthesis and hazard-preserving optimizations while simplifying the circuit's function.

Computational models of the original and modified versions were generated using the GDA tool iBioSim [226]. Notably, to streamline the circuit's complexity, the input molecules IPTG, aTc, and Ara were replaced with their corresponding internal molecules and complexes. For instance, instead of individually modeling Ara and its complex formation AraAraC, only the complex AraAraC was considered. Furthermore, protein degradation, initially modeled in increments of one, was adjusted to occur in increments of ten, with a tenfold reduction in reaction propensity.

For stochastic model verification, the designs were exported as SBML models [115] and then translated into PRISM models [124] using PRISM's SBML-to-PRISM translator. As mentioned earlier, additional analysis was conducted using STAMINA [171].

4.3 Computational Analysis of Circuit 0x8E

Initially, the circuit underwent analysis using ODE simulation to pinpoint all twelve transitions with function hazards [81]. The results are depicted in Figure 4.3. In the graph, the blue plot



(a) Original Version

(b) Two-Inverter Version

(c) Logic-Hazard-Free Version

Figure 4.2: Three distinct logic configurations for circuit 0x8E. The inducer molecules—IPTG, aTc, and Ara—act on the circuit, with the output being YFP. The OR gate is symbolized by , and the NOR gates by . In (a), the original layout is shown, as published in [175]. Version (b) introduces two additional NOT gates to the IPTG pathway, creating an extra delay. The NOT gates are denoted by . Version (c) represents a logic-hazard-free implementation of the circuit's function. Further information on versions (b) and (c) can be found in [81].

illustrates the molecular count output of circuit 0x8E over time, while the lower section displays the input transitions. The three colored lines represent IPTG, aTc, and Ara, with a thin line indicating an off signal and a box denoting an on signal. The red shaded area highlights the observed glitch in the output production. As per the insights from Chapter 3, the first five glitches are identified as *static 1→1 glitches*, while the last four are *static 0→0 glitches*. It is evident that not all function hazards result in glitching behavior. Some transitions, like the one marked with a green box, do not exhibit glitches despite having a function hazard.

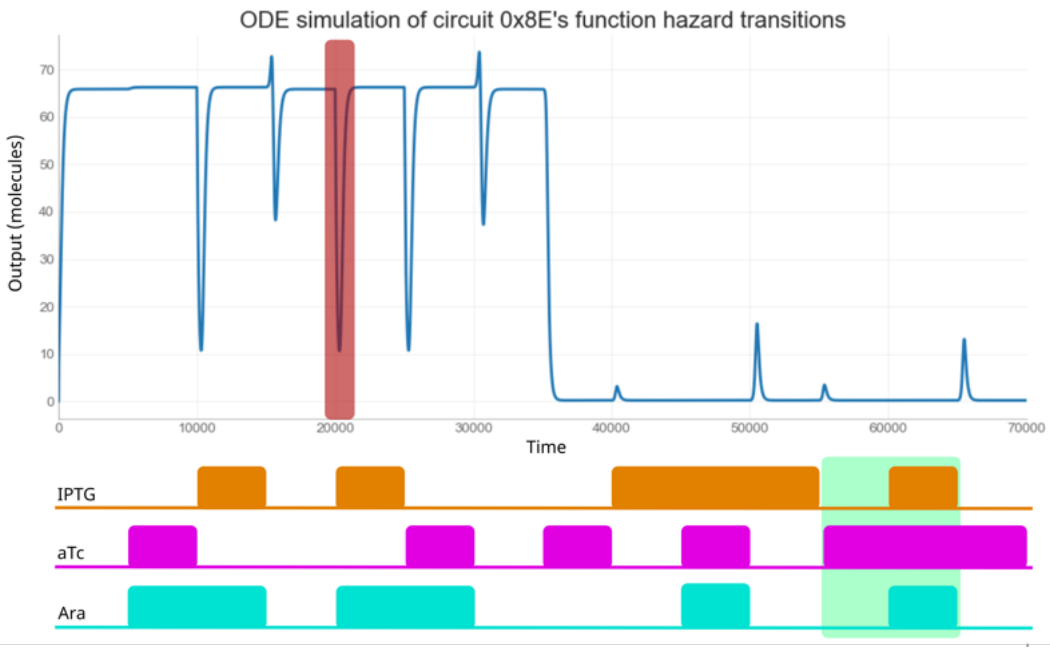


Figure 4.3: ODE simulation of circuit 0x8E, depicting molecular counts over time in blue. The red shaded area highlights the glitch observed in the laboratory during the transition from IPTG, aTc, Ara = (0, 0, 0) to (1, 0, 1). The bottom of the figure illustrates the alterations in various input combinations. Notably, the green shaded input transition, representing the transition from (0, 1, 0) to (1, 1, 1), has a function hazard but does not display glitching behavior during simulation.

In the subsequent analysis, all three circuit models were scrutinized using both stochastic simulation and stochastic model checking to ascertain the likelihood of glitching behavior of each identified transition. This dual approach was employed not only for result validation but also to demonstrate the feasibility of employing stochastic model checking for genetic circuits.

Initially, this section conducts a detailed analysis of the twelve input transitions with the

identified function hazards. Furthering the analysis, two additional failure modes are explored. Secondly, input transitions of the circuit without hazards are investigated, focusing on the probability of failure primarily attributed to noise. Lastly, the probability of steady-state failure of the circuit is also calculated.

For the analysis, molecular counts are defined as follows: a high input signal equals 60 molecules. Consistent with the previous analysis in Chapter 3, the output constraint is set at ten molecules for *static 0→0 hazards*. In other words, if the output is expected to remain low during an input transition, the run is considered a glitch if YFP exceeds the threshold of ten molecules. Conversely, for input transitions where the output is intended to stay high, the run is deemed a glitch if YFP falls below a molecule count of 30.

The stochastic simulation is based on 100,000 runs of Gillespie's algorithm [88]. The choice of 100,000 runs aims to achieve a reasonable level of confidence in the results. In statistical terms, the 95 percent confidence interval is defined by $\bar{X} \pm 1.96\frac{\sigma}{\sqrt{n}}$, where $\bar{X}$ represents the mean of the random variable, σ is the standard deviation of the population, and n is the number of runs. As a result, the confidence interval becomes more precise with an increasing number of runs. After executing the stochastic simulations in iBioSim, the models were exported as an SBML file and then converted to a PRISM model using the SBML-to-PRISM translator integrated into PRISM [124]. Finally, the PRISM model underwent stochastic model checking in STAMINA.

4.3.1 Probability of Glitching Behavior of Input Transitions with Function Hazards

Figure 4.4 presents the results of both stochastic simulation and model verification for the original design. In Figure 4.4 (a), the probability of a glitch occurring over time is depicted. The graph is constructed by conducting 100,000 Gillespie runs and terminating each run at the point of violating the property defined in Chapter 3 Equation 3.1. For instance, examining the blue curve reveals that for the input transition IPTG, aTc, Ara = (0, 0, 0) to (1, 0, 1), approximately 99 percent of the runs failed overall. This corresponds to the glitch observed in the laboratory, as illustrated in Figure 3.4 [175]. Beyond 300 time units, over 95 percent of the transitions have failed,

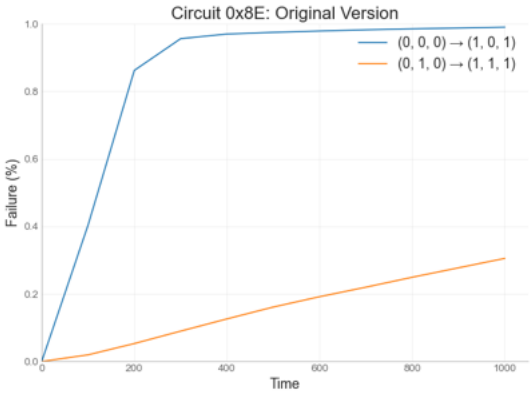
offering an explanation for the detection of the glitch during the time course experiment.

In contrast, for the input transition IPTG, aTc, Ara = (0, 1, 0) to (1, 1, 1), only 31 percent of runs failed, as indicated by the orange graph in Figure 4.4. The identical transition is highlighted in the green shaded area of Figure 4.3. Despite having a function hazard, the circuit does not manifest any glitching behavior during simulation which is supported by the lower probability obtained by stochastic simulation.

The table in Figure 4.4 (b) presents glitch probabilities for all input transitions with known glitching behavior in the original circuit implementation. The three columns in the table represent the input transition, the glitch probability simulated in iBioSim, and the glitch probability calculated with STAMINA. For instance, in the case of the input transition IPTG, aTc, Ara = (0, 0, 0) to (1, 0, 1), iBioSim predicts a 99 percent failure rate, aligning with the results from stochastic model verification indicating a probability between 98 percent and 99 percent.

It should be noted that circuits can also experience failures over time unrelated to hazards. This is primarily attributed to noise, where a random set of reactions induces an erroneous change in the YFP level. For instance, consider the blue graph in Figure 4.4; the graph exhibits a sharp increase to over 95 percent in the first 300 time steps, indicative of the glitching behavior of the circuit. Subsequently, the probability increases linearly with a small slope, representing the influence of noise. In contrast, when compared to the transition illustrated by the orange curve, there is no steep increase in probability. Instead, it steadily rises linearly, suggesting that the circuit does not manifest a glitch but rather fails due to noise. If the simulation were to run indefinitely, the probability would eventually reach 100 percent due to the inherent noisy behavior, leading to a system failure. To distinguish these types of failures from those attributable to hazards, the analysis was restricted to the first 1000 seconds.

Figure 4.5 illustrates the probability of a glitch occurring over time through 100,000 Gillespie runs for the two-inverter version of the circuit, as depicted in Figure 4.2 (b). The same input transitions analyzed for the original implementation in Figure 4.4 were considered for this modified circuit. The corresponding results from the stochastic analysis using STAMINA are presented in



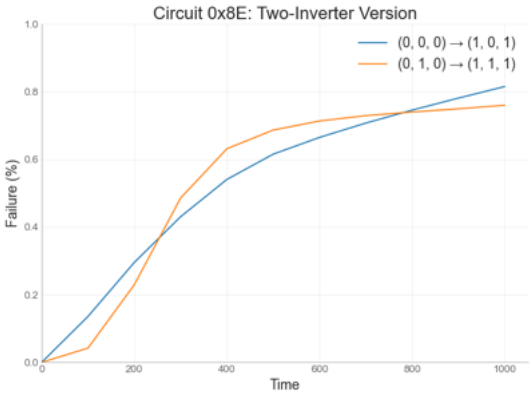
(a)

Input Transition	iBioSim	STAMINA
$(0,1,0) \rightarrow (1,1,1)$	0.30544	MIR
$(0,1,0) \rightarrow (1,0,0)$	0.72446	0.7134 - 0.7435
$(1,1,1) \rightarrow (1,0,0)$	0.92451	0.9108 - 0.9969
$(1,1,1) \rightarrow (0,1,0)$	0.90585	0.8969 - 0.9819
$(1,0,0) \rightarrow (0,1,0)$	0.76282	0.7673 - 0.7884
$(1,0,0) \rightarrow (1,1,1)$	0.30281	0.3033 - 0.3386
$(0,1,1) \rightarrow (1,0,1)$	0.98978	0.9882 - 0.9919
$(0,0,0) \rightarrow (0,1,1)$	0.82713	0.8226 - 0.8312
$(0,0,0) \rightarrow (1,0,1)$	0.99075	0.9882 - 0.9935
$(1,0,1) \rightarrow (0,1,1)$	0.98918	0.9876 - 0.9926
$(0,1,1) \rightarrow (0,0,0)$	0.86400	0.8566 - 0.8587
$(1,0,1) \rightarrow (0,0,0)$	0.86409	0.8607 - 0.8706

(b)

Figure 4.4: Glitch probabilities for input transitions of circuit 0x8E simulated using iBioSim [226] and verified in STAMINA [171]. The input order is IPTG, aTc, Ara. (a) Depicts the probability of a glitch occurring over time for input changes known to have function hazards. As selected examples, the orange curve illustrates the input transition IPTG, aTc, Ara = (0, 1, 0) to (1, 1, 1), and the blue curve represents the transition from (0, 0, 0) to (1, 0, 1). The table in (b) displays the results of the analysis for all input transitions with a function hazard. The stochastic simulation in iBioSim was based on 100,000 runs. STAMINA was run with a target probability window width of 0.1. *Max-Iterations-Reached* (MIR) indicates that the maximum iterations (default ten) were reached, indicating slow convergence.

the table in Figure 4.5 (b). Specifically, the addition of two-inverters aimed to positively influence the input transition IPTG, aTc, Ara = (0, 0, 0) to (1, 0, 1), represented by the blue curve in both Figure 4.4 and Figure 4.5.



(a)

Input Transition	iBioSim	STAMINA
(0, 1, 0) → (1, 1, 1)	0.75999	*†0.2770 - 1.0
(0, 1, 0) → (1, 0, 0)	0.79609	0.7854 - 0.8534
(1, 1, 1) → (1, 0, 0)	0.92703	0.9149 - 0.9834
(1, 1, 1) → (0, 1, 0)	0.54285	0.5381 - 0.5710
(1, 0, 0) → (0, 1, 0)	0.42337	0.4316 - 0.4529
(1, 0, 0) → (1, 1, 1)	0.32885	OOM
(0, 1, 1) → (1, 0, 1)	0.80690	*†0.3633 - 1.0
(0, 0, 0) → (0, 1, 1)	0.82647	†0.6424 - 0.9695
(0, 0, 0) → (1, 0, 1)	0.80761	*†0.4409 - 1.0
(1, 0, 1) → (0, 1, 1)	0.99970	0.9159 - 1.0
(0, 1, 1) → (0, 0, 0)	0.86894	0.8448 - 0.8766
(1, 0, 1) → (0, 0, 0)	0.95759	0.8963 - 0.9927

(b)

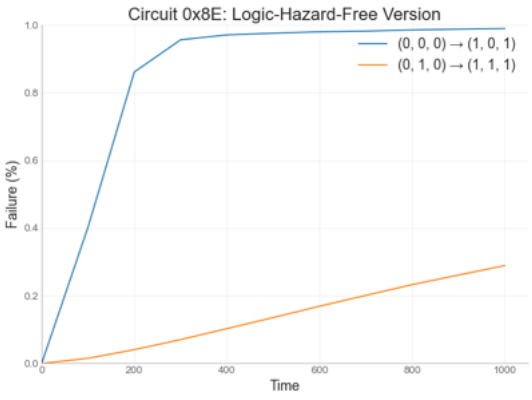
Figure 4.5: Glitch probabilities for input transitions of the two-inverter circuit 0x8E, simulated with iBioSim [226] and verified in STAMINA [171]. The order of inputs is IPTG, aTc, Ara. Panel (a) displays the probability of glitches over time for input changes known to exhibit function hazards. The orange curve represents the input transition IPTG, aTc, Ara = (0, 1, 0) to (1, 1, 1), while the blue curve represents the transition from (0, 0, 0) to (1, 0, 1), selected as examples. Panel (b) provides the results of the analysis for all input transitions with a function hazard. Stochastic simulation in iBioSim involved 100,000 runs, and STAMINA was executed with a target probability window width of 0.1 or 0.5 (†). Models marked with * failed to achieve their target probability bound due to memory constraints. *Out-of-Memory* (OOM) denotes that STAMINA couldn't achieve a probability window width of less than 0.8 before the host machine ran out of memory.

Comparing the iBioSim simulation results of the original implementation in the table in Figure 4.4 (b) with those of the modified design in Figure 4.5 (b), it is evident that the probability for the input transition (0, 0, 0) to (1, 0, 1) was improved from 99 percent to 81 percent. This improvement is visually represented by the blue curve in Figure 4.5 (a), where the initial steep increase in glitch probability is less pronounced compared to the original design. Therefore, the addition of the two-inverters to the circuit results in an improvement of 18 percent. However, it is important to note that the larger size of the circuit also imposes additional stress on the host cell [32].

In contrast, the input transition (0, 1, 0) to (1, 1, 1), represented by the orange curve in

both Figure 4.4 and 4.5 (a), exhibits a probability of less than 31 percent for the original circuit, compared to almost 76 percent for the two-inverter circuit—an increase of 45 percent. While the addition of two-inverters leads to a reduction in glitching for the blue curve, there is a trade-off with an increase for the orange curve. For other input transitions, such as (0, 1, 1) to (0, 0, 0) and (1, 1, 1) to (1, 0, 0), the impact of the two-inverters on the probabilities is negligible.

Finally, Figure 4.6 (a) presents the analysis of the same two input transitions for the logic-hazard-free version. Figure 4.6 (b) displays the results from iBioSim and STAMINA for the glitch probability of all input transitions known to exhibit glitching behavior in the logic-hazard-free implementation. It is crucial to note that the analysis presented here exclusively focuses on function hazards. However, utilizing hazard-free logic synthesis may eliminate logic hazards, consequently influencing the probability of function hazards as well.



(a)

Input Transition	iBioSim	STAMINA
$(0,1,0) \rightarrow (1,1,1)$	0.28909	0.2729 - 0.3677
$(0,1,0) \rightarrow (1,0,0)$	0.24651	0.2384 - 0.2570
$(1,1,1) \rightarrow (1,0,0)$	0.90566	0.8948 - 0.9757
$(1,1,1) \rightarrow (0,1,0)$	0.90247	0.9002 - 0.9430
$(1,0,0) \rightarrow (0,1,0)$	0.73500	0.7348 - 0.7540
$(1,0,0) \rightarrow (1,1,1)$	0.29970	0.2996 - 0.3598
$(0,1,1) \rightarrow (1,0,1)$	0.98935	0.9870 - 0.9938
$(0,0,0) \rightarrow (0,1,1)$	0.82057	0.8242 - 0.8289
$(0,0,0) \rightarrow (1,0,1)$	0.98961	0.9865 - 0.9953
$(1,0,1) \rightarrow (0,1,1)$	0.99035	*0.9817 - 0.9972
$(0,1,1) \rightarrow (0,0,0)$	0.77063	0.7653 - 0.7703
$(1,0,1) \rightarrow (0,0,0)$	0.73697	0.7345 - 0.7353

(b)

Figure 4.6: Glitch probabilities for input transitions of the logic-hazard-free circuit 0x8E, simulated with iBioSim [226] and verified in STAMINA [171]. The order of the inputs is IPTG, aTc, Ara. Panel (a) illustrates the probability of a glitch occurring over time for input changes known to have function hazards. The orange curve visualizes the input transition IPTG, aTc, Ara = (0, 1, 0) to (1, 1, 1), and the blue curve represents the transition from (0, 0, 0) to (1, 0, 1), chosen as examples. Panel (b) provides the results of the analysis for all input transitions with a function hazard. Stochastic simulation in iBioSim involved 100,000 runs, and STAMINA was executed with a target probability window width of 0.1. Models marked with * failed to achieve their target probability bound due to memory constraints.

Analysis of the logic-hazard-free version of circuit 0x8E reveals that both input transitions (0,

0, 0) to (1, 0, 1) and (0, 1, 0) to (1, 1, 1) exhibit similar behavior to the original implementation. For the blue curve, both designs show the transition glitching in 99 percent of cases. Regarding the orange curve, the logic-hazard-free version glitches in 29 percent of cases compared to 30 percent for the original implementation, indicating a negligible difference. These results align with the expectation that using hazard-free logic synthesis for circuit design eliminates only logic hazards, while function hazards remain unavoidable. However, interestingly, for three input transitions, the logic-hazard-free version outperforms the original implementation. Most notably, for the transition from (0, 1, 0) to (1, 0, 0), the original design has a glitch probability of 72 percent, while the logic-hazard-free version has a significantly reduced glitch probability of 25 percent. Similarly, for two more input transitions, (0, 1, 1) to (0, 0, 0) and (1, 0, 1) to (0, 0, 0), the logic-hazard-free version glitches in 77 percent and 74 percent, respectively, whereas the original design shows failure during those transitions in 86 percent of cases for each.

4.3.2 Comparison between SSA and Stochastic Model Verification

Comparing the results from iBioSim with those from STAMINA reveals that, in some instances, the estimated probability from iBioSim falls within or in close proximity to the bounds provided by STAMINA. For instance, considering the results for the input transition IPTG, aTc, Ara = (0, 0, 0) to (0, 1, 1) for the original circuit (Figure 4.4 (b)), iBioSim simulates a probability of 82.713 percent. STAMINA calculates a probability window of 82.26 percent to 83.12 percent for the same input transition. However, in a few cases, STAMINA fails to provide reasonably tight bounds for the calculated probabilities. For the input transition IPTG, aTc, Ara = (0, 1, 0) to (1, 1, 1) for the two-inverter circuit (Figure 4.5 (b)), STAMINA's result ranges from 27 percent to 100 percent when it runs out of memory, offering no useful information to the designer. In other instances, STAMINA provides large bounds, such as for the input transition IPTG, aTc, Ara = (1, 0, 1) to (0, 1, 1) for the two-inverter circuit (Figure 4.5 (b)), with a bound of 91.59 percent to 100 percent. Although the bound is not very tight, it still gives enough information to alert the designer that the glitch probability is likely too high.

Comparing the results from iBioSim's stochastic simulation with those from STAMINA's verification illustrates that stochastic model checking is suitable for analyzing genetic circuits and provides reliable probability bounds consistent with simulation. However, it is important to note that, in comparison to running stochastic simulations, stochastic model checking is resource-intensive in terms of both memory and runtime. All stochastic simulations and model checking tasks were executed on a computer equipped with an AMD Ryzen Threadripper 12-Core 3.5 GHz Processor and 132 GB of RAM, running Ubuntu Linux (v18.04.3). The run-times for both stochastic simulation and model verification for all input transitions and circuit versions are presented in Table 4.1.

Table 4.1: Comparison of run-times (hh:mm:ss) between stochastic simulations in iBioSim [226] and model verification in STAMINA [171]. The first column displays the input transitions, with the order of inputs being IPTG, aTc, and Ara. The stochastic simulation in iBioSim involved 100,000 Gillespie runs. The columns labeled iBioSim and STAMINA are further divided into three subcolumns presenting run-times for the original layout (**OG**) (Figure 4.2(a)), the two-inverter layout (**TI**) (Figure 4.2(b)), and the logic-hazard-free layout (**LHF**) (Figure 4.2(c)) of the circuit. All stochastic simulations and model checking tasks were executed on a computer equipped with an AMD Ryzen Threadripper 12-Core 3.5 GHz Processor and 132 GB of RAM, running Ubuntu Linux (v18.04.3).

	iBioSim			**STAMINA**		
Input Transition	OG	TI	LHF	OG	TI	LHF
$(0,1,0) \rightarrow (1,1,1)$	00:08:45	00:05:36	00:08:06	MIR	05:55:28	03:50:45
$(0,1,0) \rightarrow (1,0,0)$	00:01:43	00:02:00	00:03:08	00:00:12	00:24:42	00:01:06
$(1,1,1) \rightarrow (1,0,0)$	00:01:31	00:02:01	00:01:35	00:00:49	01:40:26	00:06:27
$(1,1,1) \rightarrow (0,1,0)$	00:01:48	00:03:43	00:01:44	00:01:43	03:21:48	00:38:27
$(1,0,0) \rightarrow (0,1,0)$	00:01:47	00:03:36	00:02:01	00:00:35	00:52:50	00:01:55
$(1,0,0) \rightarrow (1,1,1)$	00:08:54	00:10:12	00:08:01	00:23:13	OOM	02:53:42
$(0,1,1) \rightarrow (1,0,1)$	00:01:51	00:05:38	00:01:46	00:41:40	05:49:56	06:58:33
$(0,0,0) \rightarrow (0,1,1)$	00:04:51	00:05:51	00:04:35	00:26:32	06:17:22	06:30:08
$(0,0,0) \rightarrow (1,0,1)$	00:01:31	00:05:13	00:01:36	00:39:42	06:50:17	01:07:43
$(1,0,1) \rightarrow (0,1,1)$	00:01:56	00:02:05	00:01:50	00:40:47	02:44:57	06:01:4
$(0,1,1) \rightarrow (0,0,0)$	00:01:44	00:02:24	00:01:58	00:05:45	02:49:01	00:17:04
$(1,0,1) \rightarrow (0,0,0)$	00:01:43	00:02:01	00:02:13	00:02:13	01:03:09	00:13:57

The stochastic simulation for the transition IPTG, aTc, Ara = (1, 0, 0) to (1, 1, 1) in the two-inverter circuit had the longest run-time, lasting 10 minutes and 12 seconds for 100,000 simulation runs. STAMINA failed to compute an adequately tight probability window for the same input transition. On the original circuit, the transitions IPTG, aTc, Ara = (1, 1, 1) to (1, 0, 0) and

IPTG, aTc, Ara = (0, 0, 0) to (1, 0, 1) resulted in the fastest stochastic simulations, each taking 1 minute and 31 seconds. The same transition checked by STAMINA took only 49 seconds, which is 54 percent of the time it took iBioSim. The longest run-time for stochastic model checking (excluding MIR or OOM models) occurred for the input transition IPTG, aTc, Ara = (0, 1, 1) to (1, 0, 1) of the logic-hazard-free circuit, lasting 6 hours, 58 minutes, and 33 seconds. State space generation took 1 hour, 11 minutes, and 1 second, and the analysis took 5 hours, 47 minutes, and 32 seconds. All stochastic model checking runs marked with an asterisk in Figure 4.4 (b), 4.5 (b), and 4.6 (b) report the run-time to obtain the provided result and do not include the run-time between obtaining this result and running out of memory in the next iteration, as this is not reported by the tool.

4.3.3 Computational Analysis of Input Transitions without Function Hazards

Following the analysis of input transitions with known function hazards, the two-input change transitions without function hazards were examined. These transitions are still categorized as either *static 0→0* or *static 1→1*, meaning the initial and final states are both either 0 or 1, but they lack a function hazard. Despite this, they can still experience failure due to the noisy behavior of the system. Similar to the function hazard transitions, these input transitions fail if the circuit cannot maintain its state throughout the transition. The failure is also transient and corrects itself when a steady state is reached. However, as these transitions lack a function hazard, a lower probability of failure is anticipated. The probabilities of failure are presented in Table 4.2.

In comparison to the glitch probabilities presented in Figure 4.4 (b), 4.5 (b), and 4.6 (b), transitions without function hazards are less prone to failure. The median probability among all function hazard transitions for the original circuit is 86 percent, while the median for input transitions without function hazards is 52 percent. Similarly, for the two-inverter design, function hazard transitions have a median of 81 percent compared to 54 percent.

The logic-hazard-free circuit performs best, with a median of 80 percent for transitions with function hazards, but only 36 percent for transitions without. This result aligns with expectations

Input Transition	OG	TI	LHF
$(0,1,0) \rightarrow (1,1,0)$	0.195	0.255	0.071
$(1,0,0) \rightarrow (1,1,0)$	0.234	0.247	0.122
$(1,1,0) \rightarrow (1,1,1)$	0.284	0.317	0.265
$(1,1,0) \rightarrow (0,1,0)$	0.325	0.393	0.211
$(1,1,0) \rightarrow (1,0,0)$	0.719	0.719	0.209
$(1,1,1) \rightarrow (1,1,0)$	0.263	0.276	0.207
$(0,0,1) \rightarrow (1,0,1)$	0.741	0.551	0.751
$(0,1,1) \rightarrow (0,0,1)$	0.524	0.534	0.526
$(0,0,0) \rightarrow (0,0,1)$	0.666	0.537	0.532
$(1,0,1) \rightarrow (0,0,1)$	0.519	0.711	0.522
$(0,0,1) \rightarrow (0,1,1)$	0.741	0.753	0.742
$(0,0,1) \rightarrow (0,0,0)$	0.712	0.735	0.463

Table 4.2: The table presents the glitch probability of circuit 0x8E for input transitions without function hazards. It is divided into upper and lower sections, where the upper section shows *static 0→0 transitions*, and the lower half shows *static 1→1 transitions*. The first column specifies the input transitions, and each cell in the table indicates the probability of failure for each of the three circuit designs. The original circuit version is labeled **OG**, the two-inverter version as **TI**, and the logic-hazard-free version as **LHF**. The order of the inputs is IPTG, aTc, Ara.

since the transitions of the original and two-inverter designs do not have a function hazard but might still have a logic hazard, which the logic-hazard-free version eliminates.

The models employed here maintain the same abstraction, with the input molecules not being explicitly modeled but represented by their internal states, and production and degradation occurring in steps of ten. To obtain the results, iBioSim was utilized for stochastic simulation with 100,000 Gillespie runs [88], employing the same property as for the function hazard transitions, as given in Equation 3.1.

4.3.4 Computational Analysis of Steady-State Failure

In Section 3.2, an additional potential failure mode of genetic circuits was introduced, known as steady-state failures. These failures occur when the circuit is unable to attain its expected output. In such cases, if the circuit is designed to maintain a high output under specific input conditions, it falls short of achieving this goal and remains in a low steady state. This failure is particularly consequential as it compromises the intended function of the circuit, potentially leading to the absence of pharmaceutical production or the failure to release a necessary drug within a biological system.

Once again, this failure can be attributed to the inherent noisy behavior of biological systems,

introducing random variations in the circuit's behavior. This section extends the analysis of the three circuit 0x8E designs by assessing the likelihood of steady-state failures.

Steady-state failure encompasses scenarios where the circuit remains in a low output when a high output is expected, and vice versa. To analyze this behavior, the property outlined in Equation 3.2 is employed. This property checks whether the circuit achieves its expected output after a specified time t. The value of t is determined through an ODE analysis of the circuit's states to identify the time it takes to settle into its steady state. Subsequently, SSA analysis is conducted to estimate the probability of failure. The results are presented in Table 4.3.

Input State	**OG**	**TI**	**LHF**
$(0, 0, 0)$	0.058	0.088	0.057
$(0, 0, 1)$	0.001	0.002	0.003
$(0, 1, 0)$	0.042	0.051	0.045
$(0, 1, 1)$	0.098	0.123	0.103
$(1, 0, 0)$	0.029	0.006	0.030
$(1, 0, 1)$	0.091	0.096	0.103
$(1, 1, 0)$	0.022	0.024	0.006
$(1, 1, 1)$	0.050	0.088	0.051

Table 4.3: Probability of steady-state failure for circuit 0x8E. The first column presents the eight distinct possible states of the circuit. The table cells indicate the probability of the circuit failing to attain its anticipated output state. The original circuit version is denoted as **OG**, the two-inverter version as **TI**, and the logic-hazard-free version as **LHF**. The input order is IPTG, aTc, Ara.

The results presented in Table 4.3 indicate a lower probability of steady-state failure compared to transition failures. For instance, the probability of the original circuit failing to reach its expected state for IPTG, aTc, Ara = (0, 0, 1) is 0.1 percent. The median failure rate across all eight states of the original version of circuit 0x8E is 4.6 percent, notably lower than its probability of glitching for transitions with function hazards (86 percent) or those without function hazards (52 percent). A similar trend is observed for the two-inverter circuit, with a probability of steady-state failure at 7 percent compared to 81 percent and 54 percent for transition failures, and for the logic-hazard-free design (4.8 percent versus 80 percent and 36 percent). The simulation results are supported by laboratory testing, as no steady state failure for the original circuit was reported. To verify whether the circuit attains its anticipated steady state, the models were not intialized in the steady state initially. Instead, the circuit was allotted 1000 seconds to stabilize before assessing whether the constraint had been violated.

In summary, this analysis indicates that circuit 0x8E is more prone to failure due to input transitions than steady-state failures. Consequently, the results emphasize the importance of ensuring correct transition behavior during the design process.

4.4 Conclusion

This chapter underscores the benefits of employing computational analysis in genetic circuit design. ODE, SSA, and stochastic model verification analyses are all valuable tools during the design phase to ensure the proper functioning of a genetic circuit. By leveraging these tools, designers can enhance their workflow and design genetic circuits more rapidly without compromising robustness. However, the next question that arises is the trustworthiness of the model results.

As mentioned in Section 4.2, the models used for the analysis are abstracted to allow analysis using stochastic model verification. Abstracted models enable faster and more efficient analysis but compromise on the detail of obtainable information. Therefore, how far can a model be abstracted without losing critical information about the genetic circuit? The next chapter builds upon the results presented here and compares them to the results of the same analysis using four additional, different model techniques.

"He that breaks a thing to find out what it is, has left the path of wisdom."

- Gandalf the Grey

5

Comparison of Different Genetic Circuit Models

Chapter 4 highlighted the significance of computational analysis in synthetic biology. Following this work, it becomes essential to inquire whether the existing computational models can indeed ensure accurate *predictability* of GRNs. As genetic circuitry progresses towards real-world applications, understanding and accommodating environmental and noise effects on circuit performance—referred to as *robustness*—becomes imperative for ensuring the correct and safe behavior of genetic circuits [198, 34]. To enable efficient analysis, it is necessary to have a predictive model of genetic circuits, grounded in mathematical descriptions of genetic networks [179].

Properly parameterized ODE models can provide a good quantitative match and are easily generalized [119, 223, 2, 51]. However, more detailed models entail a more challenging charac-

terization effort. While re-parameterizing genetic parts through experiments yields more accurate and precise behavior predictions, it necessitates extensive training in exploring multidimensional design spaces and mapping simulations to experiments for model development. Therefore, the current situation involves a trade-off between the model's ability to quantitatively match experimental data and the need for numerous kinetic parameters to parameterize the model [112, 99, 136, 90]. Furthermore, there is a simulation-time cost associated with these models: the more complex a model is, the longer it takes for simulations to run, and the higher are the memory requirements for model-checking [39, 171, 188].

Chapter 4 demonstrated the predictive capabilities of *in silico* analysis for circuit 0x8E. Expanding upon this foundation, this chapter delves further into computational modeling and analysis. Specifically, it compares the computational model used in Chapter 4 with four additional and distinct computational models for the same circuit. This chapter evaluates and compares the relative predictability of robustness, along with its impact on design decisions, across diverse characterizations and models. Distinctions are made among various interaction models, some incorporating experimentally obtained parameters while others utilizing standard parameters derived from average values in the literature [226, 139]. Different noise sources are another distinction considered to ascertain the anticipated robustness of three distinct circuit implementations of circuit 0x8E [175], shown in Figure 4.2. The objective is to evaluate whether less complex models can yield the same conclusions as more complex ones and whether certain models offer greater benefits in analysis.

The presented results focus on determining variations in predicted circuit failure percentages across three distinct circuit layouts with identical expected functions. They are derived from utilizing ODE analysis and SSA, as discussed in the previous Chapter 4, and do not include stochastic model checking due to its memory requirements and runtime. In Section 5.1, the analysis goals are introduced, incorporating definitions for robustness and predictability, as well as extrinsic and intrinsic noise. Following this, Section 5.2 provides an overview of the various computational models employed. Finally, the outcomes of the analysis are detailed in Sections 5.3 and 5.4, covering both transient and steady-state failures.

5.1 Robustness, Predictability, and Noise in Genetic Circuit Design

When considering robustness and predictability, simulations involving noise can potentially offer additional information. However, challenges associated with parameterizing noise factors and the computational resources required for simulating stochastic models may discourage many scientists.

This work assess the predictability of robustness and its impact on design choices across various characterizations and models. The goal is to understand the effort required for parameter determination and model development in order to qualitatively evaluate the relative robustness of different design choices. This comparison involves different models, some parameterized and others utilizing default parameters obtained from the literature, as well as different noise sources to predict the robustness of three distinct circuit implementations of circuit 0x8E.

5.1.1 Robustness and Predictability

Engineered systems are expected to function correctly in diverse environments. Robustness represents a system's ability to withstand and operate under the effects of external disturbances. Synthetic biologists, akin to other engineering disciplines, must consider robustness when designing genetic circuits, especially for applications beyond the laboratory. The impact of noise on a circuit's behavior can lead to glitches, as analyzed in Chapter 4, with failure modes including *static* $0 \rightarrow 0$ and *static* $1 \rightarrow 1$ *function-hazard* transitions, *static* $0 \rightarrow 0$ and *static* $1 \rightarrow 1$ *transitions* without function hazards, and steady-state failures.

Predictability involves the computational analysis of the likelihood of erroneous behavior occurring. However, as with all engineering principles, a trade-off exists between the computational and mathematical complexity of a model and its accurate predictability. A more detailed model generally yields more accurate results, but heightened complexity necessitates more powerful computing capabilities and extended analysis times. While precise models are valuable, they come at a significant cost, demanding extensive resources, time, and expertise. Therefore, assessing whether detailed models provide additional insight into the robustness of a genetic circuit is crucial. When

the primary concern is the resilience and stability of a circuit, questioning the justification for allocating substantial resources to exhaustive characterization experiments and computationally demanding simulations becomes necessary.

5.1.2 Extrinsic and Intrinsic Noise

Chapter 4 demonstrated that ODE analysis, with its deterministic structure, captures the average behavior of a system but lacks randomness or stochasticity, consistently yielding identical results for the same initial conditions [82]. However, the intrinsic unpredictability of biochemical reactions, even at the level of a single gene [72], coupled with variability in reaction rates due to environmental differences [177, 18], introduces uncertainty into a genetic system [184, 212, 177, 127].

Extrinsic noise sources of transcriptional variability refer to cell-to-cell differences in transcriptional inputs and outputs. Beal [18] demonstrated that this cell-cell variation might be accounted for by the emergent properties of complex reaction networks, driving a log-normal distribution of gene expression levels across a population. For comparison, this work utilizes measured parameter distributions from the Cello part library [175] and implements a folded normal-distributed parameter value model to calculate the incidence of glitching behavior in a population.

Intrinsic noise represents the inherent stochasticity within the cell, influenced by factors like spatial considerations, resource distribution, and stress. These elements profoundly impact the cell's likelihood or capacity to carry out reactions. Since the chemical reactions involved are discrete events with probabilities dependent on the molecule count, they inherently reflect this stress. Therefore, to effectively capture these fluctuations, Gillespie's SSA [89, 88] was employed to analyze these models.

5.2 Overview of Different Models

The models, parameterizations, and analysis methods utilized in this study are categorized as follows: the terms *extrinsic* or *intrinsic* indicate whether the model was examined with extrinsic or intrinsic noise. Subsequently, the models are classified as the *Cello model,* developed by Moser

et al. [161, 203], the *default model*, generated in iBioSim [226], or the *abstracted model* based on stoichiometry amplification of the default model.

The Cello model employs kinetic abstractions that yield a model comparable to Hill equations [28]. Despite its simplicity, the Cello model is a benchmark in synthetic biology for several reasons: (1) it provides transparent and reproducible characterization experiments for transcriptional regulatory gates, (2) it offers gate characterization results for researchers to use in their models, and (3) the model provides simpler parameters to characterize by consolidating various regulatory kinetics into experimentally-observable variables.

The default iBioSim model [164, 226] is the most intricate, modeling each protein's transcription initiation, production, dimerization, transcription factor binding, and degradation. Finally, the abstracted model alters the default model by modifying the stoichiometry and rate of its degradation reactions.

Additionally, a comparison between default (obtained from literature) model parameter values and characterized gate parameter values was used to determine the effect on predicted circuit failure percentages. The default parameter values were used for four models and part-characterized model parameter values for each component obtained from experimentation [175] for one of the Cello models. Therefore, the five different model types and their labels are as follows:

- *Extrinsic/Cello model/Characterized parameters* (E/C/C)
- *Extrinsic/Cello model/Default parameters* (E/C/D)
- *Extrinsic/Default model/Default parameters* (E/D/D)
- *Intrinsic/Default model/Default parameters* (I/D/D)
- *Intrinsic/Abstracted model/Default parameters* (I/A/D)

5.2.1 Extrinsic Noise Model

The extrinsic noise model introduces a basic instance of static external disturbances, characterized as a random selection from a folded normal distribution for each parameter value at the

onset of each simulation run. The distribution's mean corresponds to the default parameter value in iBioSim, obtained from the literature, while the standard deviation is set at 0.4 of the mean's absolute value to mimic extrinsic noise. The noise magnitude chosen for this study is 0.4, indicating that in each simulation, a parameter value will equal $|normal(\frac{v}{v*e})|$ with the function *normal*, the default parameter value v, and the extrinsic noise e.

However, as this research primarily focuses on qualitative rather than quantitative comparisons, the precise extrinsic noise values are not crucial. Beal [18] suggests that these parameters may follow a geometric distribution rather than a normal one, posing an area for future exploration.

5.2.2 Intrinsic Noise Model

The default intrinsic model utilizes iBioSim's default settings. The assumption is made that mRNA undergoes ten translations, leading to an average production of ten proteins per mRNA overall. Each protein undergoes self-degradation, modeled in increments of one. Consequently, when a production reaction occurs, ten molecules are produced, while a degradation reaction leads to the degradation of only one protein.

In the abstracted intrinsic model, aiming to simplify the default model, both protein production and degradation are set to increments of ten for their respective reactions. This adjustment, termed stoichiometry amplification, entails that each time a production or degradation reaction takes place, ten molecules are either produced or degraded. Given the modification in the degradation reaction, a tenfold reduction in reaction propensity is necessary to accommodate the alteration.

Additionally, to streamline both models and reduce species complexity, only internal molecules and complexes are modeled, while input molecules such as IPTG, aTc, and Ara are omitted. For example, LacI, serving as an internal regulator of the circuit, is modeled alone, and the input IPTG, which binds to LacI and affects its function, is excluded from the model. This selective modeling approach focuses on key internal components rather than including both species in the analysis.

5.2.3 Model Considerations and Assumptions

This study compares different circuit layouts for each model, considering various modeling alternatives and characterizations, to ascertain which circuit is predicted to be the most robust for specific input transitions and steady-state behavior. However, this work does not aim to determine the more accurate representation of the true behavior of GRNs or quantify the magnitude of each noise source's influence on the predicted output. The primary objective is to identify any differences in robustness predictions when using abstracted models, considering different noise sources (extrinsic or intrinsic), and/or utilizing characterized parameter values versus literature-obtained values. To facilitate the analysis, certain assumptions were made, outlined as follows:

(1) The probability distribution for extrinsic noise is modeled as a truncated-normal distribution. Despite Beal's [18] suggestion that a log-normal distribution may be a more fitting description for this noise, a normal distribution for parameter values was chosen in this work to simulate extrinsic noise, prioritizing simplicity and clarity in simulations. Future work could explore the inclusion of log-normal distributions for a more nuanced representation.

(2) The chosen magnitude of noise level for the truncated-normal distribution is 0.4, derived from initial testing. However, the absolute value of intrinsic or extrinsic noise may differ for GRNs, and if the results indicate variations in robustness predictions, more precise measurements should be obtained for accurate results.

(3) While the level of extrinsic noise could potentially differ for each reaction, affecting the distribution of a parameter's value, this work assumes a uniform magnitude of noise for each reaction parameter.

(4) All simulation runs in this study assume that the change in input molecule concentrations is instantaneous, rather than a gradual process.

(5) The Cello model can utilize τ_{ON} and τ_{OFF} parameters to depict how quickly a gate turns ON or OFF. However, for this work, all different parts share the same values of τ_{ON} and

τ_{OFF} due to the absence of experimental parameter values for individual components.

5.3 Analysis of Transition Failures

Table 5.1 presents a comprehensive overview of the study's findings, showcasing the performance of three circuit layouts: the original design **(OG)**, the two-inverter design **(TI)**, and the logic hazard-free design **(LHF)**. It highlights which layouts outperform (blue) or underperform (orange) in comparison to the median failure rate for specific input transitions, models, and parameter value sets. The determination of transitions as outperforming or underperforming was based on the percent difference to the median. Let x_i be the sample failure percentage and $\bar{x}$ the median. The percent difference was calculated using the following equation:

$$\text{Percent Difference} = \frac{x_i - \bar{x}}{\frac{|x_i + \bar{x}|}{2}} \times 100 \tag{5.1}$$

A transition's percent difference was classified as preferred if it fell below -10 percent, while it was deemed worse if it exceeded 10 percent. For instance, consider the transition from (0, 1, 0) to (1, 1, 1) in the (E/C/C) model. The failure percentages for each model are as follows: **OG** 11.3 percent, **TI** 12.7 percent, and **LHF** 11.5 percent, with the median being 11.5 percent for the **LHF** model.

(1) Percent difference of **OG**:

$$\frac{11.3 - 11.5}{\frac{|11.3 + 11.5|}{2}} \times 100 = -1.75\%$$

(2) Percent difference of **TI**:

$$\frac{12.7 - 11.5}{\frac{|12.7 + 11.5|}{2}} \times 100 \approx 9.92\%$$

In this example, the **OG** implementation shows a percent difference of -1.75 percent, which does not fall below the -10 percent threshold, thus not qualifying as a preferred transition. Similarly, the **TI** model exhibits a percent difference of 9.92 percent, which also does not exceed the 10 percent threshold, thus not indicating a worse transition. Hence, the two cells of these transitions remain uncolored in Table 5.1.

This presentation facilitates the comparison of the three different circuit layouts shown in Figure 4.2 in terms of their predicted failure probabilities (better: blue or worse: orange) concerning the given noise source, model, and parameter values. The table comprises five columns representing the (E/C/C), (E/C/D), (E/D/D), (I/D/D), and (I/A/D) models, respectively. The (I/A/D) model was previously introduced in Chapter 4.

The "Inputs" column showcases the input molecule concentrations. High and low concentrations are denoted as 1 and 0, respectively, for IPTG, aTc, and Ara. For instance, the input transition $(0, 1, 0) \rightarrow (1, 1, 1)$ signifies a shift from low, high, low to high, high, high for IPTG, aTc, and Ara concentrations. Analyzing extrinsic noise modeling with the Cello model and characterized parameter values (E/C/C, first column) reveals that during the transition from (0, 1, 0) to (1, 1, 1), **TI** exhibits a higher failure probability, while **OG** experiences glitches less frequently for that transition. The values provided in the table indicate the deviation in failure percentage difference from the median for the respective model and transition. The median is represented by a dash "-". For instance, in the same example, the failure percentage for the **LHF** model represents the median for this transition. The **TI** model exhibits a percent difference of 9.92 percent, as calculated above, while the **OG** model shows a percent difference of -1.75 percent.

Moreover, the rows in the table are divided into four groups based on different input transition types. The first two groups present the results of the circuit failure analysis for *static* $0 \rightarrow 0$ and *static* $1 \rightarrow 1$ *function hazards*, while the next two groups showcase the results for *static* $0 \rightarrow 0$ and *static* $1 \rightarrow 1$ *transitions* without function hazards. Static transitions without function hazards do not inherently exhibit faulty behavior, unlike static transitions with function hazards, and are expected to behave precisely as anticipated. However, due to noise, there remains a small probability of temporary deviation from the expected behavior. Therefore, analyzing input transitions without function hazards remains crucial for comprehensive circuit analysis. Detailed tables with the absolute failure probabilities, given in percent, can be found in Appendix A.

Table 5.1: Comparison of all model predictions for of genetic circuit failures, and the most "robust" circuit choice for each model simulation. **D**: default parameter values; **C**: characterized parameter values; **OG**: original design; **TI**: two-inverter design; **LHF**: logic hazard-free design.

		Extrinsic Noise									**Intrinsic Noise**					
		Cello Model						**Default Model**			**Default Model**			**Abstracted Model**		
		C			**D**			**D**			**D**			**D**		
Circuit Failures		OG	TI	LHF	OG	TI	LHF	OG	TI	LHF	OG	TI	LHF	OG	TI	LHF
Static 0 → 0-Function Hazards	$(0,1,0) \rightarrow (1,1,1)$	-1.75	9.92	-	34.01	-	-71.11	-	83.68	-19.08	-	107.11	-22.01	-	68.68	-4.15
	$(0,1,0) \rightarrow (1,0,0)$	-6.18	13.33	-	-	44.36	-39.11	-	4.07	-102.55	-	4.24	-124.48	-	8.16	-82.55
	$(1,1,1) \rightarrow (1,0,0)$	-5.69	13.33	-	-	43.48	-40.00	-	2.09	-33.65	-	0.10	-2.88	-	1.06	-3.49
	$(1,1,1) \rightarrow (0,1,0)$	33.33	-	-33.33	6.37	-24.35	-	4.48	-38.25	-	-	-77.14	0.10	-	-38.81	2.99
	$(1,0,0) \rightarrow (0,1,0)$	31.95	-	-34.71	5.06	-26.47	-	16.93	-1.01	-	-	-87.89	2.61	6.61	-43.59	-
	$(1,0,0) \rightarrow (1,1,1)$	-1.75	10.70	-	-	60.96	-11.38	-	33.71	-1.82	-	12.02	-5.63	-3.80	1.85	-
Static 1 → 1-Function Hazards	$(0,1,1) \rightarrow (1,0,1)$	8.91	-7.02	-	10.19	-7.86	-	-	-93.98	1.69	0.10	-41.21	-	0.51	-15.42	-
	$(0,0,0) \rightarrow (0,1,1)$	-2.01	41.26	-	-4.04	15.29	-	-5.89	20.81	-	-2.07	-	1.02	0.23	-	-1.16
	$(0,0,0) \rightarrow (1,0,1)$	8.84	-6.96	-	11.06	-7.59	-	-	-54.24	2.11	0.20	-38.33	-	-	-14.09	0.51
	$(1,0,1) \rightarrow (0,1,1)$	-1.68	43.23	-	-4.72	17.84	-	-	0.11	-0.44	-	0.10	-	-	1.01	-0.20
	$(0,1,1) \rightarrow (0,0,0)$	-44.32	-	12.23	-	17.75	-0.88	-4.61	8.63	-	-	5.41	-44.07	0.34	-	-12.87
	$(1,0,1) \rightarrow (0,0,0)$	-47.51	-	8.79	-	20.99	-0.67	-	19.35	-16.01	-	25.46	-60.19	-	7.89	-15.52
Static 0 → 0-Input Transitions	$(0,1,0) \rightarrow (1,1,0)$	10.53	-	-48.28	-	1.94	-37.21	-	-	-53.33	-	23.28	-75.35	-	15.69	-73.12
	$(1,0,0) \rightarrow (1,1,0)$	9.33	-	-48.65	-	-	-34.48	-	36.92	-21.99	-	11.83	-53.99	-	9.59	-50.98
	$(1,1,0) \rightarrow (1,1,1)$	-1.75	9.92	-	-	60.96	-11.38	-	58.43	-1.60	-	10.58	-4.90	-	12.16	-16.84
	$(1,1,0) \rightarrow (0,1,0)$	34.23	-	-31.93	7.07	-23.88	-	1.87	-	-63.35	16.45	-	-17.61	9.91	-	-47.91
	$(1,1,0) \rightarrow (1,0,0)$	-6.18	13.33	-	-	44.36	-39.11	-	4.28	-133.33	0.14	-	-122.15	-	3.11	-104.18
	$(1,1,1) \rightarrow (1,1,0)$	11.52	-	-46.58	-	-	-34.48	-7.41	61.39	-	-	10.20	-7.24	-	14.60	-11.59
Static 1 → 1-Input Transitions	$(0,0,1) \rightarrow (1,0,1)$	8.91	-7.02	-	10.97	-6.90	-	3.39	-	-	-	-51.55	8.48	-	-27.74	0.67
	$(0,1,1) \rightarrow (0,0,1)$	-3.17	-	-	-	13.91	-6.76	-30.73	12.66	-	-	19.64	-2.70	-1.06	-	0.17
	$(0,0,0) \rightarrow (0,0,1)$	-3.17	-	-	-	13.79	-8.70	-1.95	26.17	-	-2.75	6.02	-	30.91	-	-10.34
	$(1,0,1) \rightarrow (0,0,1)$	-3.17	6.06	-	-	17.87	-8.78	-	0.45	-5.58	-4.99	46.78	-	-	22.56	-11.17
	$(0,0,1) \rightarrow (0,1,1)$	-2.35	41.37	-	-3.34	15.98	-	-13.81	27.98	-	-2.55	9.87	-	-84.11	3.62	-
	$(0,0,1) \rightarrow (0,0,0)$	-45.76	-	10.64	-	18.22	-2.02	-	13.86	-47.65	3.02	-	-118.51	-	5.13	-40.34

This table compares each models' predictions for best and worst performance for each transition. Orange color means that the circuit performed worse and the blue color means that the circuit performed better compared to the median, for that input.

5.3.1 Quantitative Model Prediction

The results facilitate several conclusions. The first aim is to compare the various modeling techniques and their outcomes to derive insights. For this purpose, Figure 5.1 illustrates the analysis results for all transitions of the **OG** implementation. On the y-axis, transitions are delineated into the same four groups, presented in the same sequence as in Table 5.1. Meanwhile, the x-axis represents the failure probability in percent. Each transition depicts five markers, with each marker corresponding to one of the five models, as denoted in the legend.

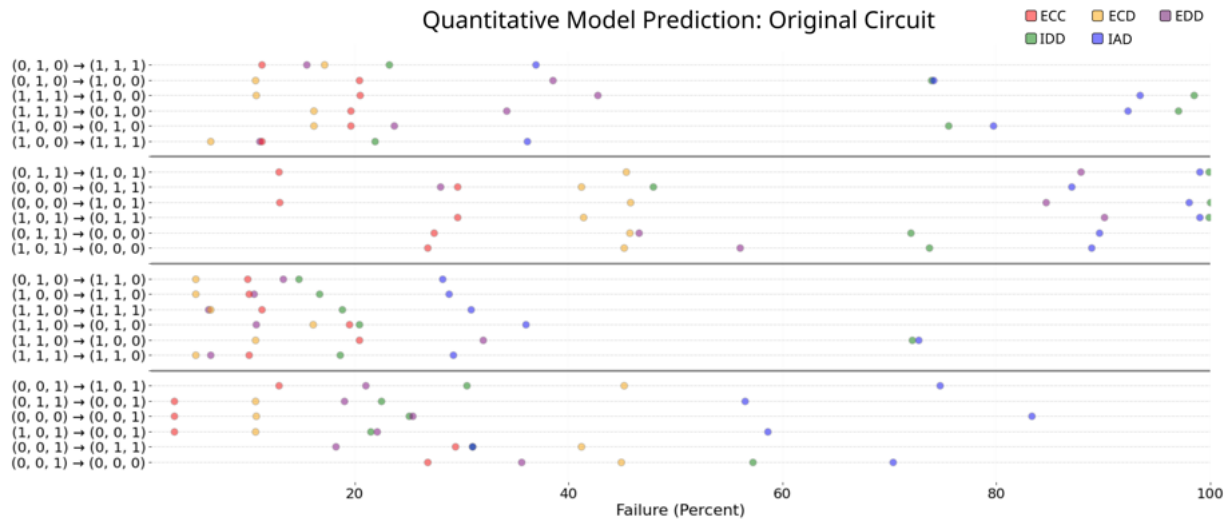


Figure 5.1: Quantitative model prediction. The figure illustrates the predicted failure probability for each transition and model concerning the **OG** layout. The transitions on the y-axis are grouped into four sections. The first two sections correspond to *static* $0 \rightarrow 0$ and *static* $1 \rightarrow 1$ *function hazards*, while the next two groups correspond to *static* $0 \rightarrow 0$ and *static* $1 \rightarrow 1$ *transitions* without function hazards. On the x-axis, the failure probability is represented in percentages, with each model indicated by a distinct marker.

The results demonstrate the challenge of comparing the magnitude of each failure probability across different models. While certain transitions, such as the $(0, 1, 0) \rightarrow (1, 1, 1)$ transition, exhibit similar predictions among the various models, others, like the $(0, 1, 1) \rightarrow (1, 0, 1)$ transition, display significant differences. Consequently, the selection of the model becomes crucial when aiming to determine the most accurate representation of the true behavior of the genetic circuit and achieve precise quantification of each failure. If this level of precision is essential for the designer, and

no characterized parts or experimental data are available, considerable laboratory work would be required. Appendix A contains the corresponding visualizations for the **TI** and **LHF** design.

5.3.2 Qualitative Model Prediction

The analysis of the qualitative predictive power involves comparing the number of preferable transitions for each circuit within a specified model. The number of preferable transitions for each circuit and model can be obtained by counting the blue fields in Table 5.1. However, for easier comparison Figure 5.2 shows the number of preferable transitions for each circuit design and model.

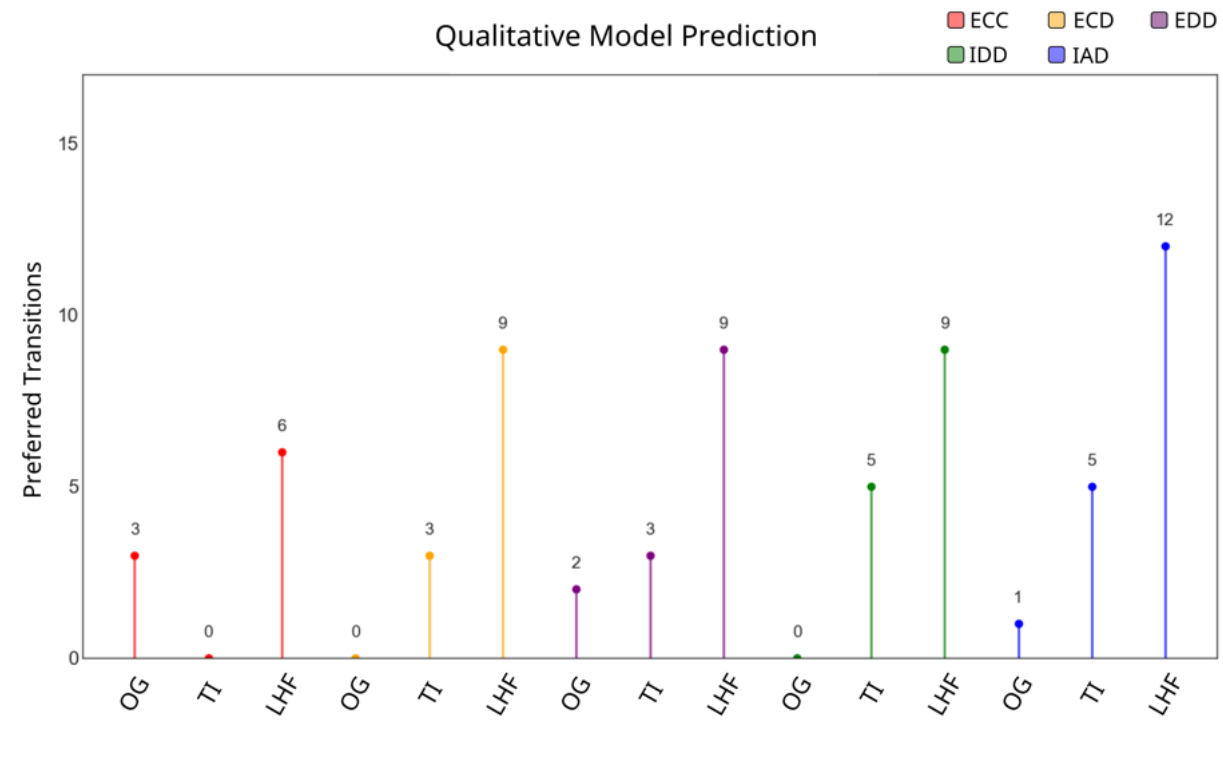


Figure 5.2: Qualitative model prediction. The y-axis denotes the number of preferred transitions, while the x-axis represents the circuit implementation. The five distinct models are distinguished by color coding, as specified in the legend. All models unanimously agree that the **LHF** design exhibits the highest number of preferred transitions. However, it is worth noting that in the (E/C/C) model, the **OG** design is rated better than the **TI** design, which contrasts with the conclusions drawn by all other models.

The results reveal several distinctions. First, there is a discrepancy in predictions between the model employing characterized parameters (E/C/C) and the other four models utilizing iBioSim's

default parameters. While all models unanimously agree that the **LHF** design exhibits the highest number of preferred transitions, the (E/C/C) model identifies the **OG** design as having more preferable transitions than the **TI** design. In contrast, the other four models agree that the **TI** design possesses more preferred transitions than the **OG** design.

Secondly, the results highlight a greater variance in outcomes among models lacking characterized parameters compared to those with characterized parameters. Specifically, the model with characterized parameters exhibits the fewest preferred transitions overall, with only six identified for the **LHF** design, in contrast to other models. For instance, the (I/A/D) **LHF** design reveals twelve preferred transitions.

Further investigating the results, Figure 5.3 extends its analysis beyond solely the preferred transitions, depicted in blue in the table, to also include the worse transitions, represented in orange. This figure illustrates the five model types on the y-axis and their corresponding scores on the x-axis. The score is determined as the difference between the number of preferred and worse transitions. Each model exhibits three markers, symbolizing the three circuit implementations.

Figure 5.3 suggests that when both preferred and worse transitions are considered, all models converge in their assessments. They unanimously conclude that the **LHF** design exhibits the best performance, followed by **OG**, and then the **TI** design. Comparing the score with the number of preferred transitions, it becomes evident that although the **TI** design boasts more preferred transitions than the **OG** design, it also has more worse transitions, leading to a lower score across all models. With the transitions evaluated, the subsequent section delves into analyzing the steady-state behavior of the designs.

5.4 Analysis of Steady-State Failures

Table 5.2 examines the occurrence of steady-state failure, specifically the probability of a given design reaching equilibrium in the wrong state. The table encompasses the five distinct models, each evaluated against the three different circuit implementations. The values presented represent the disparity in percent difference from the median, denoted by a dash "-". Similarly, blue cells signify

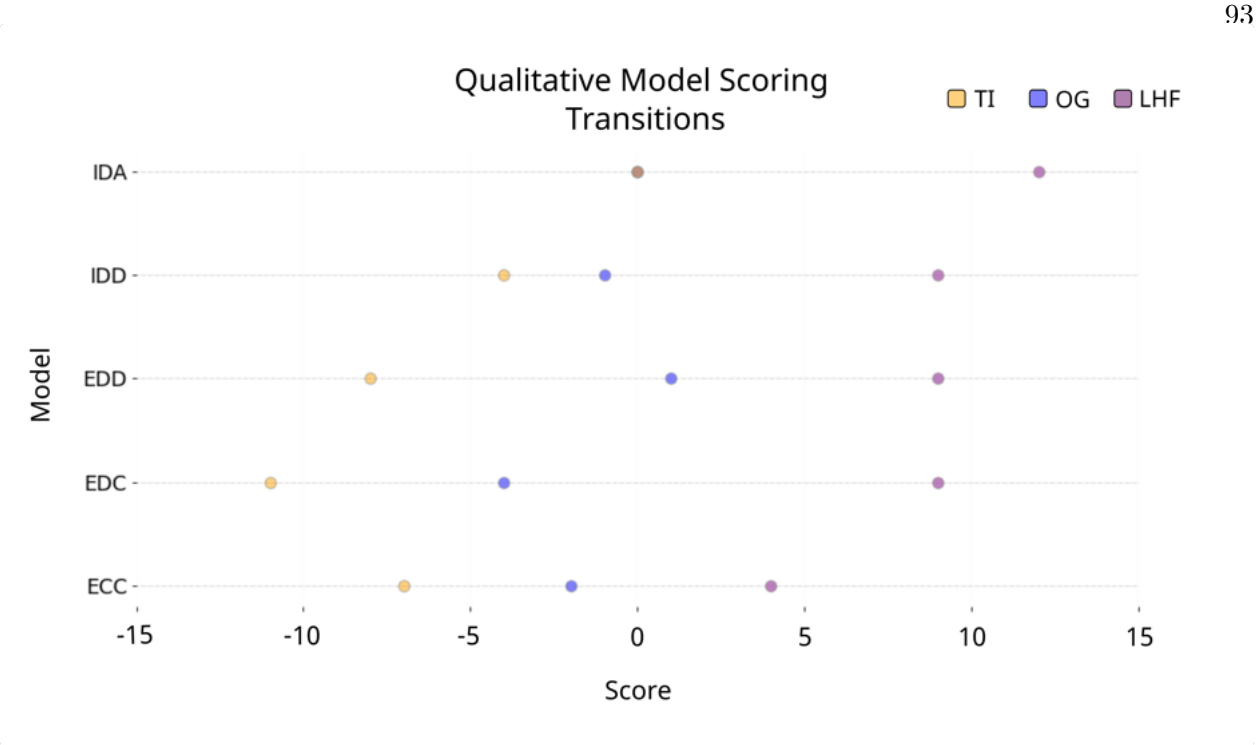


Figure 5.3: Qualitative model scoring. The figure illustrates the score of each circuit implementation within a specific modeling method. The model is depicted on the y-axis, while the x-axis represents the score. Each colored marker corresponds to one of the three circuit designs. The score is calculated as the difference between the number of preferred and worse transitions.

the preference for the circuit design in that state, while orange cells indicate a worse outcome. The identification method adheres to the previously described approach.

Table 5.2: Comparison of all model predictions for of genetic circuit failures, and the most "robust" circuit choice for each model simulation. **D**: default parameter values; **C**: characterized parameter values; **OG**: original design; **TI**: two-inverter design; **LHF**: logic hazard-free design.

	Extrinsic Noise									**Intrinsic Noise**					
	Cello Model						**Default Model**			**Default Model**			**Abstracted Model**		
	C			D			D			D			D		
Circuit Failures	OG	TI	LHF	OG	TI	LHF	OG	TI	LHF	OG	TI	LHF	OG	TI	LHF
(0, 0, 0)	-47.18	-	10.64	-0.23	17.81	-	-3.94	23.81	-	-14.43	14.29	-	-	89.54	-7.87
(0, 0, 1)	-	3.17	-	-	14.04	-13.07	-	-	-11.76	-	-	-	100.00	-	-
(0, 1, 0)	34.23	-	-31.93	7.07	-23.88	-	-	-	-53.33	-	-	-111.11	-40.00	15.38	-
(0, 1, 1)	-2.35	41.37	-	-3.34	15.98	-	-15.31	30.18	-	-12.90	37.04	-	-	8.26	-15.81
(1, 0, 0)	-7.16	12.53	-	-	44.36	-39.11	-2.38	47.53	-	50.00	-	-40.00	-	40.00	-19.61
(1, 0, 1)	8.91	-7.02	-	10.53	-6.90	-	-	2.40	-7.56	-50.85	12.66	-	13.76	-	-5.85
(1, 1, 0)	10.53	-	-48.28	-	-	-37.21	-	58.33	-131.71	-	22.22	-200.00	16.00	-	-128.57
(1, 1, 1)	-1.75	9.92	-	-	60.96	-11.38	-11.38	56.35	-	-66.67	-	15.38	-8.51	57.97	-

This table compares each models' predictions for best and worst performance for each transition. Orange color means that the circuit performed worse than the median, and blue color means that the circuit performed better than the median, for that input molecule concentration or transition.

Figure 5.4 compares the findings presented in the table by aggregating the preferred and worse identified states into a score. The y-axis in the figure represents the five models, while the x-axis denotes the score. Each colored marker corresponds to a circuit design, as indicated in the provided legend.

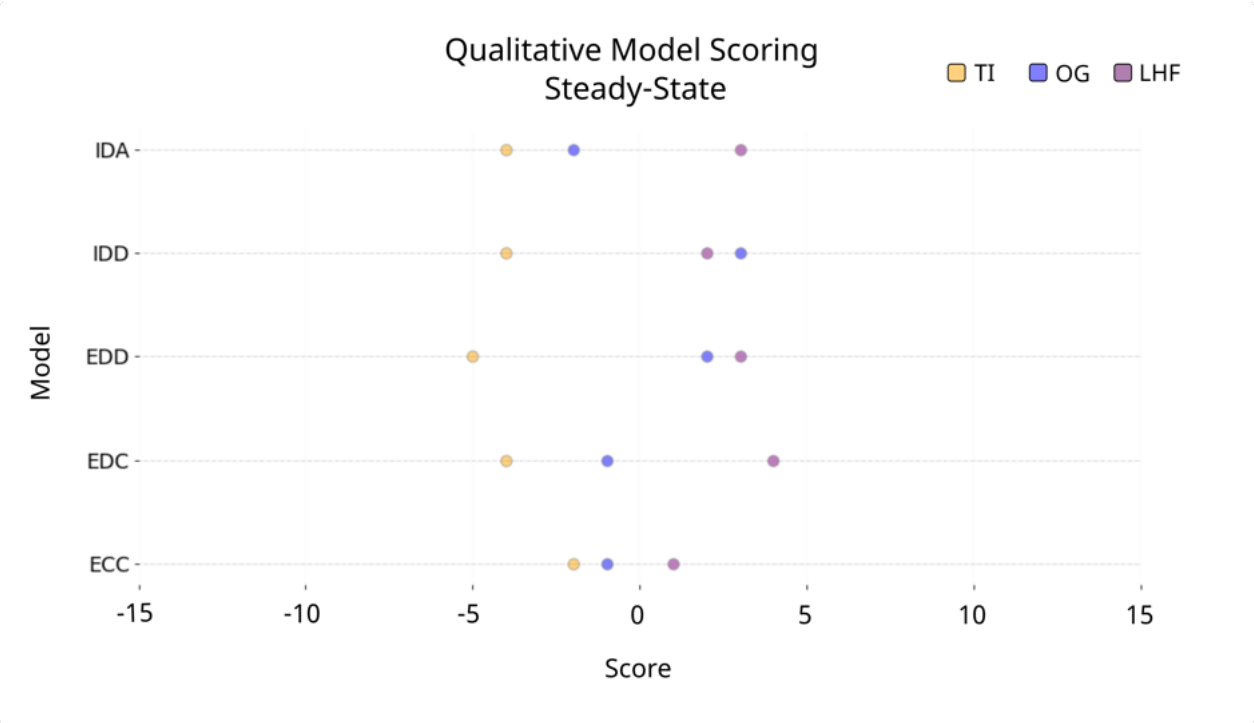


Figure 5.4: Qualitative model scoring. The figure illustrates the score of each circuit implementation within a specific modeling method. The model is depicted on the y-axis, while the x-axis represents the score. Each colored marker corresponds to one of the three circuit designs. The score is calculated as the difference between the number of preferred and worse steady-states.

Similar to the previous findings, when both preferred and worse steady-states are taken into account, all models converge on the conclusion that the **LHF** design outperforms the other two designs. The (I/D/D) model is the only exception, as it gives the **OG** design a higher score than the **LHF** design. However, when considering both the scores of the transitions and the steady-states, the **LHF** design is unanimously chosen as the most robust. However, further research is required to contextualize the results experimentally and compare the actual observed failure magnitude to the predicted failure of each model.

The predictions for the absolute percentage failure of the models, environmental conditions,

and investigated variables are displayed in Appendix A. Tables A.1, A.2, and A.3 present the likelihood of circuit failures for different input transitions in the circuit implementations depicted in Figure 4.2. Table A.1 illustrates the failure percentages of the **OG** circuit, Table A.2 provides the same information for the **TI** design, and Table A.3 presents the corresponding data for the **LHF** design.

5.5 Concluding Remarks

Ultimately, variations exist among the five different modeling techniques in certain individual cases. However, the results indicate that when the overall behavior of the circuit is the primary concern and no characterized parameters are available, a designer can initially utilize a higher-level abstract model, which is less complex, to develop an overall robust design. As the process progresses, if critical transitions are identified as highly failure-prone, the user can then transition to a lower-level abstraction model to refine the design for the specific use case.

These findings, along with further research, contribute to advancing the DMABTL pipeline. This advancement facilitates the learning and design stages by filtering out circuit layouts with a higher likelihood of glitches for input transitions that are deemed critical by the designer. Additionally, the implementation of a model generator in genetic design automation tools, which automatically incorporates intrinsic or extrinsic noise sources, would assist genetic circuit designers in applying and testing different noise levels to obtain failure predictions and assess circuit robustness. This work demonstrates that more abstracted models yield similar results to expanded or characterized models. Thus, computationally less complex models can still provide meaningful insights for design space exploration. It would be beneficial to have a "knob" feature in the future that allows easy adjustment of model abstraction or the level of noise exposure.

"There are other forces at work in this world Frodo, besides the will of evil."

- Gandalf the Grey

6

Dynamic Influence on Steady-State Behavior

Chapters 4 and 5 delve into a comprehensive analysis of circuit 0x8E, exploring various design choices by evaluating static glitches and steady-state behavior, while also considering different mathematical modeling techniques. However, dynamic hazards, as illustrated in Figure 3.2 (b), were overlooked in the preceding analysis. This omission was grounded in the argument that the circuit eventually reaches the correct state, albeit with a delay.

To reassess this perspective, the current chapter shifts its focus to the dynamic hazards associated with circuit 0xF6 and their impact on its steady-state behavior. It introduces the contrary hypothesis that the dynamic behavior of the circuit is indeed significant, influencing the circuit's steady-state behavior.

The chapter initiates with the presentation of experimental results of circuit 0xF6 in Section 6.1, providing the motivation for exploring the hypothesis. Specifically, the experimental analysis focuses on the circuit's output production over time (Section 6.1.1) and the growth of the cell cultures hosting the circuit plasmids (Section 6.1.2).

Subsequently, the chapter shifts its attention to the computational analysis of the circuit in Section 6.2, aiming to characterize the experimental results. The computational analysis employs ODE (Section 6.2.1) and SSA (Section 6.2.2) simulations, both methods from previously presented work, to elucidate the observed behavior. Section 6.3 identifies the cause of the observed behavior to substantiate the hypothesis. Finally, Section 6.4 seeks to validate the central hypothesis of this chapter by rerunning a modified version of the initial experiment.

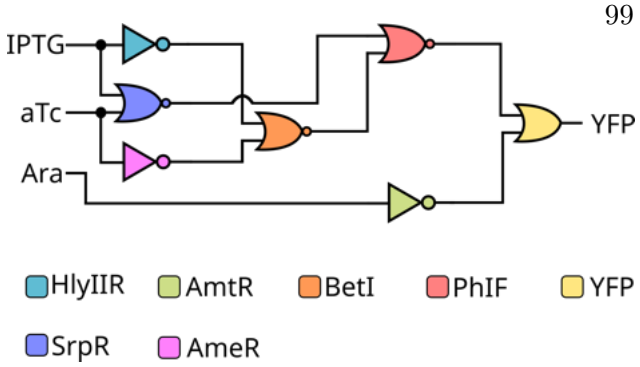
6.1 Experimental Analysis of Circuit 0xF6

Circuit 0xF6, much like circuit 0x8E, is part of the collection of 60 genetic circuits documented by Nielsen et al.[175]. Illustrated alongside its truth table in Figure 6.1, this circuit stands as one of the more extensive designs within the Cello project, comprising seven distinct logic gates. The Cello project primarily centers on the automated design of such genetic circuits through the utilization of the Cello software tool. The validation of all circuits concentrated on ensuring correct steady-state behavior, accomplished through flow cytometry. This method captures a snapshot of cells' fluorescence at a specific point in time and is frequently employed to verify a circuit's steady-state behavior by measuring the fluorescence of the output after the circuit reached its steady state. While considerations of the circuit's dynamics were not taken into account during the design process, the work does present a time series of two circuits' behavior over time. One of them is circuit 0x8E, as shown in Figure 3.4. The time series was obtained using flow cytometry to capture multiple snapshots at different time intervals.

For the work presented in this chapter, the analysis focuses on the steady-state behavior of circuit 0xF6. Initially, to replicate and verify the results of the Cello publication [175], the steady-state behavior was also confirmed using flow cytometry as shown in Figure 3.2 in Chapter 3. However,

Inputs			Output
IPTG	aTc	Ara	YFP
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	0

(a) Truth table of circuit 0xF6



(b) Logic of circuit 0xF6

Figure 6.1: Circuit 0xF6, as documented by Nielsen et al.[175]. (a) The circuit's functionality is presented via a truth table. The original configuration includes three inputs — IPTG, aTc, and Ara — and one output, YFP. (b) The circuit's logic is portrayed, featuring three NOT gates, three NOR gates, and one OR gate.

an interesting observation emerged during the validation experiment. Upon visual inspection of the deep 96-well plate, depicted in Figure 6.2, prepared for the experiment, it was noted that not all states with an expected high output were equally active. Specifically, state IPTG, aTc, Ara = (0, 1, 1), highlighted in pink in the figure, appeared to be closer to being off than on. For comparison, all states anticipating a high output are highlighted in green, while the expected off states are represented in blue, aligning with the truth table provided in Figure 6.8.

Revisiting the flow cytometry data, shown in Figure 3.2, confirmed this observation, with the detected fluorescence for this state being magnitudes lower than those of the other on states, given the logarithmic scale of the plot. As stated in Chapter 3, the circuit's output, YFP, was substituted with sfGFP, known for its enhanced stability and fluorescence, rendering detection more straightforward.

To further investigate the circuit's behavior and extend previous work, an additional analysis method was employed to characterize the circuit's behavior over time rather than just at its steady state. A plate reader was used to measure the fluorescence of cells in a 96-well plate while the cells are alive and growing. While flow cytometers count and measure each individual cell, a plate reader measures an entire well of a 96-well plate containing a complete colony of cells. Furthermore, a

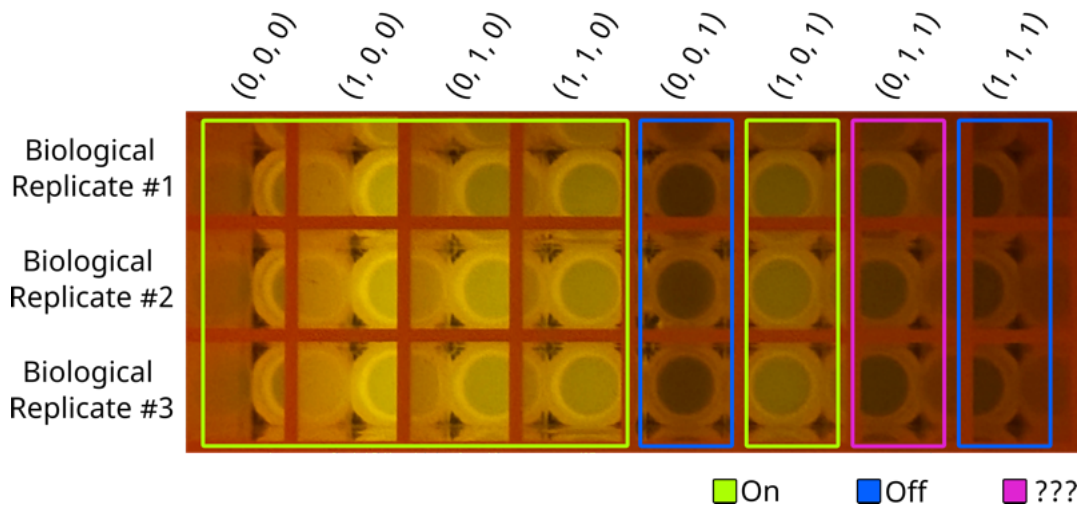


Figure 6.2: 96-well plate showing steady state of circuit 0xF6. The first three rows of the 96-well plate display three biological replicates of circuit 0xF6. Columns represent the eight different states, with those anticipated to be in an on state highlighted in green, and states with an expected low output highlighted in blue. The state marked in pink is IPTG, aTc, Ara = (0, 1, 1) - a state anticipated to have a high output. However, visually discerning the fluorescence of this state proves challenging.

plate reader offers the advantage of recording fluorescence and, consequently, the circuit's output over a long period, such as the 15-hour duration of this experiment, unlike flow cytometry.

The plate reader process begins with loading a plate containing the samples. Subsequently, the plate reader employs a laser with a specified wavelength, in this case, an excitation at 485 nm, to illuminate each well containing a sample. The laser's light interacts with the sample, and the emitted signals are measured by the plate reader, then sent for analysis to the associated software. The emission of the samples was detected at 530 nm wavelength. The measurement of one sample takes seconds, and after scanning the entire plate, the process is repeated for a user-specified time window—15 hours in this analysis. The analysis of these measurements allows for plotting both the fluorescence and the *optical density* (OD) over time. The former represents the signal emitted by the cells, while the latter is used to quantify the growth of the colonies.

6.1.1 Experimental Fluorescence Analysis of sfGFP Production

The results of the plate reader analysis quantitatively validate the visual inspection. Figure 6.3 illustrates the measured fluorescence over time for cell colonies containing circuit 0xF6 and its corresponding output plasmid producing sfGFP. The x-axis displays time in hours, while the y-axis represents fluorescence in *relative fluorescence units* (RFU). RFUs measure the fluorescence of a sample relative to a reference fluorescence. The reference fluorescence is based on sfGFP expression of cells regulated by an upstream constitutive promoter.

The first row shows the results of the measurement of biological replicate #1 highlighted in blue, with each column representing one of the eight states. The middle two rows show the results for biological replicate #2 and #3. Finally, the three measurements in the last row display the three biological replicates just expressing sfGFP without the circuit for the reference fluorescence.

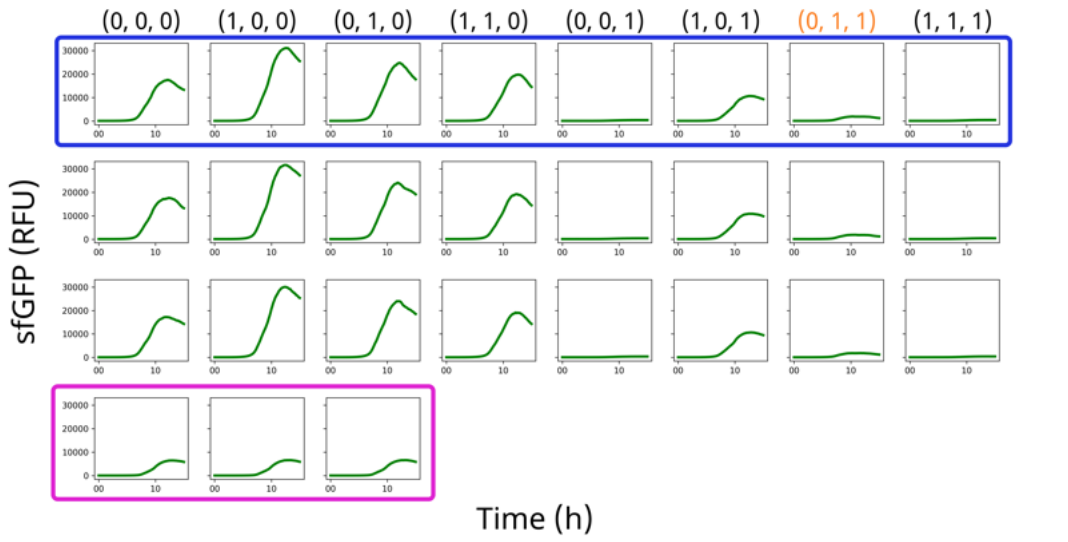


Figure 6.3: Plate reader fluorescence measurement of circuit 0xF6. The fluorescence is depicted in RFU over time in hours. The first three rows present the measurement results for the three biological replicate, with the results for biological replicate #1 highlighted in blue. The three measurements highlighted in pink represent the three biological replicates of the reference fluorescence used to calculate the RFU.

The results of the plate reader analysis align with the expected function of the circuit. Six

out of the eight states exhibit a high output signal, while two show a low output signal. Notably, as seen visually, there are variations in the strengths of the on signals among the states. Particularly, state IPTG, aTc, Ara = (1, 0, 0) displays a very strong on signal, while state (0, 1, 1) highlighted in orange, exhibits a notably weaker on signal.

Figure 6.4 illustrates the derivative of the fluorescence analysis, representing the sfGFP production rate in RFU^{-1} $Cell^{-1}$ over time in hours. The layout mirrors that of Figure 6.3, with the first three rows displaying the three biological replicates and the last row presenting the reference fluorescence.

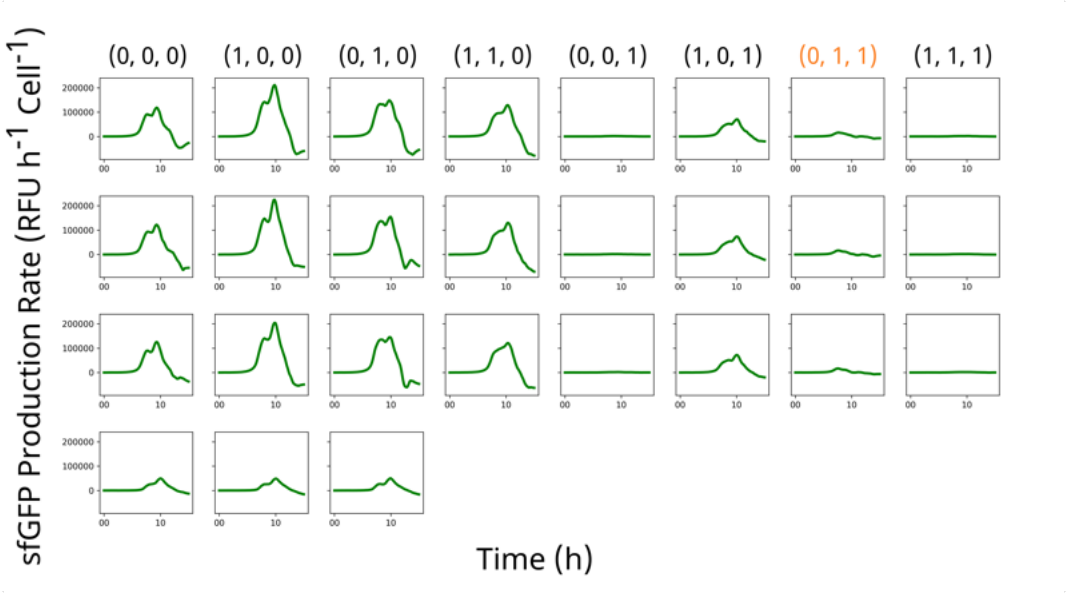


Figure 6.4: GFP production rate of circuit 0xF6. The production rate is depicted in RFU^{-1} $Cell^{-1}$ over time in hours. The first three rows display the measurement results for the three biological replicates. The last row shows the three measurements of the reference fluorescence used to calculate the RFU.

The sfGFP production rate exhibits a consistent pattern. Specifically, state (1, 0, 0) demonstrates robust sfGFP production in comparison to the other on states. This state expresses a higher concentration of sfGFP and therefore measures a higher fluorescence. Contrary, state (0, 1, 1) shows limited sfGFP production, contributing directly to the observed difference in fluorescence.

6.1.2 Experimental Monitoring of Cell Growth through OD Analysis

Differences in fluorescence levels may not necessarily correspond to distinct levels of circuit activity. Instead of indicating lower circuit activity, variations in cell quantity can also influence fluorescence levels. Therefore, alongside the fluorescence analysis, OD was measured as a reliable method for monitoring cell growth. This measurement aims to determine whether the variances observed in Figure 6.3 are attributed to fluctuations in cell concentration.

The typical bacterial growth in a pure culture is visualized in Figure 6.5. This growth is characterized by several distinctive phases, namely the *lag* phase, the *log* (or *exponential*) phase, the *stationary* phase, and the *decline* (or *death*) phase [142].

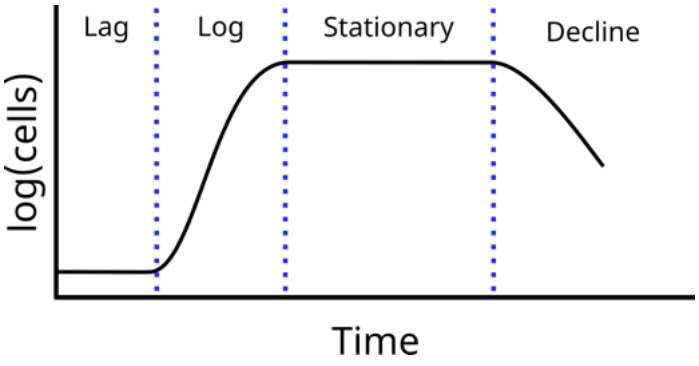


Figure 6.5: Typical bacterial growth curve in a pure culture. The graph depicts the logarithmic plot of the number of cells over time. Bacterial growth is characterized by distinctive phases. It begins with the lag phase, during which bacterial growth is nearly zero as cells adapt to the media. The subsequent log phase represents exponential bacterial growth, ultimately leading to the stationary phase marked by an equilibrium between cell growth and death. In the final decline, the number of cells dying surpasses the number of cells growing.

The lag phase is characterized by a nearly zero growth rate [142]. During this phase, cells adapt to the new media, involving the expression of specific mRNA and protein synthesis essential for the cell's survival. The duration of the lag phase can vary from minutes to hours, depending on factors such as the bacterial strain, initial cell count, and the composition of the media. Once the initial population has doubled, the lag phase transitions into the log phase [232]. The log phase is marked by exponential growth of the cell culture, therefore also often referred to as the exponential phase. Bacteria multiply rapidly, with the growth rate being proportional to the number of cells present at any given time [142]. The end of the log phase leads to the stationary phase, characterized

by a net growth rate of zero. During this phase, an equilibrium is reached between the growth and death of bacteria. Factors contributing to zero net growth include the depletion of essential nutrients in the media, hindering optimal growth and eventually leading to the decline phase [142]. The final phase, known as the decline phase, involves a net loss of bacterial cells. While there may still be some cell growth, the number of cells dying surpasses the number of new cells through division. This phase is typically exponential but slower than the log phase with depletion of essential resources being the driving factor [142].

Figure 6.6 illustrates the results of the measurement of the bacteria's growth through OD analysis. The figure layout remains consistent, with the first three rows presenting biological replicates #1 to #3, and the columns displaying all eight states. As before, the last row depicts the reference cultures. OD is measured by quantifying the light that passes through the sample and is dimensionless. In this work, light with a wavelength of 700 nm was employed.

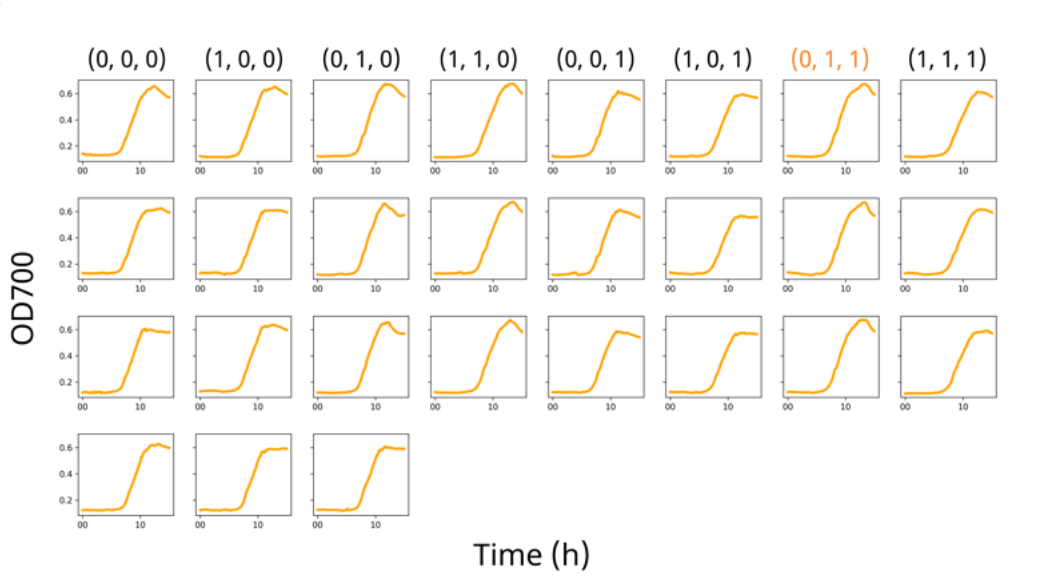


Figure 6.6: The OD is depicted at a wavelength of 700 nm over time in hours. The first three rows display the measurement results for the three biological replicates #1 to #3. The last row represents the three biological replicates of the reference cultures. The growth of the circuit containing cell cultures is comparable for all eight states across the three biological replicates. Furthermore, the cultures expressing the reference fluorescence also show comparable growth.

The OD analysis results suggest comparable growth of cell cultures for all eight states. There

is no indication that one of the states had a higher concentration of cells than the others, reinforcing the notion that the fluorescence difference is likely due to variations in sfGFP concentration rather than cell concentration. To further confirm these results, the specific growth rate (μ) of the cells is presented in Figure 6.7. The specific growth rate, derived from optical density, is depicted in the same layout over time in hours.

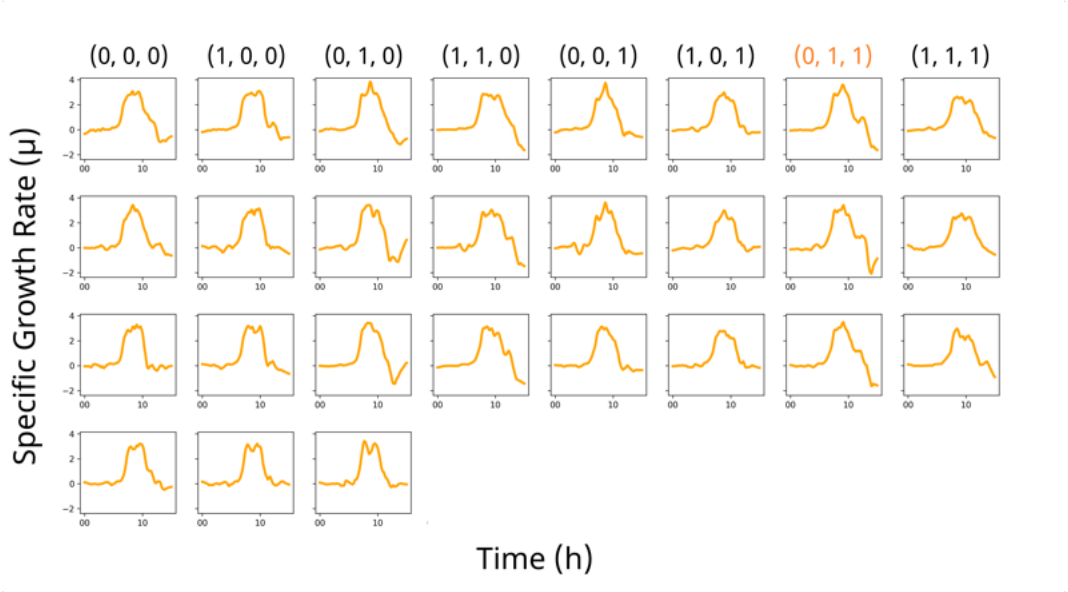


Figure 6.7: Measurement of the specific growth rate for circuit 0xF6. The specific growth rate is presented over time in hours at a wavelength of 700 nm. The first three rows showcase the measurement results for the three biological replicates, with the outcomes for biological replicate #1 depicted in the first row. The last row illustrates the three measurements corresponding to the biological replicates used as reference.

Once again, the results confirm that there is no significant difference in the specific growth rate among the various states. In all cell cultures, the growth rate initiates at a low level, the lag phase, increases during the log phase, reaches its peak, slows down in the stationary phase, and eventually drops below zero in the decline phase. Therefore, the conclusion drawn is that cell growth does not contribute to the variation in the strength of the on state. The observed differences can be attributed to the behavior of the circuit and sfGFP production itself.

6.2 Computational Analysis of Circuit 0xF6

Following the experimental analysis of circuit 0xF6, a computational model was created and examined to elucidate the observed behavior. Various types of computational analyses were employed. First, ODE analysis was conducted to identify potential glitching behavior. Second, SSA analysis was carried out to assess the probabilities of the observed faulty behavior occurring.

6.2.1 ODE Analysis of Circuit 0xF6

The computational model of circuit 0xF6 conforms to the default iBioSim model with default parameters, as introduced in Chapter 5. The ODE analysis unveiled distinct behaviors among the states, leading to the identification of three states of particular interest. The first selected state is IPTG, aTc, Ara = (0, 0, 0), illustrated in the first column of Figure 6.3. The second selected state is (1, 0, 0), corresponding to the second column in Figure 6.3, and finally, the third state is (0, 1, 1), corresponding to the seventh column in Figure 6.3. The results of the ODE analysis are depicted in Figure 6.8. The analysis employed the Runge-Kutta method with a time limit of 2100 time points.

In this figure, the sfGFP production over time for the state (0, 0, 0) is depicted as the solid blue line. The dashed blue line indicates when the circuit reaches an output of 60 molecules, occurring at time step 260. As defined in previous chapters, a high signal is considered to be at 60 molecules and above. Next, the purple line illustrates the sfGFP production over time for the input configuration (1, 0, 0). Again, the purple dashed line indicates when the circuit reaches an output of 60 molecules, achieved after 152 time units—faster than for state (0, 0, 0). It is noteworthy that the analysis predicts the circuit to produce twice as much sfGFP for this state compared to the other states, explaining the stronger on signal. Finally, the state with the weaker on signal (0, 1, 1) is depicted by the orange graph. Similar to the other two states, the dashed orange line indicates when the circuit reaches an output of 60 molecules for that transition, requiring 574 time steps. Consequently, this state achieves a high signal much later than the other states. The delay is attributed to a dynamic glitch, visible in the figure. While the simulation suggests eventual attainment of the high output state, the extended duration results from non-monotonic production. Hence, the hypothesis posits

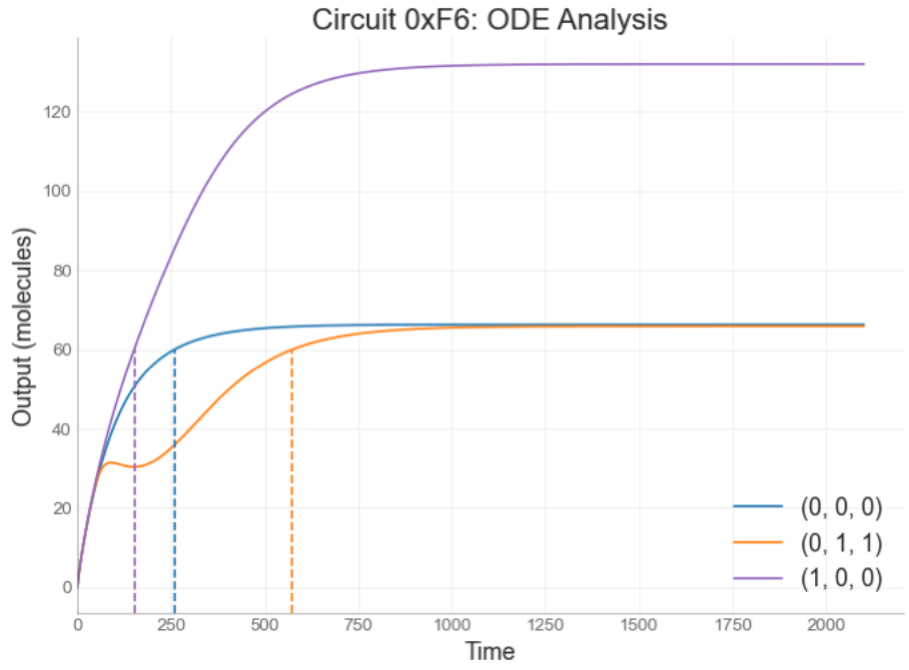


Figure 6.8: ODE analysis of circuit 0xF6. The figure presents the ODE analysis results depicting the number of output molecules over time for three states of circuit 0xF6. The circuit initiates from its initial setup, not in its steady state (0, 0, 0), and transitions to the states IPTG, aTc, Ara = (0, 0, 0), (0, 1, 1), and (1, 0, 0). Dashed lines indicate when the circuit reaches an output of 60 molecules for the respective state. The results highlight that the state with the observed lower fluorescence exhibits a dynamic glitch, causing a delayed attainment of the output of 60 molecules compared to the other two states.

that the dynamic failure influences steady-state behavior, introducing a race condition between the bacteria reaching their proper steady state and their lifecycle.

6.2.2 SSA Analysis of Circuit 0xF6

Following the ODE analysis, stochastic simulation was employed to estimate the probability of the circuit reaching its correct steady state within the time constraints imposed by the cell lifecycle. For this purpose, 10,000 SSA runs were performed to simulate the behavior and determine whether the circuit achieves 60 molecules within 260 time units. The choice of the 260-time-unit threshold was based on the duration required for the transition (0, 0, 0) to reach 60 molecules. The results

are presented in Figure 6.9. The bar graph depicts the six on states on the x-axis, illustrating the probability of each state reaching 60 molecules within 260 time points as a percentage on the y-axis.

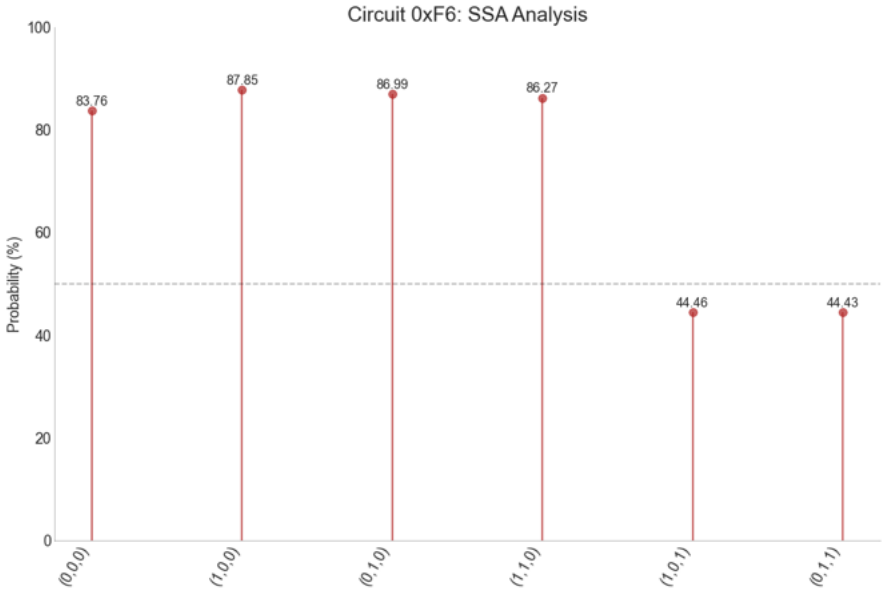


Figure 6.9: SSA analysis of circuit 0xF6. The figure illustrates the probability of each of the six states with a high output reaching 60 output molecules within 260 time units. All states that visually exhibited a high output state have a probability of reaching the expected output above 80 percent. In contrast, the states showing a dynamic glitch in the ODE analysis have a probability of below 50 percent of reaching the expected high output in time.

The results reveal that for the four on states that are clearly visible, they consistently achieve 60 molecules in over 80 percent of the runs within the specified time frame. Conversely, the two states that appear dimmer and exhibit a dynamic glitch in the ODE simulation only reach 60 molecules in less than 50 percent of the runs. It is important to note that the two off states are not visualized, as they remain off throughout the transition. As anticipated, with more time, the states stabilize, increasing the likelihood of all states reaching the correct on state. Figure 6.10 presents the same analysis, this time illustrating the probability of each on state reaching 60 molecules of output signal before 600 time units.

As depicted, the two states that previously exhibited faulty behavior and had a low probability of reaching the correct state now demonstrate an increased likelihood, surpassing 50 percent, of

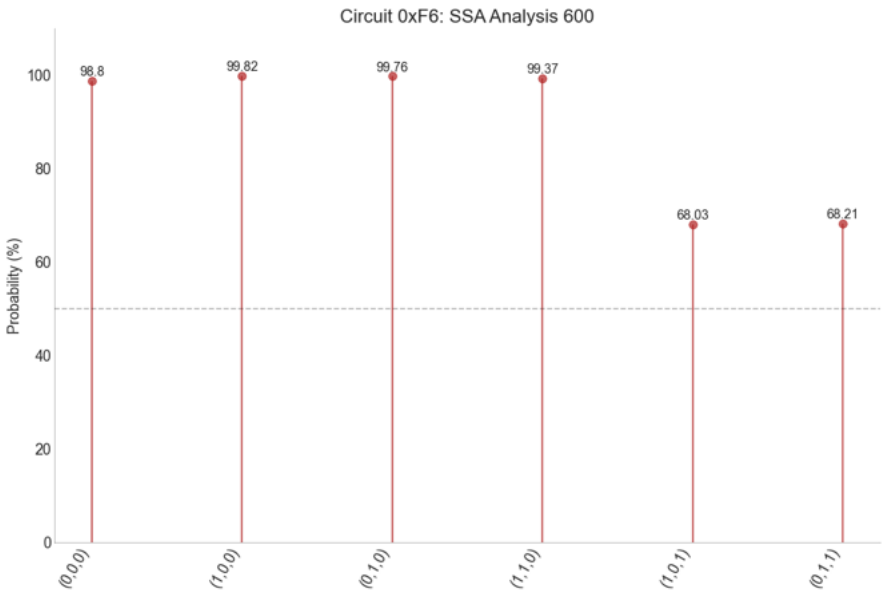


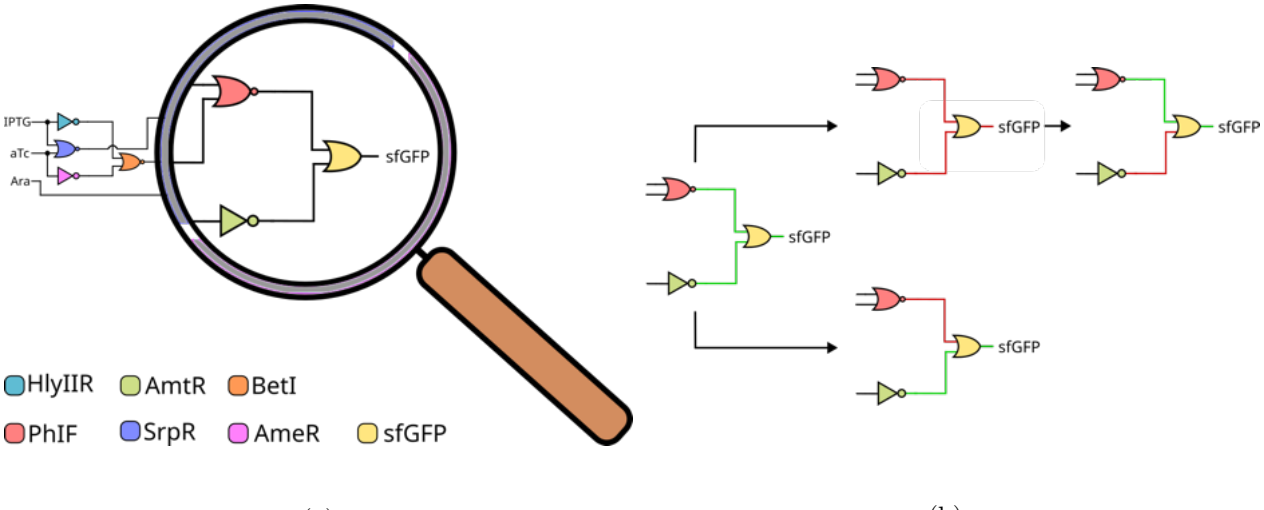
Figure 6.10: SSA analysis of circuit 0xF6. This figure illustrates the probability of each of the six states with a high output reaching 60 output molecules within 600 time units. States that visually exhibited a high output have a probability exceeding 80 percent of reaching the expected output. In contrast, states exhibiting a dynamic glitch in the ODE analysis have a probability below 50 percent of reaching the expected high output within the specified time.

achieving their on signal within the designated time. Therefore, given sufficient time, cells can eventually reach the correct steady state. However, cells have a finite lifespan and cannot sustain the exponential growth phase indefinitely. Extending the exponential growth phase by providing additional nutrients throughout this period allows for more time for the circuit to reach its correct steady state in a real-world application.

6.3 Identification of Circuit 0xF6's Glitch Cause

Having identified the dynamic failure influencing the circuit's steady state and analyzed its probability, the next step involves investigating the failure's root cause. A closer examination reveals that all states with a low Ara input reach their high output signal on time. In contrast, the two states that fail to reach a high output in time have a high Ara input. Figure 6.11 highlights circuit 0xF6, specifically focusing on the last three gates, which are based on the PhIF and AmtR

expression cassettes, as well as the sfGFP output OR gate.



(a) (b)

Figure 6.11: Identification of the glitch cause in circuit 0xF6. In (a), circuit 0xF6 and its associated logic gates are depicted, with emphasis on the three gates responsible for the circuit's faulty behavior. In panel (b), the temporal sequence and signal propagation through the circuit unfold, providing insights into the underlying cause of the dynamic glitch observed in the circuit's output. The four states successfully achieving a timely high output exhibit a low Ara input signal. This signal, traversing a single NOT gate, activates the output OR gate, leading to output production independently of the IPTG and aTc pathways. In contrast, the two high-output states exhibiting the dynamic glitch rely on the IPTG and aTc pathway to activate the final OR gate. The necessity for signals to navigate multiple layers of logic introduces a delay in output production, giving rise to the observed dynamic glitching behavior.

The sfGFP output is regulated by an OR gate, indicating that the output will be active if either or both of the PhIF or AmtR cassettes are active. Figure 6.11 visually represents the signal of the gates over time, using a red line to indicate a low signal and a green line to signify a high signal. Cello components operate on repression, so initially, after cell transformation and induction, all promoters are active since no repressors have been produced yet. As time progresses, the signal propagates through various logic levels, stabilizing the circuit in its output. All states that achieve their high output as expected follow the lower pathway depicted in the figure. In this pathway, the circuit initiates with all signals on initially, and sfGFP production remains uninterrupted, given that the lower pathway, the inverter Ara signal, never turns off, aligning with the observed behavior both computationally and experimentally.

For the two states exhibiting a dimmer on signal, the circuit also initiates with all signals on. However, in these instances, the inverter turns off due to the high Ara signal, as illustrated in the upper part of Figure 6.11 (b). Since the Ara pathway is shorter than the pathway through the PhIF gate, the signal turns on before the PhIF NOR gate reactivates. Consequently, the circuit experiences a brief interruption in sfGFP production, as indicated in the computational analysis, offering a potential explanation for the circuit's observed behavior in the laboratory.

6.4 Modified Experiment of Circuit 0xF6

Based on the hypothesis outlined earlier, the plate reader experiment was replicated using the identical circuit 0xF6 and output plasmid, again with the output plasmid containing sfGFP instead of YFP. Prior to the experiment, cells were cultured overnight in M9 media supplemented with appropriate antibiotics and subsequently transferred to a 96-well plate. The cells were then induced with inputs representing the eight possible states, and both fluorescence and OD were measured over a span of 50 hours compared to the 15 hours of the initial experiment. A detailed protocol is provided in the appendix for reference. The results, as illustrated in Figure 6.12, depict normalized fluorescence relative to OD. For normalization, fluorescence values were divided by their corresponding OD values. To mitigate the influence of small or negative OD values at the outset, likely attributed to measurement noise, a threshold was implemented to filter out such data points. As before, the data from three independent biological replicates are included in the figure.

The analysis results appear to confirm a dynamic glitch in the state where IPTG, aTc, and Ara are in the on configuration (0, 1, 1) shown in panels A07, B07, C07, and D07, thus supporting the hypothesis. However, it is noteworthy that most other on states, including (0, 0, 0), (1, 0, 0), (0, 1, 0), and (1, 1, 0), also exhibit a dynamic glitch. Nevertheless, these states demonstrate a quicker, more pronounced initial output production, with the dynamic glitch occurring later compared to the state of primary interest. Additionally, the results indicate that state (0, 0, 1), shown in panels A05, B05, C05, D05, remains consistently low throughout the experiment, while the other off state, (1, 1, 1), unexpectedly transitions to an on state instead of maintaining its anticipated low output.

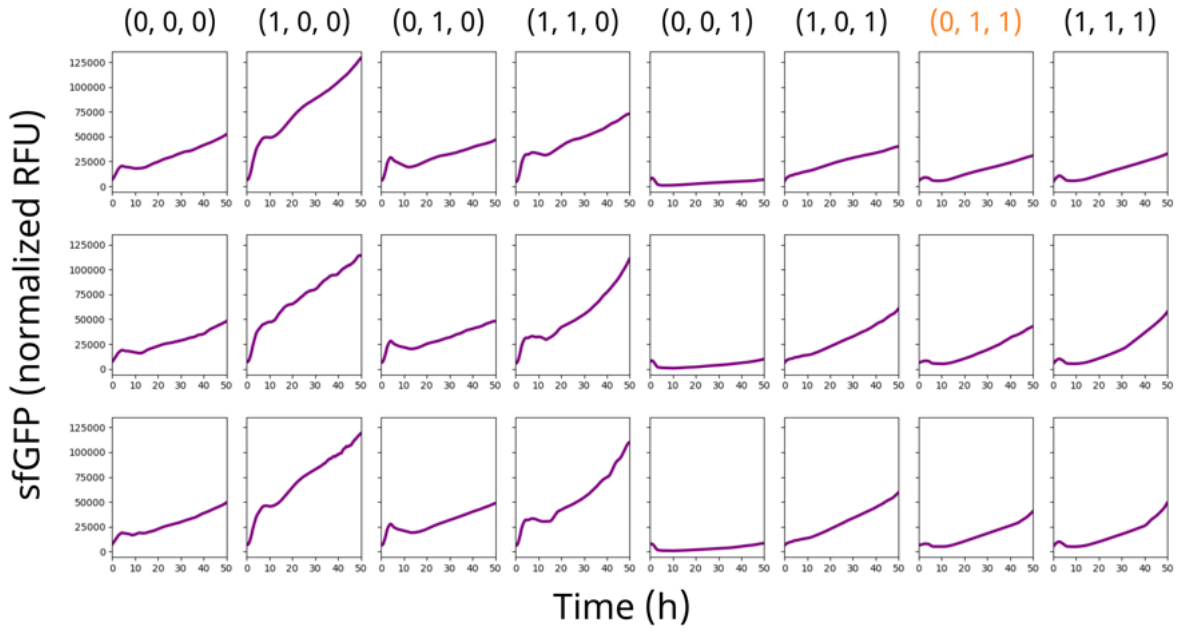


Figure 6.12: Fluorescence analysis of circuit 0xF6 over 50 hours. The figure displays fluorescence data normalized to OD for three biological replicates and all eight states of circuit 0xF6. Notably, the critical state (0, 1, 1), highlighted in orange, reveals a dynamic glitch, a phenomenon observed in several other states as well. Intriguingly, the final state appears to turn on despite its intended status to remain off.

Several factors could contribute to this phenomenon. State 8 relies on the presence of all three inducers, whereas the other off state depends solely on the inducer Ara. Consequently, it is plausible that degradation of either IPTG or aTc, the other two inducers, occurs more rapidly, leading to a state transition. This hypothesis could similarly account for the behavior observed in other states. Further experiments are warranted to validate this theory.

One potential approach involves conducting RNA-seq experiments to assess mRNA concentrations within the cell. Alternatively, microscopic analysis could enable tracking of molecule concentrations in real-time. Finally, employing different circuit functions and conducting multiple plate reader experiments could help elucidate the lifespan of the inducers.

6.5 Discussion

In the context of this dissertation, this chapter delves into the exploration of dynamic failure behavior, a facet previously overlooked in earlier analyses, employing circuit 0xF6 as a case study. Motivated by an unexpected observation during flow cytometry analysis, which prompted a more comprehensive computational examination introduced in preceding chapters, the chapter formulated its central hypothesis: The lower output observed in the state of interest can be attributed to a race condition between the delayed output production caused by a dynamic glitch and the lifespan of the cells. The computational analysis unearthed a dynamic glitch in the state of interest, prompting a re-run of the plate reader analysis over an extended time course to experimentally capture this dynamic glitch. Subsequent analysis revealed that other states also exhibit behavior divergent from the expected output, leading to a novel hypothesis: variations in the life-span of the inducers may precipitate state transitions, thereby inducing different output behaviors.

"Well, here at last, dear friends, on the shores of the sea comes the end of our fellowship in middle-earth. Go in peace! I will not say: do not weep for not all tears are an evil."

- Gandalf the White

7

Conclusion

Over the past two decades, synthetic biology has witnessed significant advancements [4, 73]. Efforts have been made to transition from using E. coli as the primary host organism to exploring other organisms like fungi and plants [147, 130]. Additionally, there has been exploration into dividing larger projects into multiple cells [91], although these endeavors remain challenging. Despite progress, the complexity of genetic circuits continues to lag behind that of genetic engineering technologies [114, 207]. As the field continues to evolve, new challenges and bottlenecks arise, necessitating ongoing attention to ensure its sustained growth.

At the core of these challenges lies the imperative to design robust genetic circuits with behavior predictable through *in silico* analysis. The ability to predict the behavior of genetic cir-

cuits, regardless of their host organisms or construction methods, is essential for both laboratory research and real-world applications. Achieving this goal requires thorough part characterization and community-wide standardization efforts, alongside the development of software tools and automation methods to facilitate the design of larger genetic circuits.

This dissertation contributes to addressing these challenges by providing insights into genetic circuit behavior and the relationship between computational and experimental analyses. Within this work, critical aspects of genetic circuit design have been explored, paving the way for future advancements in the field. Looking ahead, bridging the gap between computational and laboratory synthetic biology stands as a crucial task for future progress. Future research directions will focus on further integrating computational and experimental analyses, ultimately advancing our understanding of genetic circuits and their applications. This chapter serves as the conclusion to the dissertation, summarizing the main contributions of the research in Section 7.1. It also outlines future directions for this research in Section 7.2.

7.1 Summary

This dissertation introduces methodologies for deciphering genetic circuit failures in the field of synthetic biology. To achieve real-world impact, robust and reliable genetic circuits are indispensable. This necessitates a comprehensive understanding of faulty behavior and the application of reliable analysis methods. The presented work first establishes a definition for faulty genetic circuit behavior to allow further reproducible analysis. This work continues by introducing various analysis methods. These methods, including ODE analysis, stochastic simulations, and stochastic model checking, serve to simulate and verify the expected genetic circuit behavior. Each method brings distinct advantages, with ODE analysis providing quantitative insights, stochastic simulations capturing non-deterministic behavior, and stochastic model checking offering accurate probabilities.

The consideration of abstraction levels emerges as a crucial factor in balancing accuracy and computational complexity. Therefore, this research assesses the impact of different abstraction levels on computational models. While investing effort in characterizing genetic parts enhances model

predictions, resource constraints may limit the feasibility of comprehensive characterization. More abstracted models, involving modifications to chemical reactions and parameters derived from literature, offer a practical compromise for users to understand their genetic design's behavior. However, it is essential to note that higher modeling precision corresponds to increased computational costs.

The application of these methodologies is demonstrated through sample studies on genetic circuits. The initial analysis focuses on comparing different computational models by evaluating failure probabilities in static transitions and steady-state failures. One critique of this analysis is the exclusion of dynamic failures often deemed irrelevant under the assumption that the circuit eventually reaches the correct state with a delay. However, both experimental and computational analyses presented at the end of this work suggest otherwise. The work evaluates the previously overlooked dynamic hazards both experimentally and computationally. The results prompt a reevaluation of the hypothesis, emphasizing the need to explore the significance of dynamic behavior in influencing steady states.

The dissertation emphasizes the benefits of integrating engineering principles into synthetic biology. Throughout this research, standards have played a crucial role in enabling the use of software tools for predicting genetic circuit behavior and ensuring reproducibility in the future. Moreover, a converter was developed, leveraging the SBOL standard, to facilitate translation to the PRISM language, thereby enabling stochastic model analysis. Abstraction is integral in the entire workflow and has been extensively covered in Chapter 5. Finally, the concept of decoupling is used in Chapter 6 illustrating the separation of the analysis phase from laboratory work, further underscoring the value of systematically engineering genetic designs.

7.2 Future Work

The narrative presented in this dissertation underscores the value of computational analysis in understanding failure behavior in genetic circuits. Employing diverse analysis methods provides insights into a GRN's behavior from different perspectives. Even when information about a specific design is limited, such as when utilizing novel genetic parts, default parameters and established

methods can still furnish users with valuable results. Furthermore, experimental data allows for the validation of model predictions, feeding back into the DMABTL cycle and resulting in better and more predictive models for future applications.

However, challenges persist, and a gap remains between the computational and laboratory-based aspects of the field. Many biologists currently avoid computational analysis methods, as common designs are still simple enough to be manually created. Closing the gap between computational and laboratory synthetic biology necessitates the development of practical and enhanced computational tools that streamline the design process, automate workflows, and optimize genetic design. Updated laboratory procedures are needed to aid in model predictability and achieve faster turnaround times for real-world applications. This includes the design and characterization of larger, more complex constructs in a shorter timeframe using computational methods and automation. Ultimately, the goal is to make genetic design readily accessible and efficient for researchers in the field.

7.2.1 Failure Definition

This dissertation initially defines incorrect genetic circuit behavior as the manifestation of unwanted output behaviors. These failures encompass transient failures, which self-correct over time, and steady-state failures. Furthermore, this work explains how these undesired behaviors can arise, whether from failures on a cellular level or from a circuit's logic implementation or Boolean function.

However, the primary contention posited is that all these failures manifest through unwanted outputs. Future research endeavors could explore more sophisticated observation methods beyond output behavior to pinpoint failure causes with greater precision, whether stemming from crosstalk, roadblocking, or toxicity. Enhanced understanding facilitates more precise adjustments and opens up multiple new avenues for fine-tuning genetic circuits.

7.2.2 Computational Progress

Continuous updates are essential for GDA tools to remain aligned with evolving analysis methods, enabling designers to effectively leverage newly developed features. A valuable feature could involve an automated, streamlined analysis process, akin to the method detailed in Chapter 5. A user-friendly software tool could take a user-defined design, generate multiple versions based on characterization data, and automatically conduct analysis to pinpoint designs with the highest likelihood of success.

Moreover, this dissertation explores various analysis methods, including ODE, SSA, and stochastic model verification. Driving this work further, available paths for exploration include possibilities for hybrid approaches and utilization in different contexts, such as using stochastic model checking to explore the design space and guide the user toward different, more robust implementations of the desired function.

Furthermore, as genetic designs grow in complexity, the integration of software assistance becomes increasingly critical. Establishing a reliable analysis approach is imperative for the robust design of multi-cellular systems. This entails refining the methods outlined in this dissertation to effectively accommodate the intricacies of multi-cellular systems.

7.2.3 Future of Standards

The work in this dissertation relied on a variety of standards, which play a crucial role in ensuring the reproducibility of results and facilitating data sharing. However, the adoption of standards can be hindered by personal interests and ad hoc practices within research groups, posing a significant bottleneck to progress. Overcoming this challenge often requires new standards to add value to research endeavors without imposing excessive implementation costs. Achieving widespread acceptance of standards within the community is essential to enhance data sharing and ensure the reproducibility of published results [149]. Standardization efforts should extend beyond laboratory procedures to encompass data sharing, visualization, and mathematical modeling, facilitating seamless communication among diverse research groups [19].

Community-developed standards, such as SBOL [37, 151, 138, 189, 83], SBOL Visual [11, 185], and SBML [115, 105] utilized in this work, should take precedence over bioinformatic standards and non-standard formats. This preference is critical for promoting the sharing of comprehensive design information among collaborators. Part libraries should adhere to these standards, and software tools must offer support to enhance the capacity of laboratory-based scientists to share data and enable computational-based scientists to develop integrated software workflows. Additionally, existing standards should continuously evolve, and new standards should be developed as necessary, through open discourse involving both computational and laboratory-based researchers, as well as representatives from academia and industry. This collaborative approach ensures that standards remain relevant and effective in addressing the evolving needs of the synthetic biology community [38].

7.2.4 Genetic Part Characterization and Screening Methods

Facilitating plug-and-play functionality for genetic designs requires the establishment of data sheets akin to those used in electronic circuit designs. Enhancements in characterization methods are pivotal in this regard, mandating the adoption of high-throughput screening techniques, as demonstrated in [78, 93]. These innovative methods should streamline the collection and analysis of extensive datasets, ultimately improving the accuracy of part characterization.

Additional parameters inferred from experimental data can enhance simulations by incorporating factors such as toxicity, media composition, environmental conditions, and the influence of neighboring bacteria. Moreover, developing host-specific simulation methods can streamline the modeling process and improve accuracy.

7.2.5 Fitting of Computational Models

Experimental data, as demonstrated in Chapter 6, serves a crucial role in validating and verifying model results. Furthermore, this data can be integrated back into the iterative DMABTL cycle to refine and improve models, ensuring they align closely with experimental observations. In this dissertation, the values for high and low signals were chosen arbitrarily within the parameter

range, facilitating predictions that could potentially be validated experimentally. To precisely estimate the actual number of molecules in the cell, methods such as RNA sequencing can be utilized to fine-tune the high and low values accordingly.

Fitting models to experimental data not only enhances the understanding of the model but also sheds light on the underlying biological processes. However, this process comes with its own set of challenges. First, to justify the effort, it is essential to have protocols and preferably automation methods in place to obtain experimental data for model fitting in a timely manner. Furthermore, challenges arise concerning parameter identifiability and overfitting. Lastly, the quality of the experimental data must be of high caliber to ensure reliable model fitting outcomes.

In the context of this dissertation, conducting characterization experiments on circuit 0x8E could offer valuable insights into exploring the differences in the models and abstractions discussed in Chapter 5. This exploration would help determine whether each model yields the same conclusions and can be effectively utilized. Additionally, characterizing this circuit would establish a ground truth for comparing the models, providing a benchmark against which their predictive capabilities can be assessed.

Furthermore, data obtained from these experiments would enable a deeper investigation into the influence of intrinsic and extrinsic noise on circuit behavior. By highlighting which noise sources have the most significant impact, researchers can gain a better understanding of the underlying factors driving circuit performance. This heightened focus on noise sources would contribute to refining modeling approaches and improving the accuracy of predictive models.

7.2.6 Laboratory Workflow Automation

Automation in the laboratory plays a crucial role in reducing barriers to entry in the field. Notable automation tools include the OpenTrons [1] and Eppendorf [2] liquid handling robots. These devices streamline experimental workflows by automating various steps, thereby accelerating both the build and test processes. The benefits of automation are manifold.

[1] https://opentrons.com
[2] https://www.eppendorf.com/

First, automation significantly increases workflow efficiency, as robots can conduct experiments and execute protocols 24 hours a day, seven days a week. This continuous operation minimizes downtime and expedites research progress.

Second, automated systems are less prone to errors compared to manual processes, ensuring the reliability and reproducibility of experimental results. This reduction in errors contributes to higher-quality data and enhances the overall integrity of scientific findings.

Moreover, automation enhances the scalability of experiments, enabling researchers to conduct studies on a larger scale with minimal additional effort. This scalability is particularly advantageous for high-throughput screening studies and large-scale experiments.

Finally, automated protocols offer consistent performance, maintaining uniformity across experiments and eliminating variability between runs. Once set up and initialized, automated systems require minimal training to operate effectively, further streamlining laboratory procedures and increasing research productivity.

Bibliography

[1] Synthetic Biology: A Primer. Imperial College Press World Scientific Publishing Co. Pte. Ltd, revised edition edition.

[2] Uri Alon. An Introduction to Systems Biology: Design Principles of Biological Circuits. CRC press, 2019.

[3] Lauren B Andrews, Alec A K Nielsen, and Christopher A Voigt. Cellular checkpoint control using programmable sequential logic. SYNTHETIC BIOLOGY, page 12, 2018.

[4] Ernesto Andrianantoandro, Subhayu Basu, David K Karig, and Ron Weiss. Synthetic biology: New engineering rules for an emerging discipline. Molecular Systems Biology, 2(1):2006.0028, January 2006.

[5] Evan Appleton, Curtis Madsen, Nicholas Roehner, and Douglas Densmore. Design Automation in Synthetic Biology. Cold Spring Harbor Perspectives in Biology, 9(4):a023978, April 2017.

[6] Adam Arkin, John Ross, and Harley H McAdams. Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage λ-Infected Escherichia coli Cells. Genetics, 149(4):1633–1648, August 1998.

[7] Adam Paul Arkin. A wise consistency: Engineering biology for conformity, reliability, predictability. Current Opinion in Chemical Biology, 17(6):893–901, December 2013.

[8] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Verifying continuous time Markov chains. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Rajeev Alur, and Thomas A. Henzinger, editors, Computer Aided Verification, volume 1102, pages 269–276. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[9] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time Markov chains. ACM Transactions on Computational Logic, 1(1):162–170, July 2000.

[10] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. IEEE Transactions on Software Engineering, 29(6):524–541, June 2003.

[11] Hasan Baig, Pedro Fontanarossa, Vishwesh Kulkarni, James McLaughlin, Prashant Vaidyanathan, Bryan Bartley, Shyam Bhakta, Swapnil Bhatia, Mike Bissell, Kevin Clancy, Robert Sidney Cox, Angel Goñi Moreno, Thomas Gorochowski, Raik Grunberg, Jihwan Lee, Augustin Luna, Curtis Madsen, Goksel Misirli, Tramy Nguyen, Nicolas Le Novere, Zachary Palchick, Matthew Pocock, Nicholas Roehner, Herbert Sauro, James Scott-Brown, John T. Sexton, Guy-Bart Stan, Jeffrey J. Tabor, Logan Terry, Marta Vazquez Vilar, Christopher A. Voigt, Anil Wipat, David Zong, Zach Zundel, Jacob Beal, and Chris Myers. Synthetic biology open language visual (SBOL Visual) version 2.3. Journal of Integrative Bioinformatics, 0(0):20200045, June 2021.

[12] Hasan Baig and Jan Madsen. Simulation Approach for Timing Analysis of Genetic Logic Circuits. ACS Synthetic Biology, 6(7):1169–1179, July 2017.

[13] Hasan Baig and Jan Madsen. A top-down approach to genetic circuit synthesis and optimized technology mapping. In 9th International Workshop on Bio-Design Automation (Pittsburgh, PA), pages 1–2, 2017.

[14] Federico Barone, Francisco Dorr, and Luciano E Marasco. Design and evaluation of an incoherent feed-forward loop for an arsenic biosensor based on standard iGEM parts. Synthetic Biology, page 10, 2017.

[15] Bryan A. Bartley, Kiri Choi, Meher Samineni, Zach Zundel, Tramy Nguyen, Chris J. Myers, and Herbert M. Sauro. pySBOL: A Python Package for Genetic Design Automation and Standardization. ACS Synthetic Biology, 8(7):1515–1518, July 2019.

[16] Caleb J. Bashor and James J. Collins. Understanding Biological Regulation Through Synthetic Biology. Annual Review of Biophysics, 47(1):399–423, May 2018.

[17] Maxwell Bates, Joe Lachoff, Duncan Meech, Valentin Zulkower, Anaïs Moisy, Yisha Luo, Hille Tekotte, Cornelia Johanna Franziska Scheitz, Rupal Khilari, Florencio Mazzoldi, Deepak Chandran, and Eli Groban. Genetic Constructor: An Online DNA Design Platform. ACS Synthetic Biology, 6(12):2362–2365, December 2017.

[18] Jacob Beal. Biochemical complexity drives log-normal variation in genetic expression. Engineering Biology, 1(1):55–60, June 2017.

[19] Jacob Beal, Angel Goñi-Moreno, Chris Myers, Ariel Hecht, María del Carmen Vicente, Maria Parco, Markus Schmidt, Kenneth Timmis, Geoff Baldwin, Steffi Friedrichs, Paul Freemont, Daisuke Kiga, Elena Ordozgoiti, Maja Rennig, Leonardo Rios, Kristie Tanner, Víctor Lorenzo, and Manuel Porcar. The long journey towards standards for engineering biosystems: Are the Molecular Biology and the Biotech communities ready to standardise? EMBO reports, 21(5), May 2020.

[20] Jacob Beal, Ting Lu, and Ron Weiss. Automatic Compilation from High-Level Biologically-Oriented Programming Language to Genetic Regulatory Networks. PLoS ONE, 6(8):e22490, August 2011.

[21] Jacob Beal, Tramy Nguyen, Thomas E. Gorochowski, Angel Goñi-Moreno, James Scott-Brown, James Alastair McLaughlin, Curtis Madsen, Benjamin Aleritsch, Bryan Bartley, Shyam Bhakta, Mike Bissell, Sebastian Castillo Hair, Kevin Clancy, Augustin Luna, Nicolas Le Novère, Zach Palchick, Matthew Pocock, Herbert Sauro, John T. Sexton, Jeffrey J. Tabor,

Christopher A. Voigt, Zach Zundel, Chris Myers, and Anil Wipat. Communicating Structure and Function in Synthetic Biology Diagrams. ACS Synthetic Biology, 8(8):1818–1825, August 2019.

[22] Shimshon Belkin, Sharon Yagur-Kroll, Yossef Kabessa, Victor Korouma, Tali Septon, Yonatan Anati, Cheinat Zohar-Perez, Zahi Rabinovitz, Amos Nussinovitch, and Aharon J Agranat. Remote detection of buried landmines using a bacterial sensor. Nature Biotechnology, 35(4):308–310, April 2017.

[23] Steven A. Benner and A. Michael Sismour. Synthetic biology. Nature Reviews Genetics, 6(7):533–543, July 2005.

[24] Dennis A Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, James Ostell, Kim D Pruitt, and Eric W Sayers. GenBank. Nucleic Acids Research, 46(D1):D41–D47, January 2018.

[25] Frank T Bergmann, Richard Adams, Stuart Moodie, Jonathan Cooper, Mihai Glont, Martin Golebiewski, Michael Hucka, Camille Laibe, Andrew K Miller, David P Nickerson, Brett G Olivier, Nicolas Rodriguez, Herbert M Sauro, Martin Scharm, Stian Soiland-Reyes, Dagmar Waltemath, Florent Yvon, and Nicolas Le Novère. COMBINE archive and OMEX format: One file to share all information to reproduce a modeling project. BMC Bioinformatics, 15(1):369, December 2014.

[26] Jonathan A. Bernstein, Arkady B. Khodursky, Pei-Hsun Lin, Sue Lin-Chao, and Stanley N. Cohen. Global analysis of mRNA decay and abundance in Escherichia coli at single-gene resolution using two-color fluorescent DNA microarrays. Proceedings of the National Academy of Sciences, 99(15):9697–9702, July 2002.

[27] G. Bertani. STUDIES ON LYSOGENESIS I: The Mode of Phage Liberation by Lysogenic Escherichia coli. Journal of Bacteriology, 62(3):293–300, September 1951.

[28] Silpa Bhaskaran, Umesh P., and Achuthsankar S. Nair. Hill Equation in Modeling Transcriptional Regulation. In Vikram Singh and Pawan K. Dhar, editors, Systems and Synthetic Biology, pages 77–92. Springer Netherlands, Dordrecht, 2015.

[29] David Bikard, Wenyan Jiang, Poulami Samai, Ann Hochschild, Feng Zhang, and Luciano A. Marraffini. Programmable repression and activation of bacterial gene expression using an engineered CRISPR-Cas system. Nucleic Acids Research, 41(15):7429–7437, August 2013.

[30] Hamid Bolouri. Computational Modeling of Gene Regulatory Networks — A Primer. PUBLISHED BY IMPERIAL COLLEGE PRESS AND DISTRIBUTED BY WORLD SCIENTIFIC PUBLISHING CO., August 2008.

[31] J. Bonnet, P. Yin, M. E. Ortiz, P. Subsoontorn, and D. Endy. Amplifying Genetic Logic Gates. Science, 340(6132):599–603, May 2013.

[32] Olivier Borkowski, Francesca Ceroni, Guy-Bart Stan, and Tom Ellis. Overloaded and stressed: Whole-cell considerations for bacterial synthetic biology. Current Opinion in Microbiology, 33:123–130, October 2016.

[33] Benjamin J. Bornstein, Sarah M. Keating, Akiya Jouraku, and Michael Hucka. LibSBML: An API Library for SBML. Bioinformatics, 24(6):880–881, March 2008.

[34] Sierra M. Brooks and Hal S. Alper. Applications, challenges, and needs for employing synthetic biology beyond the lab. Nature Communications, 12(1):1390, March 2021.

[35] Jennifer A N Brophy and Christopher A Voigt. Principles of genetic circuit design. Nature Methods, 11(5):508–520, May 2014.

[36] J Brown. The iGEM competition: Building with biology. IET Synth. Biol., 1(1):4, 2007.

[37] Lukas Buecherl, Thomas Mitchell, James Scott-Brown, Prashant Vaidyanathan, Gonzalo Vidal, Hasan Baig, Bryan Bartley, Jacob Beal, Matthew Crowther, Pedro Fontanarrosa, Thomas Gorochowski, Raik Grünberg, Vishwesh Kulkarni, James McLaughlin, Göksel Mısırlı, Ernst Oberortner, Anil Wipat, and Chris Myers. Synthetic biology open language (SBOL) version 3.1.0. Journal of Integrative Bioinformatics, 20(1):20220058, March 2023.

[38] Lukas Buecherl and Chris J. Myers. Engineering Genetic Circuits: Advancements in Genetic Design Automation Tools and Standards for Synthetic Biology. Current Opinion in Microbiology, 2022.

[39] Lukas Buecherl, Riley Roberts, Pedro Fontanarrosa, Payton J. Thomas, Jeanet Mante, Zhen Zhang, and Chris J. Myers. Stochastic Hazard Analysis of Genetic Circuits in iBioSim and STAMINA. ACS Synthetic Biology, page acssynbio.1c00159, October 2021.

[40] D. Ewen Cameron, Caleb J. Bashor, and James J. Collins. A brief history of synthetic biology. Nature Reviews Microbiology, 12(5):381–390, May 2014.

[41] Barry Canton, Anna Labno, and Drew Endy. Refinement and standardization of synthetic biological parts and devices. Nature Biotechnology, 26(7):787–793, July 2008.

[42] Stefano Cardinale and Adam Paul Arkin. Contextualizing context for synthetic biology - identifying causes of failure of synthetic biological systems. Biotechnology Journal, 7(7):856–866, July 2012.

[43] Stefano Cardinale, Marcin Pawel Joachimiak, and Adam Paul Arkin. Effects of Genetic Variation on the E. coli Host-Circuit Interface. Cell Reports, 4(2):231–237, July 2013.

[44] Simeon D. Castle, Claire S. Grierson, and Thomas E. Gorochowski. Towards an engineering theory of evolution. Nature Communications, 12(1):3326, December 2021.

[45] Francesca Ceroni, Rhys Algar, Guy-Bart Stan, and Tom Ellis. Quantifying cellular capacity identifies gene expression designs with reduced burden. Nature Methods, 12(5):415–418, May 2015.

[46] D. Chandran, W.B. Copeland, S.C. Sleight, and H.M. Sauro. Mathematical modeling and synthetic biology. Drug Discovery Today: Disease Models, 5(4):299–309, December 2008.

[47] Deepak Chandran, Frank T Bergmann, and Herbert M Sauro. TinkerCell: Modular CAD tool for synthetic biology. Journal of Biological Engineering, 3(1):19, 2009.

[48] Deepak Chandran, Frank T. Bergmann, and Herbert M. Sauro. Computer-aided design of biological circuits using tinkercell. Bioengineered Bugs, 1(4):276–283, July 2010.

[49] Deepak Chandran and Herbert M. Sauro. Hierarchical Modeling for Synthetic Biology. ACS Synthetic Biology, 1(8):353–364, August 2012.

[50] Joanna Chen, Douglas Densmore, Timothy S Ham, Jay D Keasling, and Nathan J Hillson. DeviceEditor visual biological CAD canvas. Journal of Biological Engineering, 6(1):1, December 2012.

[51] Katherine C. Chen, Laurence Calzone, Attila Csikasz-Nagy, Frederick R. Cross, Bela Novak, and John J. Tyson. Integrative Analysis of Cell Cycle Control in Budding Yeast. Molecular Biology of the Cell, 15(8):3841–3862, August 2004.

[52] Ye Chen, Shuyi Zhang, Eric M. Young, Timothy S. Jones, Douglas Densmore, and Christopher A. Voigt. Genetic circuit design automation for yeast. Nature Microbiology, 5(11):1349–1360, November 2020.

[53] Ying-Ja Chen, Peng Liu, Alec A K Nielsen, Jennifer A N Brophy, Kevin Clancy, Todd Peterson, and Christopher A Voigt. Characterization of 582 natural and synthetic terminators and quantification of their design constraints. Nature Methods, 10(7):659–664, July 2013.

[54] Kiri Choi, J. Kyle Medley, Matthias König, Kaylene Stocking, Lucian Smith, Stanley Gu, and Herbert M. Sauro. Tellurium: An extensible python-based modeling environment for systems and synthetic biology. Biosystems, 171:74–79, September 2018.

[55] Natalie A Cookson, William H Mather, Tal Danino, Octavio Mondragón-Palomino, Ruth J Williams, Lev S Tsimring, and Jeff Hasty. Queueing up for enzymatic processing: Correlated signaling through coupled degradation. Molecular Systems Biology, 7(1):561, January 2011.

[56] M T Cooling, V Rouilly, G Misirli, J Lawson, T Yu, J Hallinan, and A Wipat. Standard Virtual Biological Parts: A Repository of Modular Model- ing Components for Synthetic Biology. page 8, 2010.

[57] Alan Costello and Ahmed H. Badran. Synthetic Biological Circuits within an Orthogonal Central Dogma. Trends in Biotechnology, 39(1):59–71, January 2021.

[58] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. Journal of the ACM, 42(4):857–907, July 1995.

[59] Francis Crick. Central Dogma of Molecular Biology. Nature, 227(5258):561–563, August 1970.

[60] Shixiu Cui, Xueqin Lv, Xianhao Xu, Taichi Chen, Hongzhi Zhang, Yanfeng Liu, Jianghua Li, Guocheng Du, Rodrigo Ledesma-Amaro, and Long Liu. Multilayer Genetic Circuits for Dynamic Regulation of Metabolic Pathways. ACS Synthetic Biology, 10(7):1587–1597, July 2021.

[61] M. J. Czar, Y. Cai, and J. Peccoud. Writing DNA with GenoCADTM. Nucleic Acids Research, 37(Web Server):W40–W47, July 2009.

[62] Ramiz Daniel, Jacob R. Rubens, Rahul Sarpeshkar, and Timothy K. Lu. Synthetic analog computation in living cells. Nature, 497(7451):619–623, May 2013.

[63] M Wayne Davis. ApE—a plasmid editor. Website, 2012.

[64] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A Storm is Coming: A Modern Probabilistic Model Checker. In Rupak Majumdar and Viktor Kunčak, editors, Computer Aided Verification, pages 592–600, Cham, 2017. Springer International Publishing.

[65] Domitilla Del Vecchio, Aaron J. Dy, and Yili Qian. Control theory meets synthetic biology. Journal of The Royal Society Interface, 13(120):20160380, July 2016.

[66] Anna Deplazes-Zemp. The Conception of Life in Synthetic Biology. Science and Engineering Ethics, 18(4):757–774, December 2012.

[67] Bryan S. Der, Emerson Glassey, Bryan A. Bartley, Casper Enghuus, Daniel B. Goodman, D. Benjamin Gordon, Christopher A. Voigt, and Thomas E. Gorochowski. DNAplotlib: Programmable Visualization of Genetic Designs and Associated Data. ACS Synthetic Biology, 6(7):1115–1119, July 2017.

[68] Qiang Ding and Chao Ye. Microbial cell factories based on filamentous bacteria, yeasts, and fungi. Microbial Cell Factories, 22(1):20, January 2023.

[69] H Dong, L Nilsson, and C G Kurland. Gratuitous overexpression of genes in Escherichia coli leads to growth inhibition and ribosome destruction. Journal of Bacteriology, 177(6):1497–1504, March 1995.

[70] Andreas Dräger, Nicolas Rodriguez, Marine Dumousseau, Alexander Dörr, Clemens Wrzodek, Nicolas Le Novère, Andreas Zell, and Michael Hucka. JSBML: A flexible Java library for working with SBML. Bioinformatics, 27(15):2167–2168, August 2011.

[71] Michael B. Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. Nature, 403(6767):335–338, January 2000.

[72] Michael B. Elowitz, Arnold J. Levine, Eric D. Siggia, and Peter S. Swain. Stochastic Gene Expression in a Single Cell. Science, 297(5584):1183–1186, August 2002.

[73] Lukas Endler, Nicolas Rodriguez, Nick Juty, Vijayalakshmi Chelliah, Camille Laibe, Chen Li, and Nicolas Le Novère. Designing and encoding models for synthetic biology. Journal of The Royal Society Interface, 6(suppl_4), August 2009.

[74] Drew Endy. Foundations for engineering biology. Nature, 438(7067):449–453, November 2005.

[75] Max A. English, Raphaël V. Gayet, and James J. Collins. Designing Biological Circuits: Synthetic Biology Within the Operon Model and Beyond. Annual Review of Biochemistry, 90(1):221–244, June 2021.

[76] V. Epshtein. Transcription through the roadblocks: The role of RNA polymerase cooperation. The EMBO Journal, 22(18):4719–4727, September 2003.

[77] Amin Espah Borujeni, Anirudh S. Channarasappa, and Howard M. Salis. Translation rate is controlled by coupled trade-offs between site accessibility, selective RNA unfolding and sliding at upstream standby sites. Nucleic Acids Research, 42(4):2646–2659, February 2014.

[78] Amin Espah Borujeni, Jing Zhang, Hamid Doosthosseini, Alec A. K. Nielsen, and Christopher A. Voigt. Genetic circuit characterization by inferring RNA polymerase movement and ribosome usage. Nature Communications, 11(1):5001, December 2020.

[79] Michael Florea, Henrik Hagemann, Gabriella Santosa, James Abbott, Chris N Micklem, Xenia Spencer-Milnes, Laura de Arroyo Garcia, Despoina Paschou, Christopher Lazenbatt, Deze Kong, Haroon Chughtai, Kirsten Jensen, Paul S Freemont, Richard Kitney, Benjamin Reeve, and Tom Ellis. Engineering control of bacterial cellulose production using a genetic toolkit and a new cellulose-producing strain. PNAS, page 10, 2016.

[80] Pedro Fontanarrosa. Investigating Genetic Circuit Failures. PhD thesis, University of Utah, August 2022.

[81] Pedro Fontanarrosa, Hamid Doosthosseini, Amin Espah Borujeni, Yuval Dorfan, Christopher A Voigt, and Chris Myers. Genetic Circuit Dynamics: Hazard and Glitch Analysis. ACS Synthetic Biology, page 15, 2020.

[82] Paul S Freemont and Richard I Kitney. Synthetic Biology — A Primer. IMPERIAL COLLEGE PRESS, July 2012.

[83] Michal Galdzicki, Kevin P Clancy, Ernst Oberortner, Matthew Pocock, Jacqueline Y Quinn, Cesar A Rodriguez, Nicholas Roehner, Mandy L Wilson, Laura Adam, J Christopher Anderson, Bryan A Bartley, Jacob Beal, Deepak Chandran, Joanna Chen, Douglas Densmore, Drew Endy, Raik Grünberg, Jennifer Hallinan, Nathan J Hillson, Jeffrey D Johnson, Allan Kuchinsky, Matthew Lux, Goksel Misirli, Jean Peccoud, Hector A Plahar, Evren Sirin, Guy-Bart Stan, Alan Villalobos, Anil Wipat, John H Gennari, Chris J Myers, and Herbert M Sauro. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. Nature Biotechnology, 32(6):545–550, June 2014.

[84] Miles W. Gander, Justin D. Vrana, William E. Voje, James M. Carothers, and Eric Klavins. Digital logic circuits in yeast with CRISPR-dCas9 NOR gates. Nature Communications, 8(1):15459, May 2017.

[85] Timothy S. Gardner, Charles R. Cantor, and James J. Collins. Construction of a genetic toggle switch in Escherichia coli. Nature, 403(6767):339–342, January 2000.

[86] P. Gaspar, J. L. Oliveira, J. Frommlet, M. A. S. Santos, and G. Moura. EuGene: Maximizing synthetic gene design for heterologous expression. Bioinformatics, 28(20):2683–2684, October 2012.

[87] Sina Ghaemmaghami, Won-Ki Huh, Kiowa Bower, Russell W. Howson, Archana Belle, Noah Dephoure, Erin K. O'Shea, and Jonathan S. Weissman. Global analysis of protein expression in yeast. Nature, 425(6959):737–741, October 2003.

[88] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry, 81(25):2340–2361, December 1977.

[89] Daniel T. Gillespie. Stochastic Simulation of Chemical Kinetics. Annual Review of Physical Chemistry, 58(1):35–55, May 2007.

[90] Emanuel Gonçalves, Joachim Bucher, Anke Ryll, Jens Niklas, Klaus Mauch, Steffen Klamt, Miguel Rocha, and Julio Saez-Rodriguez. Bridging the layers: Towards integration of signal transduction, regulation and metabolism into mathematical models. Molecular BioSystems, 9(7):1576, 2013.

[91] David T. Gonzales, Christoph Zechner, and T.-Y. Dora Tang. Building synthetic multicellular systems using bottom–up approaches. Current Opinion in Systems Biology, 24:56–63, December 2020.

[92] Thomas E. Gorochowski, Irem Avcilar-Kucukgoze, Roel A. L. Bovenberg, Johannes A. Roubos, and Zoya Ignatova. A Minimal Model of Ribosome Allocation Dynamics Captures Trade-offs in Expression between Endogenous and Synthetic Genes. ACS Synthetic Biology, 5(7):710–720, July 2016.

[93] Thomas E Gorochowski, Irina Chelysheva, Mette Eriksen, Priyanka Nair, Steen Pedersen, and Zoya Ignatova. Absolute quantification of translational regulation and burden using combined sequencing approaches. Molecular Systems Biology, 15(5), May 2019.

[94] F. Veronica Greco, Amir Pandi, Tobias J. Erb, Claire S. Grierson, and Thomas E. Gorochowski. Harnessing the central dogma for stringent multi-level control of gene expression. Nature Communications, 12(1):1738, December 2021.

[95] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. INFAMY: An Infinite-State Markov Model Checker. In Ahmed Bouajjani and Oded Maler, editors, Computer Aided Verification, volume 5643, pages 641–647. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[96] 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, 40(18):e141–e141, October 2012.

[97] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. Formal Aspects of Computing, 6(5):512–535, September 1994.

[98] F. C. Hartman, G. M. LaMuraglia, Y. Tomozawa, and R. Wolfenden. The influence of pH on the interaction of inhibitors with triosephosphate isomerase and determination of the pKa of the active-site carboxyl group. Biochemistry, 14(24):5274–5279, December 1975.

[99] Allison P. Heath and Lydia E. Kavraki. Computational challenges in systems biology. Computer Science Review, 3(1):1–17, February 2009.

[100] Matthias Heinemann and Sven Panke. Synthetic biology—putting engineering into biology. Bioinformatics, 22(22):2790–2799, November 2006.

[101] Nathan J. Hillson, Rafael D. Rosengarten, and Jay D. Keasling. J5 DNA Assembly Design Automation Software. ACS Synthetic Biology, 1(1):14–21, January 2012.

[102] C. Eric Hodgman and Michael C. Jewett. Cell-free synthetic biology: Thinking outside the cell. Metabolic Engineering, 14(3):261–269, May 2012.

[103] Ayaan Hossain, Eriberto Lopez, Sean M. Halper, Daniel P. Cetnar, Alexander C. Reis, Devin Strickland, Eric Klavins, and Howard M. Salis. Automated design of thousands of nonrepetitive parts for engineering stable genetic systems. Nature Biotechnology, 38(12):1466–1475, December 2020.

[104] Hsin-Ho Huang, Massimo Bellato, Yili Qian, Pablo Cárdenas, Lorenzo Pasotti, Paolo Magni, and Domitilla Del Vecchio. dCas9 regulator to neutralize competition in CRISPRi circuits. Nature Communications, 12(1):1692, December 2021.

[105] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, and the rest of the SBML Forum:, 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 Novere, 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, 19(4):524–531, March 2003.

[106] Linh Huynh, Athanasios Tsoukalas, Matthias Köppe, and Ilias Tagkopoulos. SBROME: A Scalable Optimization and Module Matching Framework for Automated Biosystems Design. ACS Synthetic Biology, 2(5):263–273, May 2013.

[107] Sonya V. Iverson, Traci L. Haddock, Jacob Beal, and Douglas M. Densmore. CIDAR MoClo: Improved MoClo Assembly Standard and New E. coli Part Library Enable Rapid Combinatorial Design for Synthetic and Traditional Biology. ACS Synthetic Biology, 5(1):99–103, January 2016.

[108] Dohyun Jeong, Melissa Klocke, Siddharth Agarwal, Jeongwon Kim, Seungdo Choi, Elisa Franco, and Jongmin Kim. Cell-Free Synthetic Biology Platform for Engineering Synthetic Biological Circuits and Systems. Methods and Protocols, 2(2):39, May 2019.

[109] Timothy S. Jones, Samuel M. D. Oliveira, Chris J. Myers, Christopher A. Voigt, and Douglas Densmore. Genetic circuit design automation with Cello 2.0. Nature Protocols, February 2022.

[110] Mads Kærn, William J. Blake, and J.J. Collins. The Engineering of Gene Regulatory Networks. Annual Review of Biomedical Engineering, 5(1):179–206, August 2003.

[111] Joanne Kamens. The Addgene repository: An international nonprofit plasmid and data resource. Nucleic Acids Research, 43(D1):D1152–D1157, January 2015.

[112] Guy Karlebach and Ron Shamir. Modelling and analysis of gene regulatory networks. Nature Reviews Molecular Cell Biology, 9(10):770–780, October 2008.

[113] M. Karnaugh. The map method for synthesis of combinational logic circuits. Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics, 72(5):593–599, 1953.

[114] Yiannis N Kaznessis. Models for synthetic biology. BMC Systems Biology, 1(1):47, December 2007.

[115] Sarah M Keating, Dagmar Waltemath, Matthias König, Fengkai Zhang, Andreas Dräger, Claudine Chaouiya, Frank T Bergmann, Andrew Finney, Colin S Gillespie, Tomáš Helikar, Stefan Hoops, Rahuman S Malik-Sheriff, Stuart L Moodie, Ion I Moraru, Chris J Myers, Aurélien Naldi, Brett G Olivier, Sven Sahle, James C Schaff, Lucian P Smith, Maciej J Swat, Denis Thieffry, Leandro Watanabe, Darren J Wilkinson, Michael L Blinov, Kimberly Begley, James R Faeder, Harold F Gómez, Thomas M Hamm, Yuichiro Inagaki, Wolfram Liebermeister, Allyson L Lister, Daniel Lucio, Eric Mjolsness, Carole J Proctor, Karthik Raman, Nicolas Rodriguez, Clifford A Shaffer, Bruce E Shapiro, Joerg Stelling, Neil Swainston, Naoki Tanimura, John Wagner, Martin Meier-Schellersheim, Herbert M Sauro, Bernhard Palsson, Hamid Bolouri, Hiroaki Kitano, Akira Funahashi, Henning Hermjakob, John C Doyle, Michael Hucka, SBML Level 3 Community members, Richard R Adams, Nicholas A Allen, Bastian R Angermann, Marco Antoniotti, Gary D Bader, Jan Červený, Mélanie Courtot, Chris D Cox, Piero Dalle Pezze, Emek Demir, William S Denney, Harish Dharuri, Julien Dorier, Dirk Drasdo, Ali Ebrahim, Johannes Eichner, Johan Elf, Lukas Endler, Chris T Evelo, Christoph Flamm, Ronan MT Fleming, Martina Fröhlich, Mihai Glont, Emanuel Gonçalves,

Martin Golebiewski, Hovakim Grabski, Alex Gutteridge, Damon Hachmeister, Leonard A Harris, Benjamin D Heavner, Ron Henkel, William S Hlavacek, Bin Hu, Daniel R Hyduke, Hidde Jong, Nick Juty, Peter D Karp, Jonathan R Karr, Douglas B Kell, Roland Keller, Ilya Kiselev, Steffen Klamt, Edda Klipp, Christian Knüpfer, Fedor Kolpakov, Falko Krause, Martina Kutmon, Camille Laibe, Conor Lawless, Lu Li, Leslie M Loew, Rainer Machne, Yukiko Matsuoka, Pedro Mendes, Huaiyu Mi, Florian Mittag, Pedro T Monteiro, Kedar Nath Natarajan, Poul MF Nielsen, Tramy Nguyen, Alida Palmisano, Jean-Baptiste Pettit, Thomas Pfau, Robert D Phair, Tomas Radivoyevitch, Johann M Rohwer, Oliver A Ruebenacker, Julio Saez-Rodriguez, Martin Scharm, Henning Schmidt, Falk Schreiber, Michael Schubert, Roman Schulte, Stuart C Sealfon, Kieran Smallbone, Sylvain Soliman, Melanie I Stefan, Devin P Sullivan, Koichi Takahashi, Bas Teusink, David Tolnay, Ibrahim Vazirabad, Axel Kamp, Ulrike Wittig, Clemens Wrzodek, Finja Wrzodek, Ioannis Xenarios, Anna Zhukova, and Jeremy Zucker. SBML Level 3: An extensible format for the exchange and reuse of biological models. Molecular Systems Biology, 16(8), August 2020.

[116] Jason R Kelly, Adam J Rubin, Joseph H Davis, Caroline M Ajo-Franklin, John Cumbers, Michael J Czar, Kim De Mora, Aaron L Glieberman, Dileep D Monie, and Drew Endy. Measuring the activity of BioBrick promoters using an in vivo reference standard. Journal of Biological Engineering, 3(1):4, 2009.

[117] Ahmad S. Khalil and James J. Collins. Synthetic biology: Applications come of age. Nature Reviews Genetics, 11(5):367–379, May 2010.

[118] M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Saldanha, and A. Taubin. Partial-scan delay fault testing of asynchronous circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17(11):1184–1199, Nov./1998.

[119] Hiroaki Kitano. Systems Biology: A Brief Overview. Science, 295(5560):1662–1664, March 2002.

[120] Joshua T Kittleson, Gabriel C Wu, and J Christopher Anderson. Successes and failures in modular genetic engineering. Current Opinion in Chemical Biology, 16(3-4):329–336, August 2012.

[121] Thomas F Knight. DARPA BioComp Plasmid Distribution 1.00 of Standard Biobrick Components, May 2002.

[122] Tom Knight. Idempotent Vector Design for Standard Assembly of Biobricks:. Technical report, Defense Technical Information Center, Fort Belvoir, VA, January 2003.

[123] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic Model Checking. In Marco Bernardo and Jane Hillston, editors, Formal Methods for Performance Evaluation, volume 4486, pages 220–270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[124] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, Computer Aided Verification, pages 585–591, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[125] Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic Model Checking: Advances and Applications. In Rolf Drechsler, editor, Formal System Verification, pages 73–121. Springer International Publishing, Cham, 2018.

[126] Roberta Kwok. Five hard truths for synthetic biology. Nature, 463(7279):288–290, January 2010.

[127] Ioannis Lestas, Johan Paulsson, Nicholas E. Ross, and Glenn Vinnicombe. Noise in Gene Regulatory Networks. IEEE Transactions on Automatic Control, 53(Special Issue):189–200, January 2008.

[128] Pietro Liò and Paolo Zuliani. Automated Reasoning for Systems Biology and Medicine. Springer, 2019.

[129] Chang C Liu, Lei Qi, Julius B Lucks, Thomas H Segall-Shapiro, Denise Wang, Vivek K Mutalik, and Adam P Arkin. An adaptor from translational to transcriptional control enables predictable assembly of complex regulation. Nature Methods, 9(11):1088–1094, November 2012.

[130] Wusheng Liu and C. Neal Stewart. Plant synthetic biology. Trends in Plant Science, 20(5):309–317, May 2015.

[131] Chunbo Lou, Brynne Stanton, Ying-Ja Chen, Brian Munsky, and Christopher A Voigt. Ribozyme-based insulator parts buffer synthetic circuits from genetic context. Nature Biotechnology, 30(11):1137–1142, November 2012.

[132] Yuan Lu. Cell-free synthetic biology: Engineering in an open world. Synthetic and Systems Biotechnology, 2(1):23–27, March 2017.

[133] Matthew W. Lux, Brian W. Bramlett, David A. Ball, and Jean Peccoud. Genetic design automation: Engineering fantasy or scientific renewal? Trends in Biotechnology, 30(2):120–126, February 2012.

[134] C. Ma and R.W. Simons. The IS10 antisense RNA blocks ribosome binding at the transposase translation initiation site. The EMBO Journal, 9(4):1267–1274, April 1990.

[135] James T. MacDonald, Chris Barnes, Richard I. Kitney, Paul S. Freemont, and Guy-Bart V. Stan. Computational design approaches and tools for synthetic biology. Integrative Biology, 3(2):97, 2011.

[136] Daniel Machado, Rafael S Costa, Miguel Rocha, Eugénio C Ferreira, Bruce Tidor, and Isabel Rocha. Modeling formalisms in Systems Biology. AMB Express, 1(1):45, 2011.

[137] C. Madsen, C. J. Myers, N. Roehner, C. Winstead, and Zhen Zhang. Utilizing stochastic model checking to analyze genetic circuits. In 2012 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), pages 379–386, San Diego, CA, May 2012. IEEE.

[138] Curtis Madsen, Angel Goñi Moreno, Umesh P, Zachary Palchick, Nicholas Roehner, Christian Atallah, Bryan Bartley, Kiri Choi, Robert Sidney Cox, Thomas Gorochowski, Raik Grünberg, Chris Macklin, James McLaughlin, Xianwei Meng, Tramy Nguyen, Matthew Pocock, Meher Samineni, James Scott-Brown, Ysis Tarter, Michael Zhang, Zhen Zhang, Zach Zundel, Jacob Beal, Michael Bissell, Kevin Clancy, John H. Gennari, Goksel Misirli, Chris Myers, Ernst Oberortner, Herbert Sauro, and Anil Wipat. Synthetic Biology Open Language (SBOL) Version 2.3. Journal of Integrative Bioinformatics, 16(2), June 2019.

[139] Curtis Madsen, Chris J. Myers, Tyler Patterson, Nicholas Roehner, Jason T. Stevens, and Chris Winstead. Design and Test of Genetic Circuits Using iBioSim. IEEE Design & Test of Computers, 29(3):32–39, June 2012.

[140] Curtis Madsen, Zhen Zhang, Nicholas Roehner, Chris Winstead, and Chris Myers. Stochastic Model Checking of Genetic Circuits. ACM Journal on Emerging Technologies in Computing Systems, 11(3):1–21, December 2014.

[141] Lidija Magdevska. Computational design of synchronous sequential structures in biological systems. Journal of Computational Science, page 8, 2017.

[142] Raina M. Maier and Ian L. Pepper. Bacterial Growth. In Environmental Microbiology, pages 37–56. Elsevier, 2015.

[143] A. B. Makar, K. E. McMartin, M. Palese, and T. R. Tephly. Formate assay in body fluids: Application in methanol poisoning. Biochemical Medicine, 13(2):117–126, June 1975.

[144] Jeanet Mante and Chris J. Myers. Advancing reuse of genetic parts: Progress and remaining challenges. Nature Communications, 14(1):2953, May 2023.

[145] Jeanet Mante, Nicholas Roehner, Kevin Keating, James Alastair McLaughlin, Eric Young, Jacob Beal, and Chris J. Myers. Curation Principles Derived from the Analysis of the SBOL iGEM Data Set. ACS Synthetic Biology, page acssynbio.1c00225, September 2021.

[146] Mario Andrea Marchisio. Introduction in Synthetic Biology. Learning Materials in Biosciences. Springer Singapore, Singapore, 2018.

[147] Leonardo Martins-Santana, Luisa C. Nora, Ananda Sanches-Medeiros, Gabriel L. Lovate, Murilo H. A. Cassiano, and Rafael Silva-Rocha. Systems and Synthetic Biology Approaches to Engineer Fungi for Fine Chemical Production. Frontiers in Bioengineering and Biotechnology, 6:117, October 2018.

[148] William H. Mather, Jeff Hasty, Lev S. Tsimring, and Ruth J. Williams. Translational Cross Talk in Gene Networks. Biophysical Journal, 104(11):2564–2572, June 2013.

[149] Yukiko Matsuoka, Samik Ghosh, and Hiroaki Kitano. Consistent design schematics for biological systems: Standardization of representation in biological engineering. Journal of The Royal Society Interface, 6(suppl_4), August 2009.

[150] Harley H. McAdams and Adam Arkin. Stochastic mechanisms in gene expression. Proceedings of the National Academy of Sciences, 94(3):814–819, February 1997.

[151] James Alastair McLaughlin, Jacob Beal, Göksel Mısırlı, Raik Grünberg, Bryan A. Bartley, James Scott-Brown, Prashant Vaidyanathan, Pedro Fontanarrosa, Ernst Oberortner, Anil Wipat, Thomas E. Gorochowski, and Chris J. Myers. The Synthetic Biology Open Language (SBOL) Version 3: Simplified Data Exchange for Bioengineering. Frontiers in Bioengineering and Biotechnology, 8:1009, September 2020.

[152] James Alastair McLaughlin, Chris J. Myers, Zach Zundel, Göksel Mısırlı, Michael Zhang, Irina Dana Ofiteru, Angel Goñi-Moreno, and Anil Wipat. SynBioHub: A Standards-Enabled Design Repository for Synthetic Biology. ACS Synthetic Biology, 7(2):682–688, February 2018.

[153] James Alastair McLaughlin, Chris J. Myers, Zach Zundel, Nathan Wilkinson, Christian Atallah, and Anil Wipat. Sboljs: Bringing the Synthetic Biology Open Language to the Web Browser. ACS Synthetic Biology, 8(1):191–193, January 2019.

[154] J. Kyle Medley, Kiri Choi, Matthias König, Lucian Smith, Stanley Gu, Joseph Hellerstein, Stuart C. Sealfon, and Herbert M. Sauro. Tellurium notebooks—An environment for reproducible dynamical modeling in systems biology. PLOS Computational Biology, 14(6):e1006220, June 2018.

[155] Deepak Mishra, Tristan Bepler, Brian Teague, Bonnie Berger, Jim Broach, and Ron Weiss. An engineered protein-phosphorylation toggle network with implications for endogenous network discovery. Science, 373(6550):eaav0780, July 2021.

[156] 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, 27(7):973–979, April 2011.

[157] Goksel Misirli, Jennifer Hallinan, and Anil Wipat. Composable Modular Models for Synthetic Biology. ACM Journal on Emerging Technologies in Computing Systems, 11(3):1–19, December 2014.

[158] Göksel Misirli, Tramy Nguyen, James Alastair McLaughlin, Prashant Vaidyanathan, Timothy S. Jones, Douglas Densmore, Chris Myers, and Anil Wipat. A Computational Workflow for the Automated Generation of Models of Genetic Designs. ACS Synthetic Biology, 8(7):1548–1559, July 2019.

[159] Tom Mitchell, Jacob Beal, and Bryan Bartley. pySBOL3: SBOL3 for Python Programmers. ACS Synthetic Biology, 11(7):2523–2526, July 2022.

[160] Tae Seok Moon, Chunbo Lou, Alvin Tamsir, Brynne C. Stanton, and Christopher A. Voigt. Genetic programs constructed from layered logic gates in single cells. Nature, 491(7423):249–253, November 2012.

[161] Felix Moser, Amin Espah Borujeni, Amar N Ghodasara, Ewen Cameron, and Yongjin Park. Dynamic control of endogenous metabolism with combinatorial logic circuits. Molecular Systems Biology, page 18, 2018.

[162] Brian Munsky and Mustafa Khammash. The Finite State Projection Approach for the Analysis of Stochastic Noise in Gene Networks. IEEE Transactions on Automatic Control, 53(Special Issue):201–214, January 2008.

[163] Vivek K Mutalik, Joao C Guimaraes, Guillaume Cambray, Colin Lam, Marc Juul Christoffersen, Quynh-Anh Mai, Andrew B Tran, Morgan Paull, Jay D Keasling, Adam P Arkin, and Drew Endy. Precise and reliable gene expression via standard transcription and translation initiation elements. Nature Methods, 10(4):354–360, April 2013.

[164] 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, 25(21):2848–2849, November 2009.

[165] Chris J Myers. Asynchronous Circuit Design. John Wiley & Sons, 2001.

[166] Chris J. Myers. Computational Synthetic Biology: Progress and the Road Ahead. IEEE Transactions on Multi-Scale Computing Systems, 1(1):19–32, January 2015.

[167] Chris J. Myers. Engineering Genetic Circuits. Chapman and Hall/CRC, 0 edition, April 2016.

[168] Chris J. Myers, Gary Bader, Padraig Gleeson, Martin Golebiewski, Michael Hucka, Nicolas Le Novere, David P. Nickerson, Falk Schreiber, and Dagmar Waltemath. A brief history of COMBINE. In 2017 Winter Simulation Conference (WSC), pages 884–895, Las Vegas, NV, December 2017. IEEE.

[169] Chris J. Myers, Nathan Barker, Hiroyuki Kuwahara, Kevin Jones, Curtis Madsen, and Nam-Phuong D. Nguyen. Genetic design automation. In Proceedings of the 2009 International Conference on Computer-Aided Design, pages 713–716, San Jose California, November 2009. ACM.

[170] Chris J. Myers, Jacob Beal, Thomas E. Gorochowski, Hiroyuki Kuwahara, Curtis Madsen, James Alastair McLaughlin, Göksel Mısırlı, Tramy Nguyen, Ernst Oberortner, Meher Samineni, Anil Wipat, Michael Zhang, and Zach Zundel. A standard-enabled workflow for synthetic biology. Biochemical Society Transactions, 45(3):793–803, June 2017.

[171] Thakur Neupane, Chris J. Myers, Curtis Madsen, Hao Zheng, and Zhen Zhang. STAMINA: STochastic Approximate Model-Checker for INfinite-State Analysis. Computer Aided Verification, pages 540–549, Cham, 2019. Springer International Publishing.

[172] Tramy Nguyen. Asynchronous Genetic Circuit Design. PhD thesis, University of Utah, December 2019.

[173] Tramy Nguyen, Timothy S. Jones, Pedro Fontanarrosa, Jeanet V. Mante, Zach Zundel, Douglas Densmore, and Chris Myers. Design of Asynchronous Genetic Circuits. Proceedings of the IEEE, 107(7):1356–1368, July 2019.

[174] Tramy Nguyen, Nicholas Roehner, Zach Zundel, and Chris J. Myers. A Converter from the Systems Biology Markup Language to the Synthetic Biology Open Language. ACS Synthetic Biology, 5(6):479–486, June 2016.

[175] 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, 352(6281):aac7341–aac7341, April 2016.

[176] Amir Pandi, Mathilde Koch, Peter L. Voyvodic, Paul Soudier, Jerome Bonnet, Manish Kushwaha, and Jean-Loup Faulon. Metabolic perceptrons for neural computing in biological systems. Nature Communications, 10(1):3880, December 2019.

[177] Johan Paulsson. Summing up the noise in gene networks. Nature, 427(6973):415–418, January 2004.

[178] William R. Pearson. Using the FASTA Program to Search Protein and DNA Sequence Databases, volume 24, pages 307–332. Humana Press, New Jersey, February 1994.

[179] Jean Peccoud. Synthetic Biology : Fostering the cyber-biological revolution. Synthetic Biology, 1(1):ysw001, 2016.

[180] Jean-Denis Pédelacq, Stéphanie Cabantous, Timothy Tran, Thomas C Terwilliger, and Geoffrey S Waldo. Engineering and characterization of a superfolder green fluorescent protein. Nature Biotechnology, 24(1):79–88, January 2006.

[181] Giansimone Perrino, Andreas Hadjimitsis, Rodrigo Ledesma-Amaro, and Guy-Bart Stan. Control engineering and synthetic biology: Working in synergy for the analysis and control of microbial systems. Current Opinion in Microbiology, 62:68–75, August 2021.

[182] Ira E. Phillips and Pamela A. Silver. A New Biobrick Assembly Strategy Designed for Facile Protein Engineering. April 2006.

[183] A. Polynikis, S.J. Hogan, and M. Di Bernardo. Comparing different ODE modelling approaches for gene regulatory networks. Journal of Theoretical Biology, 261(4):511–530, December 2009.

[184] Priscilla E. M. Purnick and Ron Weiss. The second wave of synthetic biology: From modules to systems. Nature Reviews Molecular Cell Biology, 10(6):410–422, June 2009.

[185] Jacqueline Y. Quinn, Robert Sidney Cox, Aaron Adler, Jacob Beal, Swapnil Bhatia, Yizhi Cai, Joanna Chen, Kevin Clancy, Michal Galdzicki, Nathan J. Hillson, Nicolas Le Novère, Akshay J. Maheshwari, James Alastair McLaughlin, Chris J. Myers, Umesh P, Matthew Pocock, Cesar Rodriguez, Larisa Soldatova, Guy-Bart V. Stan, Neil Swainston, Anil Wipat, and Herbert M. Sauro. SBOL Visual: A Graphical Language for Genetic Designs. PLOS Biology, 13(12):e1002310, December 2015.

[186] J. M. Raser. Noise in Gene Expression: Origins, Consequences, and Control. Science, 309(5743):2010–2013, September 2005.

[187] K.V.B.V. Rayudu, P Srihari Rao, and K S R Krishna Prasad. Testing and diagnosis faults in FinFet circuits based on advanced test algorithm. In 2018 International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering (ICRIEECE), pages 2782–2792, Bhubaneswar, India, July 2018. IEEE.

[188] Riley Roberts, Thakur Neupane, Lukas Buecherl, Chris J. Myers, and Zhen Zhang. STAMINA 2.0: Improving Scalability of Infinite-State Stochastic Model Checking. In Bernd Finkbeiner and Thomas Wies, editors, Verification, Model Checking, and Abstract Interpretation, volume 13182, pages 319–331. Springer International Publishing, Cham, 2022.

[189] Nicholas Roehner, Jacob Beal, Kevin Clancy, Bryan Bartley, Goksel Misirli, Raik Grünberg, Ernst Oberortner, Matthew Pocock, Michael Bissell, Curtis Madsen, Tramy Nguyen, Michael Zhang, Zhen Zhang, Zach Zundel, Douglas Densmore, John H. Gennari, Anil Wipat, Herbert M. Sauro, and Chris J. Myers. Sharing Structure and Function in Biological Design with SBOL 2.0. ACS Synthetic Biology, 5(6):498–506, June 2016.

[190] Nicholas Roehner and Chris J. Myers. Directed Acyclic Graph-Based Technology Mapping of Genetic Circuit Models. ACS Synthetic Biology, 3(8):543–555, August 2014.

[191] Nicholas Roehner, Zhen Zhang, Tramy Nguyen, and Chris J. Myers. Generating Systems Biology Markup Language Models from the Synthetic Biology Open Language. ACS Synthetic Biology, 4(8):873–879, August 2015.

[192] Sascha Rollié, Michael Mangold, and Kai Sundmacher. Designing biological systems: Systems Engineering meets Synthetic Biology. Chemical Engineering Science, 69(1):1–29, February 2012.

[193] Alvaro Sanchez, Sandeep Choubey, and Jane Kondev. Stochastic models of transcription: From single molecules to single cells. Methods, 62(1):13–25, July 2013.

[194] Javier Santos-Moreno, Eve Tasiudi, Joerg Stelling, and Yolanda Schaerli. Multistable and dynamic CRISPRi-based synthetic circuits. Nature Communications, 11(1):2746, December 2020.

[195] Dipali G. Sashital, Blake Wiedenheft, and Jennifer A. Doudna. Mechanism of Foreign DNA Selection in a Bacterial Adaptive Immune System. Molecular Cell, 46(5):606–615, June 2012.

[196] Eric W Sayers, Jeffrey Beck, Evan E Bolton, Devon Bourexis, James R Brister, Kathi Canese, Donald C Comeau, Kathryn Funk, Sunghwan Kim, William Klimke, Aron Marchler-Bauer, Melissa Landrum, Stacy Lathrop, Zhiyong Lu, Thomas L Madden, Nuala O'Leary, Lon Phan, Sanjida H Rangwala, Valerie A Schneider, Yuri Skripchenko, Jiyao Wang, Jian Ye, Barton W Trawick, Kim D Pruitt, and Stephen T Sherry. Database resources of the National Center for Biotechnology Information. Nucleic Acids Research, 49(D1):D10–D17, January 2021.

[197] Katherine A Schaumberg, Mauricio S Antunes, Tessema K Kassaw, Wenlong Xu, Christopher S Zalewski, June I Medford, and Ashok Prasad. Quantitative characterization of genetic parts and circuits for plant synthetic biology. Nature Methods, 13(1):94–100, January 2016.

[198] Tobias Schladt, Nicolai Engelmann, Erik Kubaczka, Christian Hochberger, and Heinz Koeppl. Automated Design of Robust Genetic Circuits: Structural Variants and Parameter Uncertainty. ACS Synthetic Biology, 10(12):3316–3329, December 2021.

[199] Thomas Schlitt and Alvis Brazma. Current approaches to gene regulatory network modelling. BMC Bioinformatics, 8(S6):S9, September 2007.

[200] Zachary Sents, Thomas E. Stoughton, Lukas Buecherl, Payton J. Thomas, Pedro Fontanarrosa, and Chris J. Myers. SynBioSuite: A Tool for Improving the Workflow for Genetic Design and Modeling. ACS Synthetic Biology, 12(3):892–897, March 2023.

[201] Reshma P Shetty, Drew Endy, and Thomas F Knight. Engineering BioBrick vectors from BioBrick parts. Journal of Biological Engineering, 2(1):5, December 2008.

[202] Ting Shi, Pingping Han, Chun You, and Yi-Heng P. Job Zhang. An in vitro synthetic biology platform for emerging industrial biomanufacturing: Bottom-up pathway design. Synthetic and Systems Biotechnology, 3(3):186–195, September 2018.

[203] Jonghyeon Shin, Shuyi Zhang, Bryan S Der, Alec AK Nielsen, and Christopher A Voigt. Programming Escherichia coli to function as a digital display. Molecular Systems Biology, 16(3), March 2020.

[204] Rafael Silva-Rocha, Esteban Martínez-García, Belén Calles, Max Chavarría, Alejandro Arce-Rodríguez, Aitor de las Heras, A. David Páez-Espino, Gonzalo Durante-Rodríguez, Juhyun Kim, Pablo I. Nikel, Raúl Platero, and Víctor de Lorenzo. The Standard European Vector Architecture (SEVA): A coherent platform for the analysis and deployment of complex prokaryotic phenotypes. Nucleic Acids Research, 41(D1):D666–D675, January 2013.

[205] Piro Siuti, John Yazbek, and Timothy K Lu. Synthetic circuits integrating logic and memory in living cells. Nature Biotechnology, 31(5):448–452, May 2013.

[206] Sean C. Sleight and Herbert M. Sauro. Visualization of Evolutionary Stability Dynamics and Competitive Fitness of Escherichia coli Engineered with Randomized Multigene Circuits. ACS Synthetic Biology, 2(9):519–528, September 2013.

[207] Adrian L. Slusarczyk, Allen Lin, and Ron Weiss. Foundations for the design and implementation of synthetic genetic circuits. Nature Reviews Genetics, 13(6):406–420, June 2012.

[208] Lucian Smith, F Bergmann, A Garny, T Helikar, J Karr, D Nickerson, H Sauro, and Matthias König. Simulation experiment description markup language (SED-ML): Level 1 version 4. J Integr Bioinform, 18:2021–0021, 2021.

[209] Christina D Smolke. Building outside of the box: iGEM and the BioBricks Foundation. nature biotechnology, 27(12):4, 2009.

[210] Rotem Sorek, C. Martin Lawrence, and Blake Wiedenheft. CRISPR-Mediated Adaptive Immune Systems in Bacteria and Archaea. Annual Review of Biochemistry, 82(1):237–266, June 2013.

[211] Brynne C Stanton, Alec A K Nielsen, Alvin Tamsir, Kevin Clancy, Todd Peterson, and Christopher A Voigt. Genomic mining of prokaryotic repressors for orthogonal logic gates. Nature Chemical Biology, 10(2):99–105, February 2014.

[212] Peter S. Swain, Michael B. Elowitz, and Eric D. Siggia. Intrinsic and extrinsic contributions to stochasticity in gene expression. Proceedings of the National Academy of Sciences, 99(20):12795–12800, October 2002.

[213] Mao Taketani, Jianbo Zhang, Shuyi Zhang, Alexander J. Triassi, Yu-Ja Huang, Linda G. Griffith, and Christopher A. Voigt. Genetic circuit design automation for the gut resident species Bacteroides thetaiotaomicron. Nature Biotechnology, 38(8):962–969, August 2020.

[214] Xiao Tan, Justin H. Letendre, James J. Collins, and Wilson W. Wong. Synthetic biology in the clinic: Engineering vaccines, diagnostics, and therapeutics. Cell, 184(4):881–898, February 2021.

[215] Huseyin Tas, Lewis Grozinger, Angel Goñi-Moreno, and Victor de Lorenzo. Automated design and implementation of a NOR gate in Pseudomonas putida. Synthetic Biology, 6(1):ysab024, October 2021.

[216] Logan Terry, Jared Earl, Sam Thayer, Samuel Bridge, and Chris J. Myers. SBOLCanvas: A Visual Editor for Genetic Designs. ACS Synthetic Biology, 10(7):1792–1796, July 2021.

[217] Mukund Thattai and Alexander van Oudenaarden. Intrinsic noise in gene regulatory networks. Proceedings of the National Academy of Sciences, 98(15):8614–8619, July 2001.

[218] Soichiro Tsuda. Synthetic Biology. In Bharat Bhushan, editor, Encyclopedia of Nanotechnology, pages 1–5. Springer Netherlands, Dordrecht, 2015.

[219] Prashant Vaidyanathan, Bryan S. Der, Swapnil Bhatia, Nicholas Roehner, Ryan Silva, Christopher A. Voigt, and Douglas Densmore. A Framework for Genetic Logic Synthesis. Proceedings of the IEEE, 103(11):2196–2207, November 2015.

[220] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In 26th Annual Symposium on Foundations of Computer Science (Sfcs 1985), pages 327–338, Portland, OR, USA, 1985. IEEE.

[221] Cristina Vilanova and Manuel Porcar. iGEM 2.0—refoundations for engineering biology. Nature Biotechnology, 32(5):420–424, May 2014.

[222] Christopher A Voigt. Genetic parts to program bacteria. Current Opinion in Biotechnology, 17(5):548–557, October 2006.

[223] Eberhard O Voit. A first course in systems biology. Garland Science, 2017.

[224] Dagmar Waltemath, Richard Adams, Frank T Bergmann, Michael Hucka, Fedor Kolpakov, Andrew K Miller, Ion I Moraru, David Nickerson, Sven Sahle, Jacky L Snoep, and Nicolas Le Novère. Reproducible computational biology experiments with SED-ML - The Simulation Experiment Description Markup Language. BMC Systems Biology, 5(1):198, December 2011.

[225] Beibei Wang, Huayi Yang, Jianan Sun, Chuhao Dou, Jian Huang, and Feng-Biao Guo. BioMaster: An Integrated Database and Analytic Platform to Provide Comprehensive Information About BioBrick Parts. Frontiers in Microbiology, 12:593979, January 2021.

[226] Leandro Watanabe, Tramy Nguyen, Michael Zhang, Zach Zundel, Zhen Zhang, Curtis Madsen, Nicholas Roehner, and Chris Myers. IBIOSIM 3: A Tool for Model-Based Genetic Circuit Design. ACS Synthetic Biology, 8(7):1560–1563, July 2019.

[227] Peng-Fei Xia, Hua Ling, Jee Loon Foo, and Matthew Wook Chang. Synthetic genetic circuits for programmable biological functionalities. Biotechnology Advances, 37(6):107393, November 2019.

[228] Wang Xiang, Wang Xiao-Cui, Jiang Jing-Bo, Sun Jin-Jing, Pang Shu-Song, He Han-Hong, and Wang Wei-Ke. Genetic Circuit for the Early Warning of Lung Cancer using iBioSim. ITM Web of Conferences, page 5, 2016.

[229] Yiyu Xiang, Neil Dalchau, and Baojun Wang. Scaling up genetic circuit design for cellular computing: Advances and prospects. Natural Computing, 17(4):833–853, December 2018.

[230] Fusun Yaman, Swapnil Bhatia, Aaron Adler, Douglas Densmore, and Jacob Beal. Automated Selection of Synthetic Biology Parts for Genetic Regulatory Networks. ACS Synthetic Biology, 1(8):332–344, August 2012.

[231] Andrew I. Yao, Timothy A. Fenton, Keegan Owsley, Phillip Seitzer, David J. Larsen, Holly Sit, Jennifer Lau, Arjun Nair, Justin Tantiongloc, Ilias Tagkopoulos, and Marc T. Facciotti. Promoter Element Arising from the Fusion of Standard BioBrick Parts. ACS Synthetic Biology, 2(2):111–120, February 2013.

[232] George T. Yates and Thomas Smotzer. On the lag phase and initial decline of microbial growth curves. Journal of Theoretical Biology, 244(3):511–517, February 2007.

[233] Fangfei Yin, Fei Wang, Chunhai Fan, Xiaolei Zuo, and Qian Li. Biosensors based on DNA logic gates. View, 2(2):20200038, April 2021.

[234] Boyan Yordanov, Neil Dalchau, Paul K. Grant, Michael Pedersen, Stephen Emmott, Jim Haseloff, and Andrew Phillips. A Computational Method for Automated Characterization of Genetic Components. ACS Synthetic Biology, 3(8):578–588, August 2014.

[235] Michael Zhang, James Alastair McLaughlin, Anil Wipat, and Chris J. Myers. SBOLDesigner 2: An Intuitive Tool for Structural Genetic Design. ACS Synthetic Biology, 6(7):1150–1160, July 2017.

[236] Zhen Zhang, Tramy Nguyen, Nicholas Roehner, Goksel Misirli, Matthew Pocock, Ernst Oberortner, Meher Samineni, Zach Zundel, Jacob Beal, Kevin Clancy, Anil Wipat, and Chris J. Myers. libSBOLj 2.0: A Java Library to Support SBOL 2.0. IEEE Life Sciences Letters, 1(4):34–37, December 2015.

[237] T. I. Zohdi and P. Wriggers. Computational micro-macro material testing. Archives of Computational Methods in Engineering, 8(2):131–228, June 2001.

A

Failure Predictions of Circuit 0x8E

This appendix presents predictions regarding the percentage failure rates across various models, environmental conditions, and investigated variables as analyzed in Chapter 5. Tables A.1, A.2, and A.3 provide insights into the likelihood of circuit failures during different input transitions for the circuit implementations depicted in Figure 4.2. Specifically, Table A.1 presents the percentages of circuit failures in the original design circuit transitions, while Table A.2 offers analogous information for the two-inverter design. Additionally, Table A.3 offers corresponding data for the logic-hazard-free design. Furthermore, this section encompasses quantitative model predictions for the two-inverter (Figure A.1) and logic-hazard-free implementation (Figure A.2). The final table, Table A.4, showcases the absolute failure percentages of each model and circuit in achieving their

expected steady state. Subsequently, Figures A.3, A.4, and A.5 depict the quantitative predictions for each model regarding their attainment of steady state.

Table A.1: Results of the input transition failure analysis of the original design of circuit 0x8E

		Original Design				
	Input	E/C/C	E/D/C	E/D/D	I/D/D	I/D/A
0-Function Hazards	$(0,1,0) \rightarrow (1,1,1)$	11.3	17.2	15.5	23.2	36.9
	$(0,1,0) \rightarrow (1,0,0)$	20.4	10.7	38.5	73.9	74.1
	$(1,1,1) \rightarrow (1,0,0)$	20.5	10.8	42.7	98.5	93.4
	$(1,1,1) \rightarrow (0,1,0)$	19.6	16.2	34.2	97.0	92.3
	$(1,0,0) \rightarrow (0,1,0)$	19.6	16.2	23.7	75.5	79.7
	$(1,0,0) \rightarrow (1,1,1)$	11.3	6.5	11.1	21.9	36.1
1-Function Hazards	$(0,1,1) \rightarrow (1,0,1)$	12.9	45.4	87.9	99.9	99.0
	$(0,0,0) \rightarrow (0,1,1)$	29.6	41.2	28.0	47.9	87.0
	$(0,0,0) \rightarrow (1,0,1)$	13.0	45.8	84.6	100.0	98.0
	$(1,0,1) \rightarrow (0,1,1)$	29.6	41.4	90.1	99.9	99.0
	$(0,1,1) \rightarrow (0,0,0)$	27.4	45.7	46.6	72.0	89.6
	$(1,0,1) \rightarrow (0,0,0)$	26.8	45.2	56.0	73.7	88.9
0-Input Transitions	$(0,1,0) \rightarrow (1,1,0)$	10.0	5.1	13.3	14.8	28.2
	$(1,0,0) \rightarrow (1,1,0)$	10.1	5.1	10.6	16.7	28.8
	$(1,1,0) \rightarrow (1,1,1)$	11.3	6.5	6.3	18.8	30.9
	$(1,1,0) \rightarrow (0,1,0)$	19.5	16.1	10.8	20.4	36.0
	$(1,1,0) \rightarrow (1,0,0)$	20.4	10.7	32.0	72.1	72.7
	$(1,1,1) \rightarrow (1,1,0)$	10.1	5.1	6.5	18.6	29.2
1-Input Transitions	$(0,0,1) \rightarrow (1,0,1)$	12.9	45.2	21.0	30.5	74.7
	$(0,1,1) \rightarrow (0,0,1)$	3.1	10.7	19.0	22.5	56.5
	$(0,0,0) \rightarrow (0,0,1)$	3.1	10.8	25.4	25.1	83.3
	$(1,0,1) \rightarrow (0,0,1)$	3.1	10.7	22.1	21.5	58.6
	$(0,0,1) \rightarrow (0,1,1)$	29.4	41.2	18.2	31.0	31.0
	$(0,0,1) \rightarrow (0,0,0)$	26.8	44.9	35.6	57.2	70.3

Table A.2: Results of the input transition analysis of the two-inverter design of circuit 0x8E

		Two Inverter Design				
	Input	E/C/C	E/D/C	E/D/D	I/D/D	I/D/A
0-Function Hazards	$(0,1,0) \rightarrow (1,1,1)$	12.7	12.2	37.8	76.7	75.5
	$(0,1,0) \rightarrow (1,0,0)$	24.8	16.8	40.1	77.1	80.4
	$(1,1,1) \rightarrow (1,0,0)$	24.8	16.8	43.6	98.6	94.4
	$(1,1,1) \rightarrow (0,1,0)$	14.0	11.9	22.2	43.0	62.3
	$(1,0,0) \rightarrow (0,1,0)$	14.2	11.8	19.8	29.4	47.9
	$(1,0,0) \rightarrow (1,1,1)$	12.8	12.2	15.6	24.7	38.2
1-Function Hazards	$(0,1,1) \rightarrow (1,0,1)$	11.0	37.9	31.7	65.7	84.4
	$(0,0,0) \rightarrow (0,1,1)$	45.9	50.0	36.6	48.9	86.8
	$(0,0,0) \rightarrow (1,0,1)$	11.1	38.0	48.5	67.7	85.1
	$(1,0,1) \rightarrow (0,1,1)$	46.7	51.9	90.2	100.0	100.0
	$(0,1,1) \rightarrow (0,0,0)$	43.0	54.6	53.2	76.0	89.3
	$(1,0,1) \rightarrow (0,0,0)$	43.5	55.8	68.0	95.2	96.2
0-Input Transitions	$(0,1,0) \rightarrow (1,1,0)$	9.0	5.2	13.3	18.7	33.0
	$(1,0,0) \rightarrow (1,1,0)$	9.2	5.1	15.4	18.8	31.7
	$(1,1,0) \rightarrow (1,1,1)$	12.7	12.2	11.5	20.9	34.9
	$(1,1,0) \rightarrow (0,1,0)$	13.8	11.8	10.6	17.3	32.6
	$(1,1,0) \rightarrow (1,0,0)$	24.8	16.8	33.4	72.0	75.0
	$(1,1,1) \rightarrow (1,1,0)$	9.0	5.1	13.2	20.6	33.8
1-Input Transitions	$(0,0,1) \rightarrow (1,0,1)$	11.0	37.8	20.3	18.0	56.5
	$(0,1,1) \rightarrow (0,0,1)$	3.2	12.3	29.4	27.4	57.1
	$(0,0,0) \rightarrow (0,0,1)$	3.2	12.4	33.7	27.4	61.0
	$(1,0,1) \rightarrow (0,0,1)$	3.4	12.8	22.2	36.4	73.5
	$(0,0,1) \rightarrow (0,1,1)$	45.8	50.0	27.7	35.1	78.8
	$(0,0,1) \rightarrow (0,0,0)$	42.7	53.9	40.9	55.5	74.0

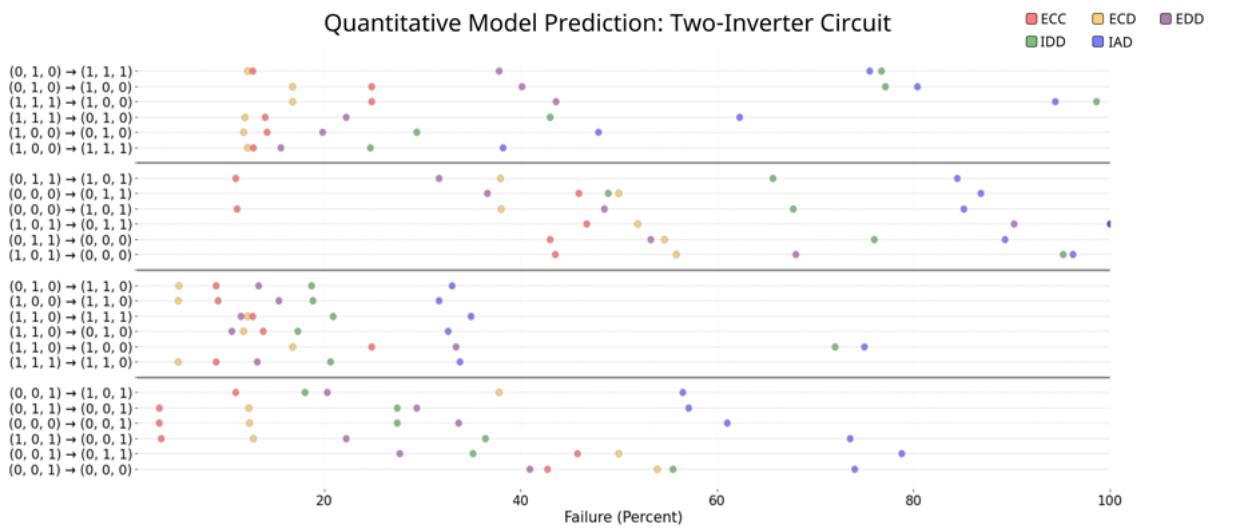


Figure A.1: Quantitative model prediction. The figure illustrates the predicted failure probability for each transition and model concerning the two-inverter layout. The transitions on the y-axis are grouped into four sections. The first two sections correspond to *static* $0 \to 0$ and *static* $1 \to 1$ *function hazards*, while the next two groups correspond to *static* $0 \to 0$ and *static* $1 \to 1$ *transitions* without function hazards. On the x-axis, the failure probability is represented in percentages, with each model indicated by a distinct marker.

Table A.3: Results of the input transition failure analysis of the logic-hazard-free design of circuit 0x8E

		Logic Hazard Free Design				
	Input	E/C/C	E/D/C	E/D/D	I/D/D	I/D/A
0-Function Hazards	$(0,1,0) \rightarrow (1,1,1)$	11.5	5.8	12.8	18.6	35.4
	$(0,1,0) \rightarrow (1,0,0)$	21.7	7.2	12.4	17.2	30.8
	$(1,1,1) \rightarrow (1,0,0)$	21.7	7.2	30.4	95.7	90.2
	$(1,1,1) \rightarrow (0,1,0)$	10.0	15.2	32.7	97.1	95.1
	$(1,0,0) \rightarrow (0,1,0)$	10.0	15.4	20.0	77.5	74.6
	$(1,0,0) \rightarrow (1,1,1)$	11.5	5.8	10.9	20.7	37.5
1-Function Hazards	$(0,1,1) \rightarrow (1,0,1)$	11.8	41.0	89.4	99.8	98.5
	$(0,0,0) \rightarrow (0,1,1)$	30.2	42.9	29.7	49.4	85.8
	$(0,0,0) \rightarrow (1,0,1)$	11.9	41.0	86.4	99.8	98.5
	$(1,0,1) \rightarrow (0,1,1)$	30.1	43.4	89.7	99.9	98.8
	$(0,1,1) \rightarrow (0,0,0)$	48.6	45.3	48.8	46.0	78.5
	$(1,0,1) \rightarrow (0,0,0)$	47.5	44.9	47.7	39.6	76.1
0-Input Transitions	$(0,1,0) \rightarrow (1,1,0)$	5.5	3.5	7.7	6.7	13.1
	$(1,0,0) \rightarrow (1,1,0)$	5.6	3.6	8.5	9.6	17.1
	$(1,1,0) \rightarrow (1,1,1)$	11.5	5.8	6.2	17.9	26.1
	$(1,1,0) \rightarrow (0,1,0)$	10.0	15.0	5.5	14.5	20.0
	$(1,1,0) \rightarrow (1,0,0)$	21.7	7.2	6.4	17.4	22.9
	$(1,1,1) \rightarrow (1,1,0)$	5.6	3.6	7.0	17.3	26.0
1-Input Transitions	$(0,0,1) \rightarrow (1,0,1)$	11.8	40.5	20.3	33.2	75.2
	$(0,1,1) \rightarrow (0,0,1)$	3.2	10.0	25.9	21.9	57.2
	$(0,0,0) \rightarrow (0,0,1)$	3.2	9.9	25.9	25.8	55.0
	$(1,0,1) \rightarrow (0,0,1)$	3.2	9.8	20.9	22.6	52.4
	$(0,0,1) \rightarrow (0,1,1)$	30.1	42.6	20.9	31.8	76.0
	$(0,0,1) \rightarrow (0,0,0)$	47.5	44.0	21.9	14.2	46.7

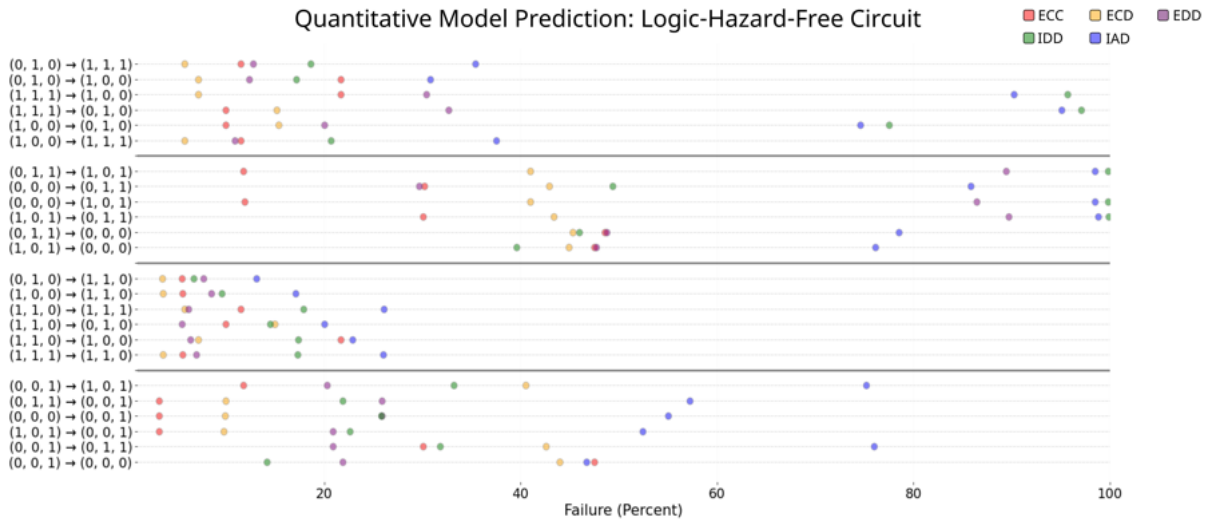


Figure A.2: Quantitative model prediction. The figure illustrates the predicted failure probability for each transition and model concerning the logic-hazard-free layout. The transitions on the y-axis are grouped into four sections. The first two sections correspond to *static* $0 \to 0$ and *static* $1 \to 1$ *function hazards*, while the next two groups correspond to *static* $0 \to 0$ and *static* $1 \to 1$ *transitions* without function hazards. On the x-axis, the failure probability is represented in percentages, with each model indicated by a distinct marker.

Table A.4: Results of the steady-state failure percentage failures

Circuit Failures		E/C/C			E/D/C			E/D/D			I/D/D			I/D/A		
		O	T	N	O	T	N	O	T	N	O	T	N	O	T	N
Wrong Steady-state	$(0,0,0)$	26.4	42.7	47.5	43.9	52.6	44.0	24.9	32.9	25.9	4.5	6.0	5.2	6.6	17.3	6.1
	$(0,0,1)$	3.1	3.2	3.1	10.6	12.2	9.3	7.2	7.2	6.4	0.0	0.0	0.0	0.3	0.1	0.1
	$(0,1,0)$	19.5	13.8	10.0	16.1	11.8	15.0	13.3	13.3	7.7	0.7	0.7	0.2	3.2	5.6	4.8
	$(0,1,1)$	29.4	45.8	30.1	41.2	50.0	42.6	18.1	28.6	21.1	2.9	4.8	3.3	11.6	12.6	9.9
	$(1,0,0)$	20.2	24.6	21.7	10.7	16.8	7.2	8.3	13.8	8.5	1.0	0.6	0.4	2.8	4.2	2.3
	$(1,0,1)$	12.9	11.0	11.8	45.0	37.8	40.5	20.6	21.1	19.1	2.2	4.2	3.7	10.1	8.8	8.3
	$(1,1,0)$	10.0	9.0	5.5	5.1	5.1	3.5	3.4	6.2	0.7	0.4	0.5	0.0	2.7	2.3	0.5
	$(1,1,1)$	11.3	12.7	11.5	6.5	12.2	5.8	5.8	11.6	6.5	0.3	0.6	0.7	4.5	8.9	4.9

WSS in %

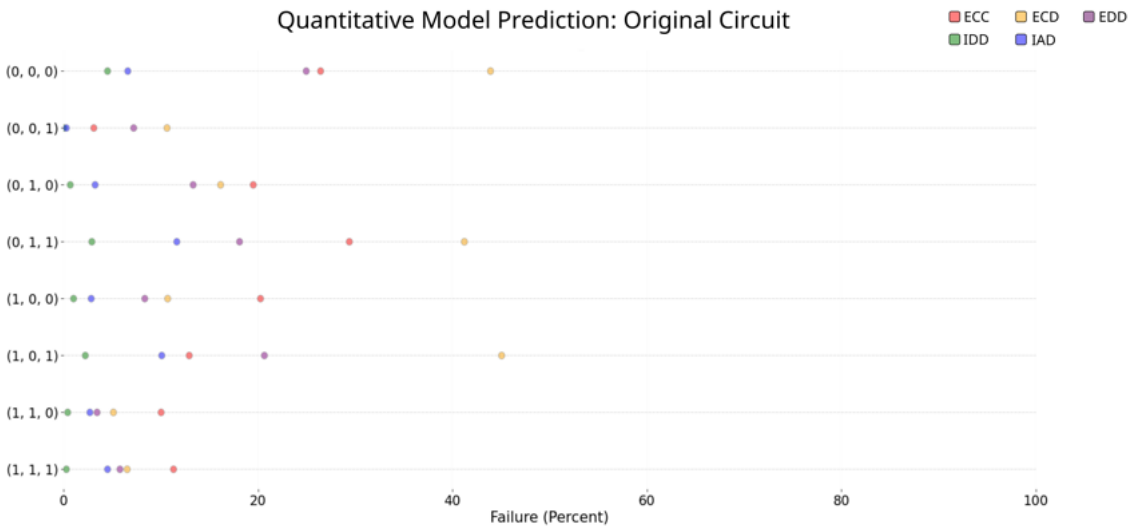


Figure A.3: Quantitative model prediction. The figure illustrates the predicted failure probability for each state and model concerning the original layout. The states are shown on the y-axis. On the x-axis, the failure probability is represented in percentages, with each model indicated by a distinct marker.

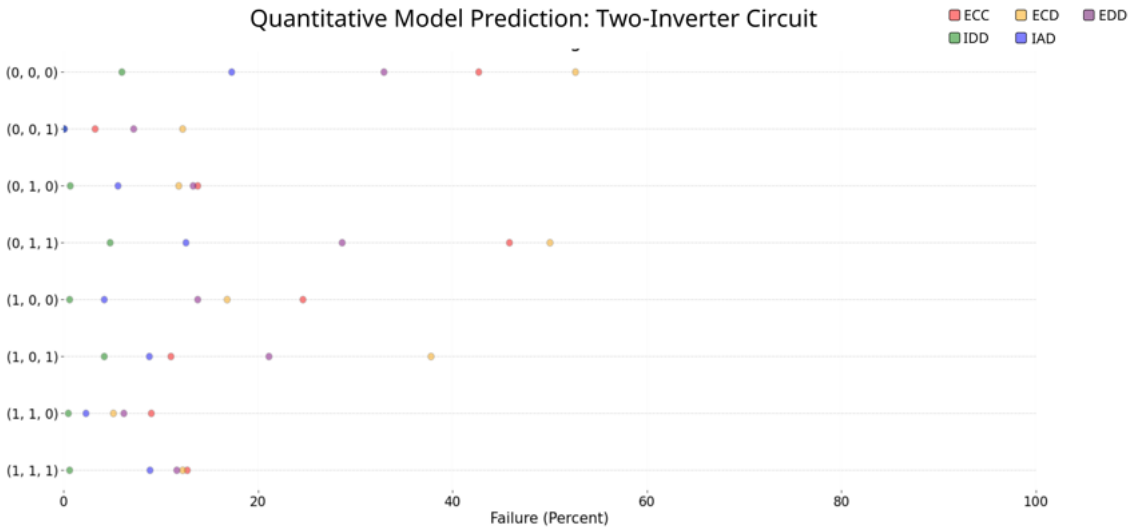


Figure A.4: Quantitative model prediction. The figure illustrates the predicted failure probability for each state and model concerning the two-inverter layout. The states are shown on the y-axis. On the x-axis, the failure probability is represented in percentages, with each model indicated by a distinct marker.

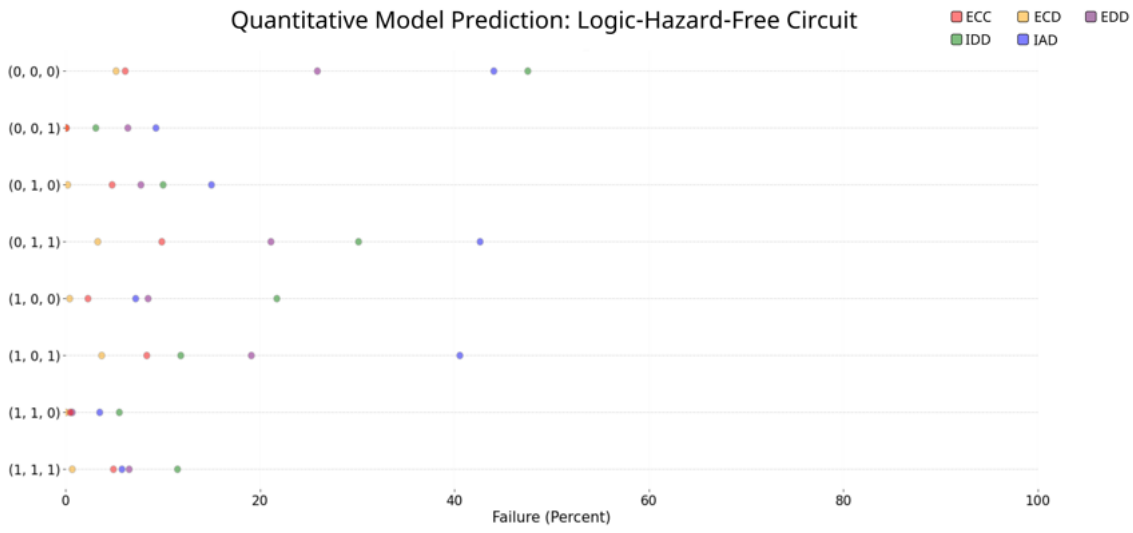


Figure A.5: Quantitative model prediction. The figure illustrates the predicted failure probability for each state and model concerning the logic-hazard-free layout. The states are shown on the y-axis. On the x-axis, the failure probability is represented in percentages, with each model indicated by a distinct marker.

B

Laboratory Protocols

This appendix offers the laboratory protocols utilized in this dissertation. The protocols assume the availability of two bacterial cultures, each harboring one of the two plasmids. One culture contains the circuit 0xF6, while the other contains the output plasmid. The two plasmids are pAN3938 [1] (circuit plasmid) and pAN4036 [2] (output plasmid). These bacterial strains can be obtained from Addgene [111].

[1] https://www.addgene.org/74697/
[2] https://www.addgene.org/74698/

Lysogeny Broth

Lysogeny broth [27] (LB), a nutrient-rich medium, serves as a cornerstone for bacterial growth. Since the 1950s, LB media formulations have stood as an industry standard for cultivating E. coli. Widely employed in molecular microbiology, these media facilitate the preparation of plasmid DNA and recombinant proteins. LB media remains a prevalent choice for maintaining and nurturing recombinant strains of E. coli.

Equipment:

- Duran bottle (250 mL)

Reagents:

- Autoclaved distilled water
- LB broth powder (Invitrogen Cat No. 12780-052)

Procedure:

(1) Add 200 mL of autoclaved distilled water to a 250 mL Duran bottle

(2) Consult the manufacturer's instructions for LB agar concentration and calculate the required mass of powder accordingly

(3) Dissolve the calculated mass of LB broth powder in 200 mL of autoclaved distilled water

(4) Shake the bottle until the powder is completely dissolved with no clumps remaining

(5) Loosen the lid of the Duran bottle until there is no resistance

(6) Autoclave for 30 minutes at 121°C

(7) Store at room temperature

M9 Media

M9 media is a commonly used synthetic growth medium in microbiology. It contains a minimal set of nutrients necessary for bacterial growth, typically including salts, a carbon source, and sometimes additional amino acids or vitamins depending on the specific experimental requirements.

Equipment:

- Duran bottle (1 L)
- Duran bottle (500 mL)
- Duran bottle (250 mL)
- Falcon tube (50 mL)
- Filter (0.22 µm)
- Syringe

Reagents:

- 1X M9 salts (Sigma-Aldrich Cat No. M6030-1KG)
- 1 mM thiamine hydrochloride (Sigma-Aldrich Cat No. T4625-10G)
- 0.4% D-(+)-glucose (Sigma-Aldrich Cat No. G8270-100G)
- 0.2% casamino acids (Sigma-Aldrich Cat No. 2240-500GM)
- 2 mM $MgSO_4$ (Sigma-Aldrich Cat No. 230391-500G)
- 0.1 mM $CaCl_2$ (Sigma-Aldrich Cat No. C3306-250G)
- Autoclaved distilled water

Procedure:

(1) Dissolve 56.4 g of M9 salts in 1 L of distilled water in Duran bottle

(2) Dissolve 100 mg of thiamine hydrochloride in 10 mL of distilled water in Falcon tube

(3) Dissolve 50 g of casamino acids in 500 mL of distilled water in Duran bottle

(4) Dissolve 24.65 g of $MgSO_4$ in 100 mL of distilled water

(5) Dissolve 14.7 g of $CaCl_2$ in 100 mL of distilled water

(6) Dissolve 20 g of glucose in 100 mL of distilled water

(7) Autoclave M9 salts, casamino acids, $MgSO_4$, and $CaCl_2$ for 30 minutes at 121°C

(8) Filter sterilize thiamine hydrochloride using a 0.22 µm filter

(9) Filter sterilize glucose using a 0.22 µm filter

(10) Add 50 mL of 5x concentrated M9 to a 250 mL Duran bottle

(11) Add 8.5 mL of filtered 1 mM thiamine hydrochloride

(12) Add 5 mL of 0.2% casamino acids

(13) Add 0.5 mL of 2 mM $MgSO_4$

(14) Add 25 µL of 0.1 mM $CaCl_2$

(15) Add 2.5 mL of 20% glucose

(16) Add 183.475 mL of distilled water

(17) Store at 4°C

Antibiotic Stocks(1000x)

Antibiotics serve a crucial role in selection processes. In synthetic biology, plasmids bestow selective antibiotic resistance upon their target bacteria when successfully transformed. The use of liquid or solid growth media supplemented with antibiotics offers a means of selecting bacteria

that have integrated the plasmid. It is noteworthy that various antibiotics are available in different concentrations. Below is the information for the antibiotics employed in this study.

Equipment:

- Falcon tube (50 mL)
- Eppendorf tube (1.5 mL)

Reagents:

- Autoclaved distilled Water
- Antibiotic powder

Procedure:

(1) Label the Falcon tube with initials, date, and content

(2) According to Table B.1, add the final volume of distilled water to a 50 mL Falcon tube

(3) Suspend the specified mass of antibiotic as indicated in Table B.1

(4) Create 1mL aliquots of the stock solution in the Eppendorf tubes

(5) Store the stock solutions in a -20°C freezer

Antibiotic	Stock[mg/mL]	Final Volume[mL]	Mass[g]
Kanamycin	50	50	2.5
Spectinomycin	60	50	3
Streptomycin	100	50	5

Table B.1: Antibiotic concentrations for stock solutions (1000x)

LB Agar Plates

LB agar plates are essential for cultivating and selecting cell cultures. They come in both selective variants, containing antibiotics, and non-selective variants, without antibiotics.

Equipment:

- Duran bottle (250 mL)
- Petri dishes
- Bunsen burner

Reagents:

- LB broth with agar powder (Invitrogen Cat No. 22700-025)
- Antibiotic stock solution (1000x)

Procedure:

(1) Pour 200 mL of distilled water into a 250 mL Duran bottle

(2) Refer to the manufacturer's instructions for LB agar concentration and calculate the required mass of powder

(3) Mix the calculated mass of LB broth powder with agar into the 200 mL of distilled water

(4) Vigorously shake the bottle until the powder completely dissolves, ensuring no clumps remain

(5) Loosen the lid of the Duran bottle until there is no resistance

(6) Autoclave for 15 minutes at 121°C

(7) Label the bottom of the petri dish with the following information: date, initials, antibiotic, plasmid, and stain

(8) Allow the LB agar to cool to approximately 60°C

(9) Light a Bunsen burner and work close to it

(10) Add 20 µL of antibiotic (1000x concentration) to 20 mL of LB agar

(11) Carefully pour approximately 20 mL of LB agar with antibiotic into each petri dish

(12) Gently swirl each petri dish on the table to ensure the agar coats the sides evenly

(13) Cover half of each petri dish with its lid and leave them near the Bunsen burner to allow condensation to evaporate for 30 to 60 minutes

(14) Stack the petri dishes, tape around them, and store them at 4°C

Streaking Out Bacteria

Streaking out bacteria is the process of transferring bacteria from a glycerol stock solution onto a petri dish for growing cell cultures.

Equipment:

- Petri dish with LB agar and antibiotic
- Bunsen burner
- Inoculating loops
- Bacteria glycerol stock (stored at -80ºC)

Procedure:

(1) Ignite the Bunsen burner

(2) Uncover the petri dish

(3) Dip the inoculation loop into the bacterial glycerol stock

(4) Streak the inoculation loop in the pattern shown in Figure B.1

(5) Seal the petri dish

(6) Incubate the petri dishes overnight at 37°C

Streaking out:

(1) Begin by streaking out a wide area at the top of the petri dish

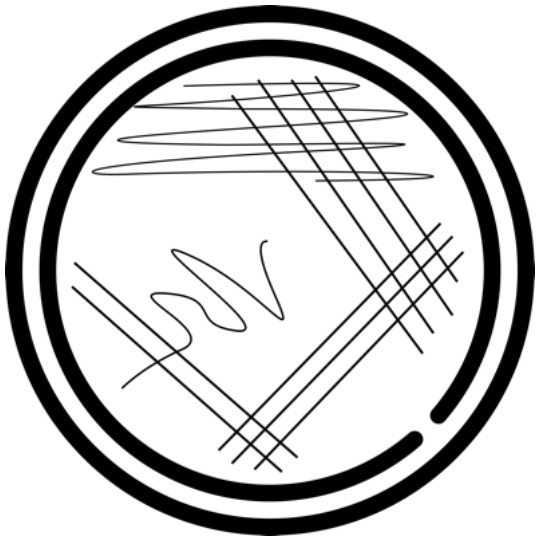


Figure B.1: Pattern for streaking out bacteria.

(2) Rotate the petri dish 90 degrees and draw four straight lines from your initial deposit across the dish's new top

(3) Rotate the petri dish 90 degrees again and draw three lines at the top, starting from the four lines drawn previously

(4) Rotate the petri dish 90 degrees once more and draw two lines at the top, starting from the three lines drawn before

(5) Conclude by drawing a squiggly line to the center of the petri dish

Overnight Culture

The overnight culture can be utilized directly, reinoculated into a larger volume of LB broth, or plated to cultivate either a lawn of bacterial growth or single isolated colonies.

Equipment:

- Bunsen burner
- Falcon tube (50 mL)
- Inoculating loops

Reagents:

- Growth media
- Antibiotic stock solution
- Bacterial glycerol stock (stored at -80°C) or petri dish with bacterial cultures

Procedure:

(1) Ignite the Bunsen burner and work closely with it

(2) Pour 5 mL of media into a 50 mL Falcon Tube

(3) Add 5 µL of each antibiotic from the 1000x stock solution

(4) Using an inoculation loop, pick up a colony from the petri dish or glycerol stock

(5) Stir the inoculation loop in the media to disperse the bacterial colony

(6) Tighten the lid until you feel resistance to allow limited airflow

(7) Incubate the Falcon tube overnight (12-16 hours) at 37°C

Plasmid Preparation

Plasmid preparation involves the extraction and purification of plasmid DNA. In this study, the QIAprep Spin Miniprep Kit [3] (cat. nos. 27104 and 27106) was utilized.

Golden Gate Assembly

Golden Gate assembly is a molecular cloning technique that enables researchers to simultaneously and directionally assemble multiple DNA fragments into a single construct using Type IIS restriction enzymes and T4 DNA ligase. This process is performed in vitro.

Equipment:

[3] https://www.qiagen.com/

- Thermocycler
- NanoDrop spectrophotometer
- PCR tubes

Reagents:

- Miniprepped DNA part plasmids
- Type IIS restriction enzyme (NEB Cat No. R3733S)
- Autoclaved distilled water
- T4 DNA ligase buffer (NEB Cat No. B0202S)

Procedure:

(1) Calibrate the NanoDrop spectrophotometer using 1.2 µL of distilled water

(2) Measure the concentration of each DNA part by analyzing 1.2 µL of each sample on the NanoDrop spectrophotometer

(3) Calculate the volume (in microliters) needed for the selected backbone using the formula:

$$\frac{40 \text{ fmol}}{\left(\frac{(\text{DNA concentration}\times 10^{-9})\times 10^{15}}{(\text{Part Size in bp}\times 607.4)+157.9}\right)}$$

(4) Calculate the volume (in microliters) needed for the selected DNA part using the formula:

$$\frac{80 \text{ fmol}}{\left(\frac{(\text{DNA concentration}\times 10^{-9})\times 10^{15}}{(\text{Part Size in bp}\times 607.4)+157.9}\right)}$$

(5) Add autoclaved distilled water to a total volume of 26 µL

(6) Add 2 µL of T4 DNA ligase buffer

(7) Add 1 µL of Type IIS restriction enzyme

(8) Add 1 µL of T4 DNA ligase to a total volume of 30 µL

(9) Place the PCR tube into the thermocycler

(10) Run the following thermocycler program:

- 37°C for 5 minutes
- 37°C for 5 minutes
- 16°C for 5 minutes

(11) Repeat steps 6 to 7 for 80 cycles

(12) Run the following thermocycler program:

- 37°C for 20 minutes
- 80°C for 20 minutes

(13) Keep the reaction at 12°C until samples are removed and stored at 4°C

Transformation

Transformation is the pivotal process of introducing a desired plasmid into a cell by traversing the cell membrane. The following protocol outlines transformation using the heat shock method.

Equipment:

- Petri dishes with antibiotic
- Bunsen burner
- Cell spreader
- Water bath
- Ice

Reagents:

- Chemical competent cells (NEB Cat No. C3019H)

- Outgrowth media (NEB Cat No. C3019H)
- Miniprepped DNA

Procedure:

(1) Thaw chemical competent cells on ice

(2) Use the NanoDrop to measure the plasmid concentration of miniprep

(3) Calculate the volume of miniprep in microliters needed to obtain 100 ng of plasmid

(4) Light the Bunsen burner and work close to it

(5) Add the determined microliters of plasmid to 0.05 mL competent cells

(6) Incubate for 30 minutes

(7) Heat shock at 42°C in a water bath for 45 seconds

(8) Quickly transfer to ice (-20°C) and incubate for 2 minutes

(9) Add 200 mL of outgrowth media (included with cells)

(10) Incubate at 37°C for one hour with a rotation speed of 250 rpm

(11) Spread the cells onto petri dishes and allow them to grow overnight at 37°C

Flow Cytometry

Flow cytometry is a technique used to analyze characteristics of cells as they flow in a fluid stream through a beam of light. This method allows for the simultaneous measurement of multiple parameters, such as cell size, count, and fluorescence intensity, providing valuable insights into cellular properties and functions. The flow cytometry analysis in this study followed the protocol provided in the supplemental materials of the work by Nielsen et al. [175].

Induction

The induction protocol was employed to drive the circuit into its eight possible states before transferring it into a 96-well plate for analysis in the plate reader to measure fluorescence and cell growth.

Equipment:

- Bunsen burner
- Eppendorf tubes (1.5 mL)
- 96-well plate

Reagents:

- Inducers
- Growth media
- Antibiotic
- Double-transformed cells containing the circuit
- Single-transformed cells containing the reporter unit (RPU)

Procedure:

(1) Prepare eleven Eppendorf tubes

(2) Ignite the Bunsen burner and work nearby

(3) Add 1 mL of growth media to each Eppendorf tube

(4) Add 1 µL of antibiotic to each tube

(5) Add 10 µL of inducer to each tube

(6) Add 10 µL of overnight culture containing bacteria to each tube

(7) Transfer 200 µL of media containing induced cells per well to the 96-well plate

Plate Reader

The plate reader was utilized to monitor cell growth over time by measuring OD and fluorescence intensity as cells grew and remained viable

Equipment:

- Plate reader
- 96-well plate containing induced cells

Procedure:

(1) Set the excitation wavelength

(2) Set the emission wavelength

(3) Set the OD measurement wavelength

(4) Set the shaking speed

(5) Set the averaging parameters

(6) Select samples, standards, and blanks for measurement

(7) Run the plate reader for the specified duration (e.g., 15 or 50 hours)