August 2020
Master's Degree Thesis

A Parallel Approach to Perform Threshold Value Analysis and Verification of Genetic Logic Circuit Models

Graduate School of Chosun University

Department of Computer Engineering

Sanaullah

A Parallel Approach to Perform Threshold Value Analysis and Verification of Genetic Logic Circuit Models

유전자 회로의 문턱값 분석 및 검증의 효율적인 병렬 처리 연구

August 28, 2020

Graduate School of Chosun University

Department of Computer Engineering

Sanaullah

A Parallel Approach to Perform Threshold Value Analysis and Verification of Genetic Logic Circuit Models

Advisor: Prof. Jeong A. lee
Co-Advisor: Dr. Hasan Baig

A thesis submitted in partial fulfillment of the requirements for a Master's degree

May, 2020

Graduate School of Chosun University

Department of Computer Engineering

Sanaullah

사나 울라 석사 논문 승인

위원장 조선대학교 교수 **김판구** (인)

위 원 조선대학교 교수 **강문수** (인)

위 원 조선대학교 교수 **이정아** (인)

2020년 06월

조선대학교 대학원

Table of Contents

List of Abbreviations and Acronyms

DNA	Deoxyribonucleic Acid
$RNAs$	Ribonucleic Acids
GFP	Green Fluorescent Protein
YFP	Yellow Fluorescent Protein
$SBOL$	System Biology Open Language
$SBML$	Systems Biology Markup Language
GDA	Genetic Design Automation
CAD	Computer-Aided Design
EDA	Electronic Design Automation
$D-VASim$	Dynamic Virtual Analyzer and Simulator
$LabVIEW$	Laboratory Virtual Instrument Engineering Workbench
ODE	Ordinary Differential Equation
SSA	Stochastic Simulation Algorithm
kd	Degradation Rate
OC_{DUTH}	User Define Upper Threshold Value
OC_{DLTH}	User Define Lower Threshold Value
PT	Possible Threshold Value
OC_E	Output Consistency
T_E	Estimate Time Delay
O_C	Output Concentration
C_{inC}	Input Concentration Level
S_T	Settling Time
T_D	Assumed Time Delay
$in-vitro$	Laboratory Experiments
S_P	Species Input
$I/O\ Species$	Output/Output Species
EM	External Modifiers
DRM	Direct Reaction Method
FRM	First Reaction Method

List of Figures

List of Tables

Abstract

A Parallel Approach to Perform Threshold Value Analysis and Verification of Genetic Logic Circuit Models

Sanaullah
Advisor: Prof. *Lee*, *Jeong – A*, Ph.D.
Department of Computer Engineering
Graduate School of Chosun University
Co-Advisor: Dr. *Hasan Baig*, Ph.D.
Center for Quantitative Medicine
University of Connecticut Health, USA

Synthetic biology is an area of research where researchers in engineering and biology are synergetically working to create biological parts, systems, and devices. At the heart of a synthetic biological system exist genetic logic circuits. Such circuits seek to implement desired logic functions inside a biological cell. The design of gene circuits commonly requires simulating their mathematical models and to identify whether or not the circuit is reacting appropriately. Compared to electronic circuits, genetic circuits show a non-deterministic behavior and are therefore much harder to characterize. Normally, genetic circuits are designed and experimented with in-vitro. This process is characterized by a very tedious design-flow including careful handling of the apparatus. To which end, several approaches have been designed to enable modeling, simulation, and testing of gene circuits in-silico. The computational tools accelerate the design process and help reduce human error in the process. It is obviously useful to have

a computational tool that enables all of the above in addition to providing a lab-like dynamic circuit a simulation where specie concentrations could be modified and their effect observed on run-time. D-VASim is the first tool to provide this functionality where users can change the input protein concentrations during the simulation. This enables faster prototyping of the circuit design and further accelerates the circuit design process. A fundamental step before the simulation of a genetic circuit is its threshold value estimation. D-VASim implements an algorithm for threshold value and timing analyses of genetic circuits. However, a parallel implementation of the algorithm can reduce the computation time of D-VASim which becomes even more necessary when it is known that D-VASim can still take up to hours for processing complex genetic circuits.

In this thesis, we propose a parallel approach for faster threshold value analysis and verification of genetic logic circuits. We also modify the algorithm to reduce inconsistencies in the output species concentration. The estimated threshold value is now more accurate which can be seen from experimentation for long simulation runtimes. We introduce further modifications to reduce the deviation in the algorithm runtimes across multiple simulations runs at the same parameter settings. Overall, with the proposed modifications to D-VASim, its usability has considerably improved and the threshold value and timing analysis algorithm are now significantly faster as in the case of NOT gate by up to 16 times.

한 글 요 약

유전자 회로의 문턱값 분석 및 검증의 효율적인 병렬 처리 연구

사나 울라
지도 교수: 이정아
컴퓨터공학과
대학원, 조선대학교
공동-지도 교수: 하산 바이 그
정량 의학 센터
미국 코네티컷 대학교

합성생물학은 공학자와 생물학자들이 생물학적 부품, 시스템, 장치를 만들기 위해 함께 연구하고있는 융합연구분야로서, 합성생물시스템에서는 유전자 논리회로가 핵심적 역할을 한다. 유전자 논리회로는 세포 내 생체회로에서 발생하는 논리함수를 구현한다. 유전자 회로의 설계는 일반적으로 그들의 수학적 모델을 시뮬레이션하고 회로가 이에 따라 적절하게 반응하는지 여부를 식별하는 것으로, 전자회로에 비해 유전자 회로는 비결정론적 행동을 보여서 특징짓기가 훨씬 어렵다. 유전자 회로는 체외 실험(in-vitro)으로 설계되고 실험된다. 이 과정은 실험 기구를 세심하게 다루는 등 매우 지루한 설계 흐름이 특징이다. 이러한 연유로, 컴퓨터 모형을 활용한 유전자 회로의 모델링, 시뮬레이션 및 테스트를 수행하는 몇 가지 설계방식이 그동안 제안되었다. 컴퓨터 모형의 계산 도구는 설계 프로세스를 가속화하고 사람의 실수를 줄이는 데 도움이 된다. 이에 더 나아가서, 가상의 실험실에서 사용자가 시밀레이션 중에 투입되는 특정

농도를 변경하고 이의 영향을 런타임 중에 관찰할 수 있는, 동적 회로 시뮬레이션을 제공할 수 있다면, 매우 유용할 것이다. D-VASim은 사용자가 시뮬레이션 중에 단백질 농도 투입을 동적으로 변경할 수 있는 기능을 제공하는 첫 번째 설계도구이다. 이를 통하여, 회로 설계의 프로세스를 더욱 가속화할 수 있고, 보다 빠른 프로토타이핑이 가능하였다. 유전자 회로의 임계값 분석은 시뮬레이션과 논리 검증 이전의 기본적인 단계로서, D-VASim은 유전자 회로의 임계값 및 타이밍 분석을 위한 알고리즘을 구현하였지만, 알고리즘의 병렬화를 고려하지는 않아서, D-VASim 은 유전자 회로의 문턱값(임계값) 분석과 검증 시간이 여전히 충분히 빠르지 못하다는 문제가 있었다.

본 논문에서는, D-VASim의 고속처리를 위하여, 유전자 회로 문턱값 분석 및 검증의 병렬처리 기법을 제시한다. 이와 더불어, 추정된 문턱값으로 유전자 논리회로 결과값이 유지되고, 안정화될 수 있도록 알고리즘을 수정하였으며, 이를 시밀레이션을 통하여 확인하였다. 또, D-VASim에서 동일한 매개변수를 사용하는 경우, 시뮬레이션 실행 시간의 편차가 감소될 수 있도록, 알고리즘을 수정하였다. NOT 논리회로의 경우, 본 논문에서 제시한, 유전자 회로의 임계값 및 타이밍 분석을 위한 알고리즘의 병렬처리는 기존의 방식보다 16배 빠른 속도로 처리하였으며, 본 논문에서 제시한 이러한 기법들은 D-VASim의 고속처리를 가능하게 하여, 이의 효용성을 크게 개선하였다.

I. Introduction

A. An Overview of GDA Tools

Biologists and engineers are working together on genetic devices to improve their reliability and accuracy [1]. Researcher have worked on the development of genetic design automation (GDA) tools [2] for the use of design, test, verification and synthesis processes of genetic circuits. One of the main research focus is the GDA tools' role in reducing laboratory experimentation by offloading most of the design process to computer analysis **.** Computational techniques in genetic circuit design involve ordinary differential equations and stochastic simulations. Standardized computerized format is a must in order to realize genetic circuits. The two widely used standards to represent the behavior and the structure of a genetic model are the Systems Biology Markup Language (SBML) [3], and the Synthetic Biology Open Language (SBOL) **r**espectively. SBML allow users to define the behavior of a circuit and SBOL, on the other hand, is used to illustrate genetic designs graphically.

The understanding of a biological system of any living organism includes, for example, genes, which contain all the information related to biological organism (like height, hair color, blood group and so on). Furthermore, each gene is a part of DNA which produces a specific protein through the processes called transcription followed by the processes of translation [5]. There are two types of gene expressions, constitutive and regulated. The gene is explicitly constant in constitutive gene expression and is dependent on regulate gene expression. Lac Operon is one of the standardized example of transcription processes and it was the first gene regulatory network to be explored correctly and clearly [6]. The experiment results in Figure 1, of lac operon show that no transcription process

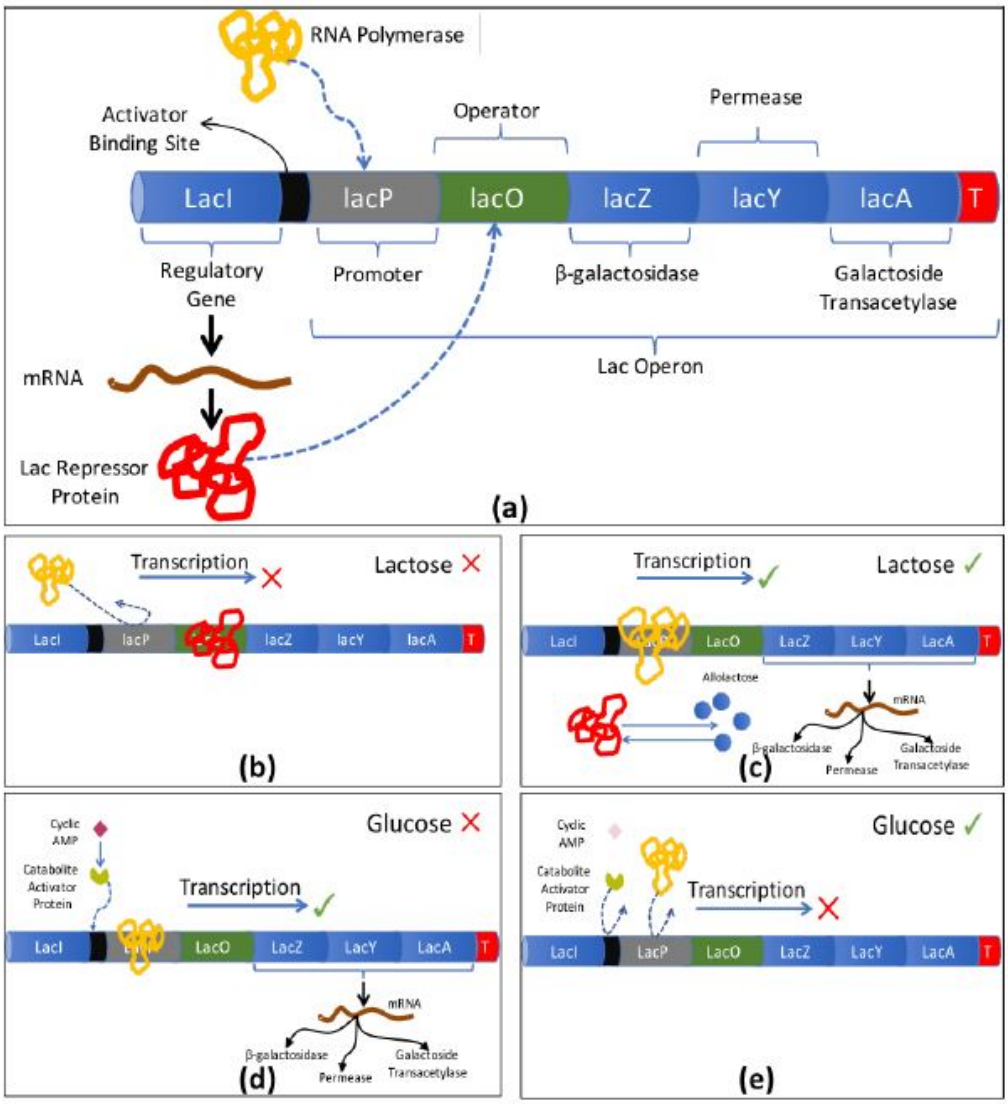


Figure 1: Transcriptional regulation of lac operon. (a) Structure of Lac operon. No transcription when (b) lactose is absent and (e) glucose is present. Transcription begins when (c) lactose is present and (d) glucose is absent. (Image courtesy Hasan Baig, PhD Thesis - Methods and Tools for the analysis, verification and synthesis of genetic logic circuits.) [4]

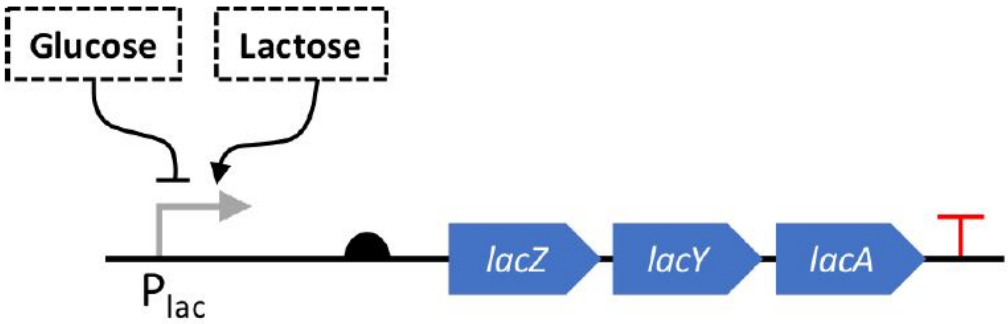


Figure 2: SBOL visual representation (or SBOLv) of lac operon genetic system. (Image courtesy Hasan Baig, PhD Thesis - Methods and Tools for the analysis, verification and synthesis of genetic logic circuits.)[4]

occurs when lactose is absent and glucose is present. Transcriptions begins when lactose is present and glucose is absent. The genetic logic in lac operon can be extracted but it gives a low-level detail of how lac operon works [4],

SBOL is the standard way to represent the high-level diagrams of genetic systems [4]. In Figure 2, the SBOL visual (or SBOLv) diagram of the genetic system for lac operon would be something similar to the genetic logic diagram of lac operon in Figure 1. And, in this thesis we focus the use of SBML models of these genetic logic circuits to test the tools and method [7], [8].

B. Motivation

1. Challenges

Biologists are not only working on biological systems but they are also constructing an artificial complex system of biological components, in a similar way as electronic circuits have been designed and constructed [9], [10]. However, in comparison to electronic circuits which have same physical quantity as input

and output signals, genetic circuits have different quantities at their inputs and outputs [11]. Additionally, electronic engineer develops different logic gates (such as, AND, NAND and NOT gate) to control the function of cells. The current methodology to design genetic circuits is trial and error laboratory experimentation. In this method (trial and error) thousands of circuits are tested but only few of them actually work. For this reason, the current genetic circuit design process is costly and time consuming. To this end, a methodology which allow users to capture and analyze the stochastic behavior of biological systems dynamically in-silicon is necessary. Successful computer simulations can increase the chance that the model will work as expected in living organisms [12]. Thus, there are several challenges related to the field of GDA. Successfully solving these issues will not only increase designer productivity but will also increase the reliability and robustness of genetic circuit models designed. These challenges are,

1. Virtual experimentation; it would be very helpful for biologists or design engineers to have a tool which allows them to digitally perform laboratory experiments.

2. Timing and Threshold Analysis: Unfortunately, no GDA tools allow users to perform timing and threshold values analysis but in contrast with EDA tools, users can perform timing analysis.

3. Effortless circuit designing: simple and straightforward mechanism should be developed for biologist and designer engineer.

4. Automatic logic Validation: Automatic compilation is possible with the validation. This automated approach is very helpful for analyzing the

genetic logic circuits.

2. Standards

SBML and SBOL [7], [8] are two major standards to represent a genetic model's behavior and structure respectively. SBML standard is independent of any other specific software language which is why it can be used to exchange necessary information of a biological model among the different software tools. However, the different SBML-synthesis tools may have their own icons to represent a standard and which is why the same processes are represented differently in different tools. In order to solve this problem and consistently represent biological models, the SBOL standard developed a document for all genetic models. SBOL standards generate the same XML document for each tool in order to use by other software tools.

Several GDA tools have been developed to assist users in the design of genetic devices [13]. These tools use different standards and codes. There are about 290 tools which support the SBML Standard for construction and simulation of genetic logic circuits. About 30 of them are GDA tools which support sequence editing, design composition, optimization and technology mapping. Of these tools, some serve as toolboxes or plugins for commercial applications including Oracle, MATLAB, and Mathematica while some are application programming interfaces, and others are independent tools for model construction or simulation such as iBioSim [14], CellDesigner [15], or COPASI [16]. However, none of them allows users to interact with the model during runtime.

3. Dynamic Virtual Analyzer and Simulator Tool

Dynamic Virtual Analyzer and Simulator or D-VASim [17] is developed on the Laboratory Virtual Instrument Engineering Workbench (LabVIEW) programming platform, which is a graphical programming language commonly used for rapid project development and prototyping (www.ni.com). The basic data flow of D-VASim virtual simulation and analysis environment tools is shown in Figure 3. D-VASim [17] is an interactive virtual laboratory environment for the simulation and analysis of genetic logic circuit models represented in System Biology Markup Language or SBML [7]. D-VASim applies both deterministic and stochastic analyses to extract and validate Boolean logic from the SBML model. D-VASim helps replace the time-consuming in-vitro experiments (laboratory experiments) needed to analyze and design genetic circuits [7]. The input to D-VASim is genetic circuit model where D-VASim allows the user to interact with the model, observe its behavior, and make direct changes in the concentration of input protein(s) at run-time. D-VASim is capable of estimating the threshold value for a genetic circuit model [18]. The threshold value refers to the minimum input protein(s) concentration that causes the average output protein(s) concentration to cross the input protein(s) concentration. The Boolean function of a genetic logic circuit can be verified by extracting the logic behavior from the simulation data [19]. Thus threshold value and timing analysis are used to verify a genetic logic circuit's intended boolean function. This methodology is useful in helping the user extract the Boolean logic of the bio-models without any prior knowledge of the expected behavior of the model.

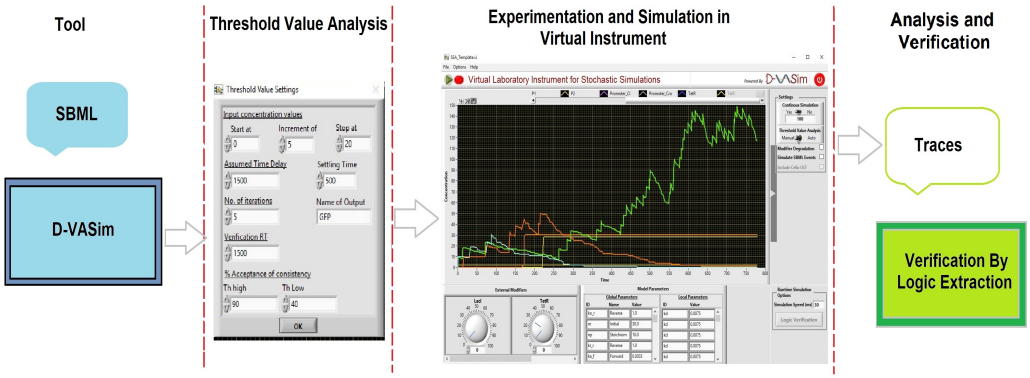


Figure 3: The basic flow of D-VASim. The tool takes user-specified parameters in threshold concentration analysis prior to circuit simulation. The boolean logic is extracted and verified from the simulation data.

4. Threshold Value and Timing Analysis

D-VASim is the first synthetic biology tool which is capable of estimating the threshold value for a genetic logic circuit model automatically [18]. D-VASim helps replace the time-consuming in-vitro (laboratory) experiments needed to analyze and verify the genetic logic circuit models [7]. D-VASim can take up to few hours in estimating and verification of the threshold values for complex genetic logic circuit models owing to its serial execution necessitated by data dependency. Such analysis and verification times are still acceptable as compared to the laboratory experiments where number of days are required to spend only for a single input combination and specific parameter set. However, there is potential for much faster approaches to genetic circuit simulation.

5. SBML Models

D-VASim applies the both deterministic and stochastic analyses to obtain and validate SBML Model Boolean logic [17]. D-VASim is the only bio-simulation

tool available that provides a dynamic simulation environment. The main drawback of D-VASim is its current lack of support for genetic circuits with inputs larger than 3 [19].

C. Thesis Contributions

The aim of this research is to enhance the advancement of GDA tools for threshold value and timing analysis, verification, and Boolean logic extraction of genetic logic circuits. The thesis targets are the development of the following methods and algorithms to address the aforementioned challenges.

1. We introduce a fast approach to verify the possible threshold value that a parallelized procedure. The updated algorithm is used by D-VASim v1.3.

2. We propose a parallel-implemented algorithm significantly faster by upto 16 times in comparison to the algorithm previously employed in D-VASim. We also optimize the algorithm for consistent run-times across multiple simulation runs at the same parameter settings reducing the worst-case standard deviation in run-times from 6.637 to 1.841. The proposed algorithm also estimates the threshold value more accurately as compared to previous algorithm.

3. We enhance D-VASim to be able to process *n*-input genetic logic circuits.

D. Thesis Layout

The thesis is structured as follows. In Chapter II, we discuss previous work related to the proposed threshold value analysis and verification of genetic logic circuit

models. We then introduce a fast and robust threshold value and timing analysis approach in Chapter III and describe the D-VASim's input limitation followed by the *n*-input methodlogy. Furhtermore, experimental results on case study have been presented. Finally, we conclude the thesis in Chapter IV.

II. Threshold Value Analysis and Verification of Genetic Logic Circuit Models

A. Threshold Value and Timing Analysis

In the wet lab, biologists are either provided with the ready-made biological model available in a test tube or are given a specification or recipe from which to prepare the model in the lab. Their duty is to analyze the model and verify its functional behavior. This analysis is done interactively, among other things, by increasing the molecular concentration of input species at any instant of time and observing the effects.

As mentioned in Chapter 1, Dynamic virtual analyzer and simulator or D-VASim [17] is the first interactive virtual-laboratory environment for the simulation and analysis of genetic logic circuit models. In the context of genetic logic circuits, the threshold value is defined as the minimum input protein(s) concentration that causes the average output protein(s) concentration to cross the input protein(s) concentration [18]. Each circuit has a different threshold value, which needs to be estimated. The threshold value and timing analyses are used to verify the Boolean function of a genetic logic circuit by extracting the observed logic behavior from the simulation results. To be able to determine the Boolean functionality of a genetic circuit is useful in two ways: first, it allows a user to verify more complex genetic logic circuits, built by cascading several genetic logic gates; second, it helps the user to extract the Boolean logic of a bio-model even when the user does not have any prior knowledge about the expected behavior of the model. Next, the knowledge of Boolean logic helps the user design a circuit for some practical application, for example, a gene circuit

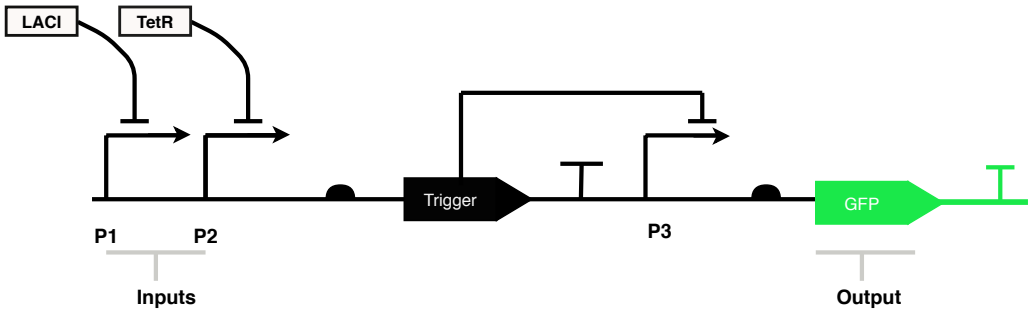


Figure 4: An example OR genetic logic circuit. LacI and TetR are input proteins while GFP is the output protein.

that can identify and attack a particular type of cancer by triggering the immune system.

A 2-input genetic OR gate is shown in Figure 1 with the promoter proteins LacI and TetR as the inputs and green fluorescent protein (GFP) as the output. These two input proteins may have different threshold concentrations. For example, LacI may trigger the output when its concentration is 5 molecules and TetR may trigger the output at 10 molecules. Setting the upper input threshold to 10 would provide the correct answer, i.e., for the molecular counts (LacI, TetR = 4,7), the gate remains off, whereas for (LacI, TetR = 7,4), LacI may trigger the output, but the output may not be considered logic 1 until the output concentration increases beyond 10 molecules. Simulation results indicate that the triggered output in such cases is highly unstable, i.e., it oscillates between logic 0 and logic 1 [20]; therefore, such states are considered transitory or undefined states.

In the previous example, instead of estimating the threshold values of each input protein separately, the algorithm in D-VASim estimates the global upper and lower threshold values for all inputs and provides a single threshold concentration of 10 molecules for the entire circuit. D-VASim treats the entire

circuit as a black box such that individual threshold values and intermediate circuit components do not matter. D-VASim obtains the threshold concentration that is sufficient to trigger each circuit component from the inputs to the final output. However, the individual threshold concentrations of all the intermediate circuit components can also be analyzed in D-VASim.

B. Algorithm Explanation

D-VASim uses Gillespie's direct stochastic simulation algorithm [21], [22] to simulate the stochastic nature of the biological models, and it supports ten different continuous solvers for deterministic simulations. The algorithm used in D-VASim for threshold value and propagation delay analysis of genetic circuits is shown in Algorithm 1.

The propagation delay is the time from when the input concentration reaches its threshold value until the corresponding output concentration crosses the same threshold value [18]. The algorithm requires some user-defined parameters: C_{in} is the value of input protein(s) concentration at which the tool begins the threshold value analysis. The input concentration is increased by Inc in each iteration to observe if the resulting concentration affects the output concentration, whereas C_{inE} specifies the input concentration at which the analysis should stop. The algorithm also needs an initial assumption of the propagation delay, T_D. It is necessary to wait until this time has elapsed before changing the input combination for extracting the correct logic behavior of a circuit. The tool starts with this assumed value of T_D and continues to improve toward the approximate value. For simulation, if no node in the genetic circuit is initialized, then some outputs exhibit unstable behavior for a certain amount of time. The parameter S_T

Algorithm 1 Threshold value analysis

```
1: procedure BEGIN
       INITIALIZE (C_in, Inc, C_inE, T_D, S_T, i, O_S, V_T,
       OC_DUTh, OC_DLTh)
       /*
       C_in = Initial input concentration at which the analysis should start from
       Inc = Increment Value: Value added to the previous concentration level until the concentration level reaches C_inE
       C_inE = End concentration of input at which the analysis should stop
       T_D = Assumed time delay
       S_T = Settling time
       O_S = Name of output specie
       i = Number of iterations to verify the consistency of results
       V_T = Amount of time to verify a model for each iteration i
       OC_DUTh and OC_DLTh = user-defined percentage acceptance of output consistency for upper and lower threshold
       values respectively
       /*
2:     for1 all possible input combinations do
3:      if (C_inC == 0 ) then  /* C_inC = current input concentration level*/
4:        Determine initial output concentration (C_Oinit)
5:      else
6:        while1 (C_inC <= C_inE) do
7:          while2 (T_C1 <= T_D) do  /* T_C1 = current time 1 */
8:            Execute Simulation
9:            if (C_OS > C_inC) then /* C_OS = output concentration of selected specie*/
10:             PT = C_inC  /* PT = Possible Threshold Value*/
11:             /* Verification process*/
12:             for2 number of iterations i do
13:               while3 (T_C2 <= V_T) do  /* T_C2 = current time 2*/
14:                 Execute Simulation
15:                 if (T_C2 >= S_T) then
16:                   Trigger the input to the value of PT
17:                   Store the output concentration data in array
18:               end
19:                 Take running average of all output i arrays
20:           end
21:           Estimate time delay (T_E) and consistency (OC_E)
22:           Terminate loop2
23:       end
24:       if (C_OS > C_inC) do
25:         if (OC_E > OC_DUTh) then
26:           Consider lower threshold value = 0 if not found already
27:           Return the results and terminate all loops
28:         else if (OC_E < OC_DLTh)
29:           Save lower threshold level and resume analysis
30:         else
31:           Resume analysis
32:       C_inC = C_inC + Inc
33:       T_C1 = T_C2 = 0
34:     end
35:    end
36:   end
37: end
```

(settling time) helps approximately specify the amount of time during which the circuit outputs are expected to become stable before triggering the circuit inputs for a correct threshold value estimation. Subsequently, the algorithm verifies the

obtained threshold value for a predefined i (number of iterations). The algorithm obtains the average propagation delay by running the model for the length of time defined by V_T (number of time to verify each iteration i) in each i^{th} iteration and also identifies the extent to which the average output for the estimated threshold value is consistent. This is demonstrated using the sample parameter values shown in Table 1.

Parameters		**Values**
Input concentration	C_{in}	0
Incremental value	Inc	2.75
End concentration	C_{inE}	15
Assumed time delay	T_D	800
Settling time	S_T	200
Number of iterations	i	10
Amount of time to verify each iteration (i)	V_T	1000
User defined upper threshold value percentage	OC_{DUTh}	90
User defined lower threshold value percentage	OC_{DLTh}	30

Table 1: Sample values of the user-defined parameters for threshold value analysis

1. Possible Threshold Value

Consider the time-scale plots drawn for the sample values from Table 1 shown in Figure 5. The algorithm triggers the input concentration to certain levels after a specific time delay and observes the output behavior of the circuits. It requires a specific input combination to determine the threshold value, which indicates that all the input combinations need to be checked one by one until the one that triggers the output concentration is found. For some circuits such as OR, NOR, AND, and NAND, the output transition can be observed by triggering both of the inputs to the same concentration level at the same time. The algorithm, therefore,

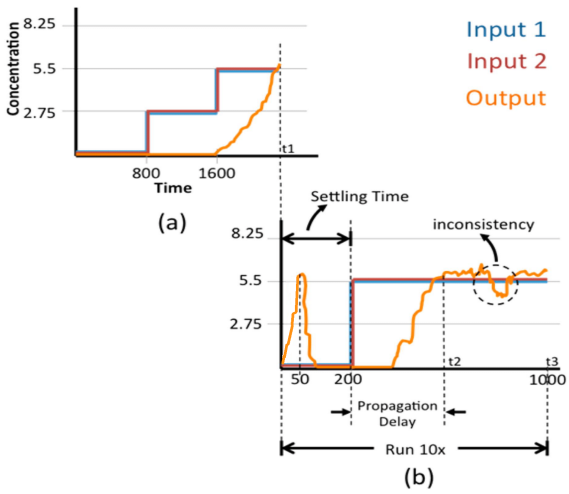


Figure 5: Sample time scale plots of the genetic AND gate.(Image courtesy Hasan Baig and Jan Madsen - Simulation Approach for Timing Analysis of Genetic Logic Circuits) [18].

triggers both the inputs combinations from 00 to 11 first, instead of following the traditional pattern of $00 \rightarrow 01 \rightarrow 10 \rightarrow 11$. Because of this, the estimation process is relatively faster for some circuits, - for example, an AND gate [18].

For a genetic AND gate, the case of the input combination "11" is shown in Figure 5. To determine whether the output concentration crosses the input concentration, we need the initial output protein concentration for the input logic combination "00". Therefore, both the inputs are maintained at zero concentration for the first 800 time units (T_D), and the average initial output concentration is obtained. After the specified time (T_D) has passed, the input concentration is incremented to the next level as indicated by line 32 in Algorithm 1. Notably, the algorithm also operates for the case where the average initial output concentration is high - for example, a NOT gate - by iteratively incrementing the input concentration until the output concentration falls below the input concentration level.

2. Threshold Value Verification

The point t_1 (time 1) in Figure 5a is where the output protein concentration crosses that of the input. This is the possible threshold value of 5.5 molecules for the given example. Subsequently, the value for which the algorithm initiates a separate loop and simulates the circuit model for a defined number of iterations i, which is 10 in this case, needs to be verified. This loop is defined in lines $13-21$ in Algorithm 1. The input protein must only be triggered after the initial concentration of output is settled as shown in Figure 5b to measure the propagation delay correctly. At this time, the algorithm allows the user to specify a period of time called the settling time (S_T), as mentioned before, after which the initial output is expected to become stable. Only after S_T amount of time has elapsed does the algorithm trigger the inputs. The propagation delay may be incorrectly estimated if a small value of S_T is chosen; therefore, depending upon the complexity of the given circuit, the user should carefully select this value. In the example shown in Figure 5b, if instead of 200, a value of 50 time units is chosen for S_T, the algorithm would estimate the propagation delay to be zero. This is because, at 50 time units, the output concentration would already be above the threshold level.

The output data from all i iterations (10 in this case) were averaged to obtain the average estimated propagation delay and the inconsistency present in the sample time-scale plot for the estimated threshold values. Inconsistency means the fall of output concentration down to the level of input concentration or below after it has crossed the input concentration. The concept is illustrated in Figure 5b. Inconsistency is calculated by averaging the output data that are less than the input concentration after the output crosses the input concentration for the first

time. Thus, in Figure 5b, the consistency is estimated between the two points t_2 and t_3 and this time duration is specified by the parameter V_T in Table 1, which is the amount of time to verify the model in each iteration i.

The algorithm accepts the estimated threshold value based on the user-defined parameter, % acceptance of consistency, shown as OC_{DUTh} (for upper threshold level) and OC_{DLTh} (for lower threshold level) in Table 1. The results are accepted if the estimated consistency is greater (for upper threshold) and less (for lower threshold) than the user-defined values, OC_{DUTh}, and OC_{DLTh}, respectively. This is shown in the lines $26-31$ of Algorithm 1. The results are otherwise discarded and the algorithm resumes the analysis from point t_1, shown in Figure 5a. The percentage output consistency is calculated using Equation 1.

$$\%\ output\ consistency = \frac{O_{t2-t3} - D}{O_{t2-t3}} * 100 \tag{1}$$

where O_{t2-t3} is the average output data between $t2$ and $t3$, and D (deviation) which defines the number of times the output data is found to be deviated from the expected (greater or less than the) threshold value. This deviation is defined differently for the two cases. When the initial input concentration is low, D is the number of times the output is observed to be less than the possible threshold value as shown in Figure 5b and vice versa. If the user-defined output consistency threshold cannot be satisfied, the current results are discarded and the analysis is resumed from point t_1.

III. A Parallel Threshold Value Analysis Algorithm

As mentioned in Chapter 1, there are opportunities to improve and optimize the threshold value and timing analysis algorithm which is currently used in D-VASim tool which can be used to simulate a large and complex genetic circuits in a feasible time. Our contribution lies in vectorizing the whole algorithm along verification procedure with modifications for analyzing the threshold value more robust, accurately and consistently for long enough time.

A. Proposed Algorithm

In Algorithm 1, the main $for1$ loop simulate the possible threshold value between from line $2-37$ and the $for2$ loop verified the possible threshold value from line $12-20$ have been parallelized by taking care of data dependencies minimizing the time complexity from $O(N^2)$ to $O(N)$ where n is the number of input combinations of the given circuit model. The previous serial algorithm pursue a pre defined pattern in checking all the possible input combinations to estimate that 1-input combination can be used for estimation of threshold value. As mentioned in previous Chapter 2, instead of the old pattern $00 \rightarrow 01 \rightarrow 10 \rightarrow 11$, the proposed algorithm followed $00 \rightarrow 11$ to stimulate the evaluation process for some genetic circuit models like the AND logic gate for which the output concentration can be analyzed to triggering the both inputs at the same time to high. In the same way, to ensure this accelerated and robusted estimation, the proposed algorithm must also follow a pre-defined pattern for genetic circuit models with more than 2-inputs. There are some internal data dependencies between the different input combinations. We absorb the algorithm and carefully

removed those such data dependencies by maintaining any problematic to local as global variables and applied some other programming techniques which have eliminated a pattern the need to carefully define. In our proposed algorithm, instead of analyzing the input combinations one-by-one in finding the specific input combinations, all the possible input combinations are checked in parallel pattern given the required computation cost. Thus, it is possible to fully parallelize $for1$ to obtain reduced computation time.

The $for2$ loop in Algorithm 1 which is the main execution in the verification method to runs for user-defined i number of iterations. In each ith iteration, the Gillespie stochastic algorithm [21], [22] is called which execute an array of numbers from which a number selected randomly and then the two internal Next Reaction Value and Next Reaction Time parameters are calculated. After this, in the end of each ith iterations, the stored simulation data is calculated an average. In proposed algorithm, the Gillespie algorithm calls only once time, and select i numbers randomly to calculates i number of Next Reaction Time and Next Reaction Value, which are assigned to ith iteration. After this, the verification process can be execute in parallel pattern and the execution data can be averaged afterwards. The speed-up achieved from parallelization, more minimization in computation time is accomplished by replacing all the repeated call of the Gillespie's algorithm with a single call only. Also, all the calls of memory to the accumulated average which is necessary as an operand to estimate a runing average in Algorithm 1, the 19 line is no longer required in the proposed algorithm. This can be shown in Figure 6 where task flow of previous algorithm and the proposed algorithm are displaying. It should be noticed that few of the tasks have been omitted in Figure6 due to the focus only on those process which have been optimized. Parallelization of $for1$ and $for2$ result in a massive

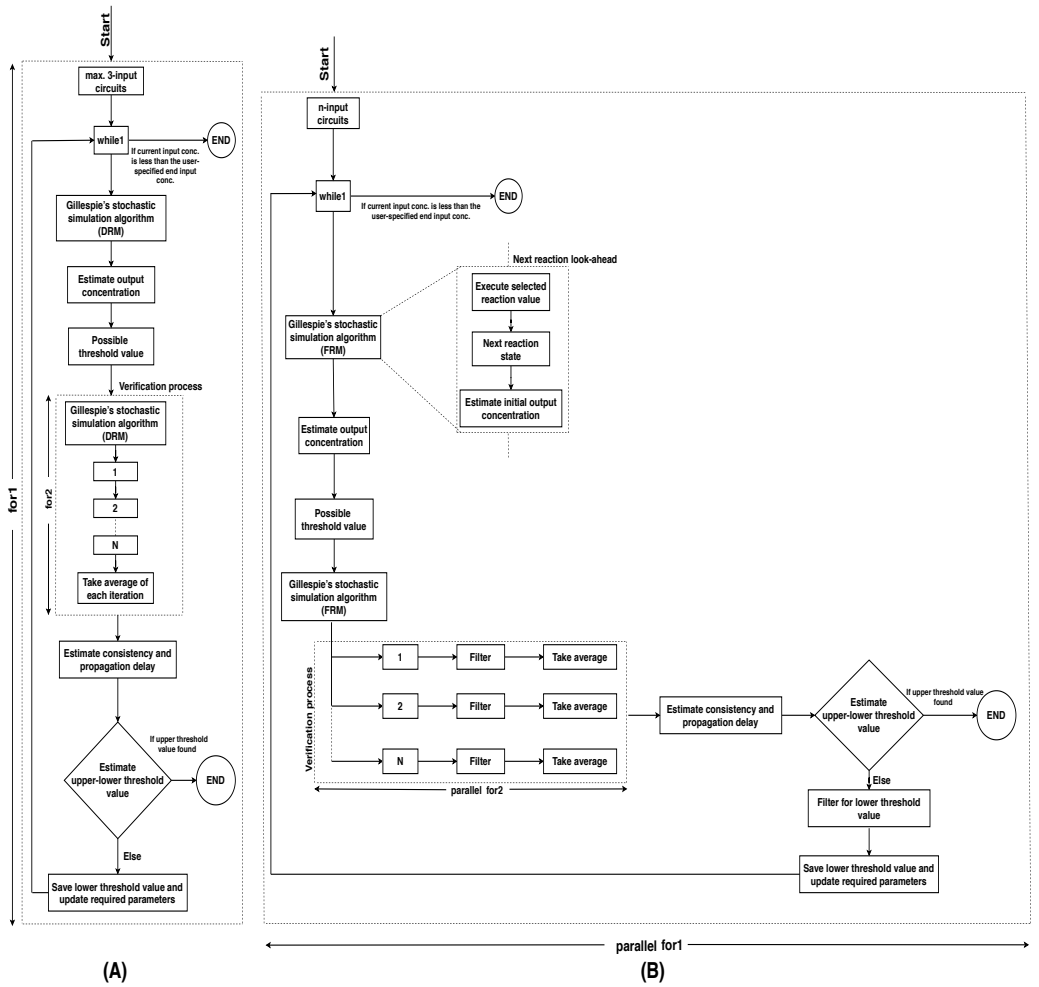
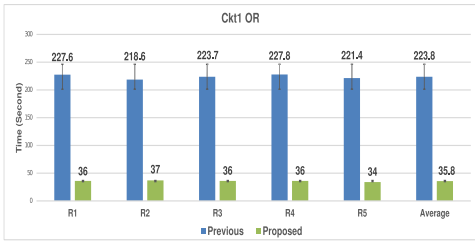


Figure 6: Task flow of (A) previous algorithm [18] and (B) the proposed algorithm

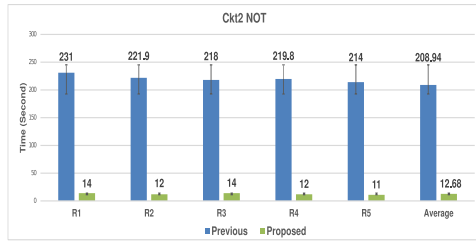
minimization of the execution time taken by the previous to proposed algorithm by upto 16 times in our test cases.

1. Results and Analysis

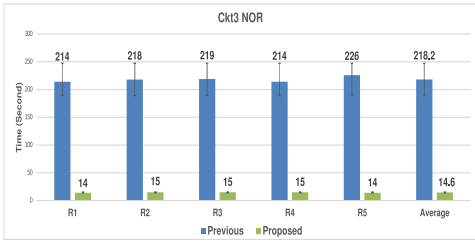
The speed-up results are shown in Figure 7 for five two inputs genetic logic circuit models are taken from [12] and [13]. The description of these genetic



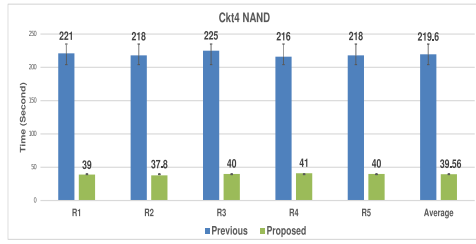
(a)



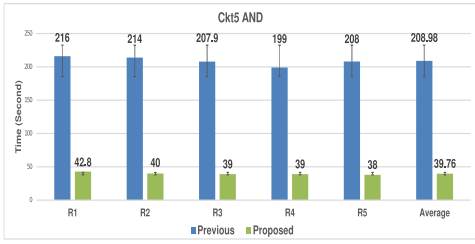
(b)



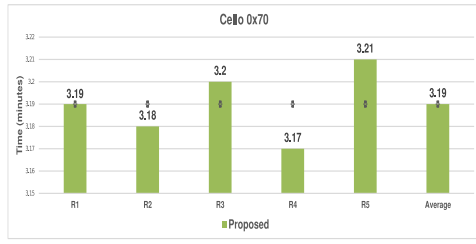
(c)



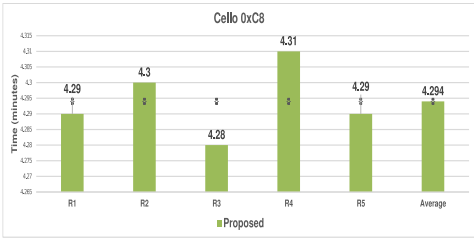
(d)



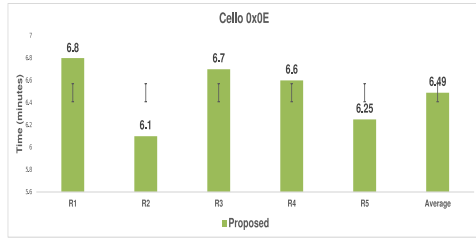
(e)



(f)



(g)



(h)

Figure 7: (a e) Times taken in threshold value analysis by the previous algorithm and proposed algorithm for five test cases along with standard deviation error bars. (f h) Results for the circuits from [7] along with the standard deviation error bars.

circuit models is available in the aforementioned references. The genetic circuits components and their kinetic reactions or interactions are defined as SBML models form and we used those models on D-VASim for execution and performed the threshold value and timing analysis. The threshold value of all genetic circuits was analyzed five times each circuit at the same user-defined parameters for the previous serial algorithm and our proposed parallel algorithm. The time taken by each of the the five simulated results of all circuits runs and its average are shown in Figure 7. for each genetic circuit in separate plots. It can be shown from the presented results in Figure 7 that the parallelized algorithm implementation has minimized the computation time required to obtain the threshold value and timing analysis manyfold thereby adding to the utility of the virtual laboratory environment. It can be seen that the maximum speed-up is not more than 16 times (Ckt2 NOT) and no less than 5 execution times in the shown test cases. The circuits from 4 named 0x70, 0xC8, 0x0E are circuits with inputs greater than 2 and therefore cannot be run using the previous algorithm. They are thus only tested and verified using the parallel algorithm.

The reason behind this because the whole process of obtaining the threshold value include simulated stochastic computations. If possible threshold value is obtain incorrect, the verification execution method fails and the whole execution has to be repeated again by increasing the input value in C_{in}. Therefore, the expanse of deviation time is notably high in the different execution. It is clear that, an increase in the number of inputs results in an exponential increase in the number of input cases to be evaluated. This should result in an exponential rise in computational resources and/or time taken. In the context of how the proposed algorithm's performance scales as input complexity increases, we have tested 2-, 4-, 6- and up to 8-input circuits. Their respective runtimes obtained were: 31 sec

(NAND Gate), 2.14 minutes (custom 4-input circuit), 3.12 minutes (custom 6-input circuit) and 9 minutes (for a custom-designed 8-input circuit). The parallel algorithm can theoretically process all input cases in parallel, however, in practice, the time taken would depend upon the computational resources at hand. The times obtained above do present an exponential trend; but, the comparison between circuits with different inputs in terms of the time taken also largely depends upon the number of levels of the circuit. Consider Fig. 7 for example, where results for different 3-input Cello circuits are presented. We can see that the threshold value estimation for Cello 0x70 (2 gates) takes 3.19 minutes on average while for Cello 0x0E (4 gates), it takes 6.49 minutes on average [7].

The previous serial execution algorithm uses the Direct Method (DM) [21], [22] approach of Gillepsie's stochastic algorithm. DM used the time at any reaction happens in whole procedure and complete that reaction type later. The algorithm normally took that time at which any reaction happens next. In our proposed algorithm we used the other approach which is the First Reaction Method(FRM) [21], [22], which estimate the Next Reaction Time for each reaction first and then uses the only one with the smallest value of time (i.e., the next one). Both algorithms (DM and FRM) basically uses the same principles. These two Gillepsie's algorithms produce the same results; the only difference in their method of execution. Moreover, FRM is based on reaction-oriented nature that has been noted to be more appropriate parallel computation than DM [23]. Due to this enhancement, the proposed parallel implementation in the algorithm has been observed to obtain the threshold value in more consistent run-times across multiple execution runs with the same user-defined parameter settings. In Figure 7 (Standard Deviation error bars in each Graph), we present the standard deviation in the runtime calculated from five simulation runs for five

(a e) of the aforementioned 2-input genetic circuits with the serial algorithm and the proposed algorithm. The worst case of standard deviation runs of the serial algorithm is 6.637, whereas that in parallel proposed algorithm is 1.841. It is proved from these executions results that proposed algorithm is much more consistent in the multiple run-time.

The % (percentage) output consistency estimation is already described in Equation 1. It uses the output average data O_{t2-t3} between the time $t2$ and $t3$ points shown in Figure 5. The previous algorithm uses concentration values of all the output stored in between the time $t2$ and $t3$ to estimate the O_{t2-t3}. Then, the threshold value estimated the output consistency based does not produce the consistent output. When the simulation execute the runs long enough, the output concentration at some points, falls below the concentration of input as you can seen in Figure 8(a) which is an erroneous behavior. What is correct output is that the data of output below the concentration of input must be filtered out while estimating the output consistency where *Filter* is aforementioned filter as shown in Figure 6(b).

The other enhancement in proposed parallel algorithm is to be treated this problem of inconsistent output estimation is what we call the next state look ahead. Lets suppose, the pre-defined algorithm parameters are, initial concentration value C_{in} = 5 and the increment value Inc = 10. Lets assume also, the final output-concentration increases the initial-input concentrations when triggered at C_{in} = 5 molecules units. The previous serial algorithm will used to treat it as a possible threshold value and start to execute the verification procedure. If it verified, then it will output as the obtained threshold value. Again, for long enough simulation run-time, the output concentration will be inconsistent and start to falls below the input concentration level. With this the proposed next-

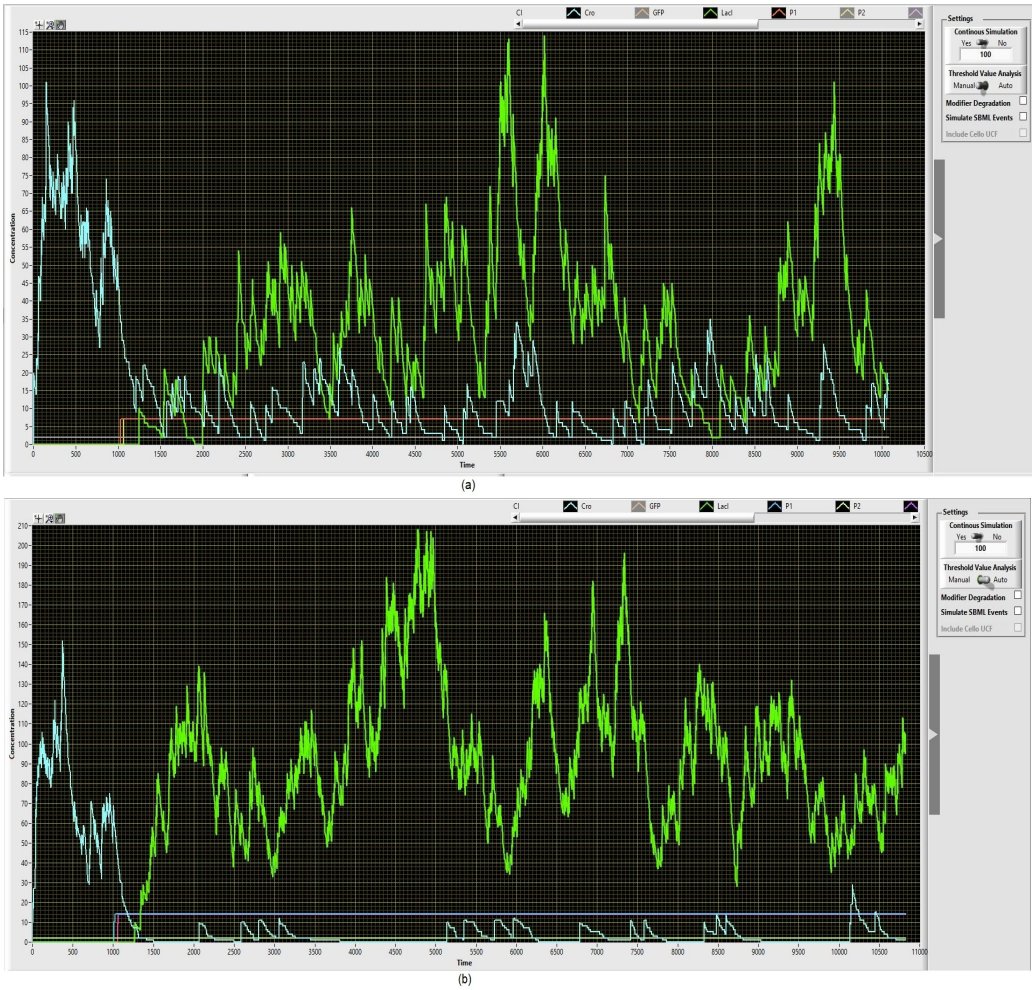


Figure 8: (a) Threshold value falling below the input concentration when the simulation is run for a sufficiently long time. (b) Threshold value consistently remaining above the input concentration after the proposed changes

state method look-ahead, the parallel algorithm will not be picked C_{in} = 5 as the possible threshold value. Rather, it also checks state of next-reaction between the current value C_{in} = 5 and C_{in} = 15 (incremented by Inc) values. Lets consider, the value of next reactions obtain at C_{in} = 8; 10; 11; 13. The algorithm will verify first whether the value of output concentration increases the value of input

concentration at any of these obtain states. We have also implemented in the proposed modification this next-state look-ahead approach in a parallel fashion. Then it estimates the success-ratio or at what % (percentage) of C_{in} initial input values of the output concentrations increased the input concentrations. If the success-ratio is tunable more than 70%, it used C_{in} = 5 values for verification process. Otherwise, it does not move forward to the verification of C_{in} = 5 value, rather, it used to increments C_{in} value by the defined *Inc* parameter value to C_{in} = 15 and need to repeats the process again. In this approach, the value which is finally used to verified as the possible threshold value is more accurate and the values of output concentrations are stable for long enough simulation run-times remains consistently stable above the values of input concentration. The comparative results is shown in Figure 8 where the value of output concentration of the previous serial algorithm is start to falling below to the value of input concentration at some points (present in Figure 8(a)) whereas the value of output concentration based on the proposed algorithm estimated threshold value never falls below the value of input concentration during the whole 10;000 unit time simulation runs (shown in Figure 8(b)).

2. Threshold Value Analysis of *n*-input Genetic Circuits

The main drawback of D-VASim is its current lack of support for genetic circuits with inputs larger than 2. With parallelization, the algorithm now supports processing of n-input circuits. The serial algorithm uses static pre-defined input combinations. Now, we first dynamically generate all possible input combinations using the specie information before the main algorithm execution. To test the performance of the parallelized algorithm on circuits with inputs larger

Threshold Value Analysis						
Genetic Logic Circuit	I_1 (*min.*)	I_2 (*min.*)	I_3 (*min.*)	I_4 (*min.*)	I_5 (*min.*)	Average (*min.*)
4-input Circuit	2.13	2.03	2.31	2.12	2.11	2.14
6-input Circuit	3.09	3.03	3.15	3.21	3.13	3.122
8-input Circuit	3.95	3.83	4.1	3.91	4.27	4.012

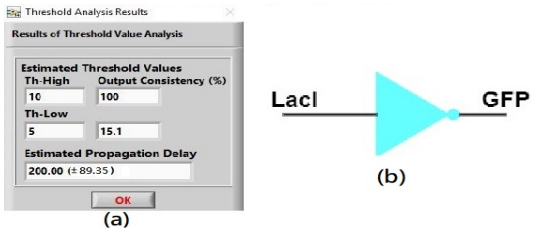
Table 2: Proposed Threshold Value Analysis Algorithm Results

than 2, we designed a reasonably complex 8-input circuit in iBioSim shown in Simulation section. For the circuit shown, the simulation was run five times using the parameters defined before and the time taken in all five runs is presented in Table 2. Clearly, the 8-input circuit takes 9:178 minutes on average. Thus, we have successfully managed to introduce support for processing n-input circuits in feasible time.

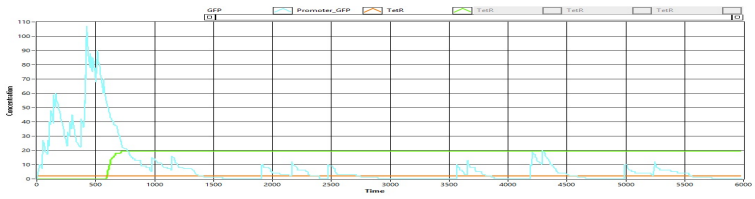
3. Experimental Results

The algorithm was tested on the standard SBML models [8] which are the first 5 circuits in Table 2. These are all only $1-$ and $2-$input genetic circuits. However, the $4-, 6-$, and $8-$inputs genetic circuits 3 were manually designed using the iBioSim tool [24].

Ckt 1 - NOT Gate The simulation results of a genetic NOT gate circuit:



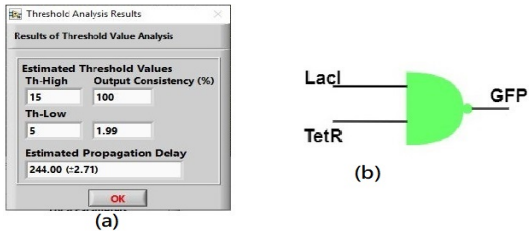
(a)



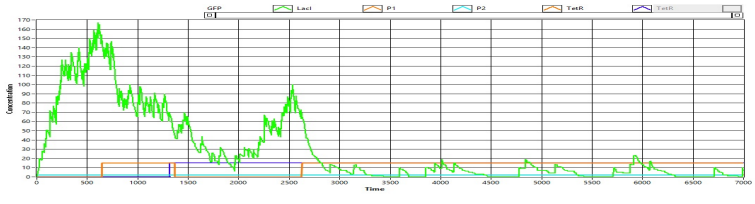
(b)

Figure 9

Ckt 2 - NAND Gate The simulation results of a genetic NAND gate circuit:



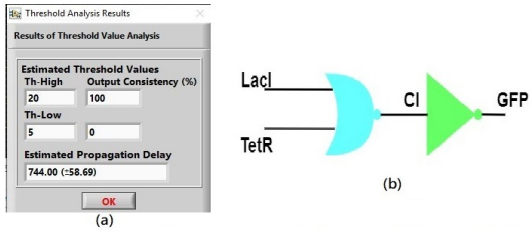
(a)



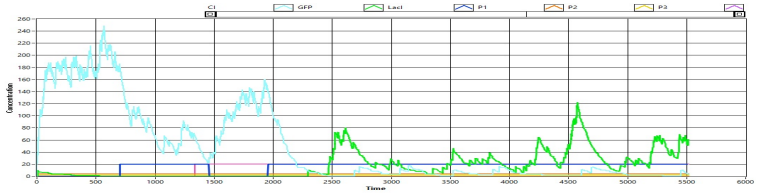
(b)

Figure 10

Ckt 3 - AND Gate The simulation results of a genetic AND gate circuit:



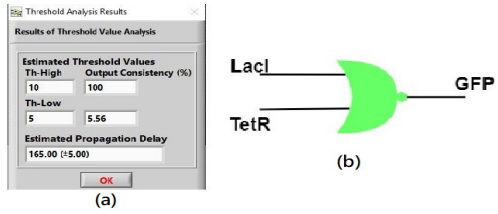
(a)



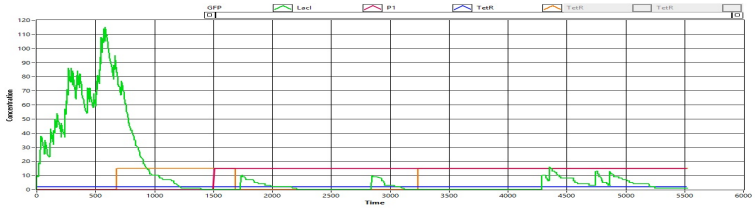
(b)

Figure 11

Ckt 4 - NOR Gate The simulation results of a genetic NOR gate circuit:



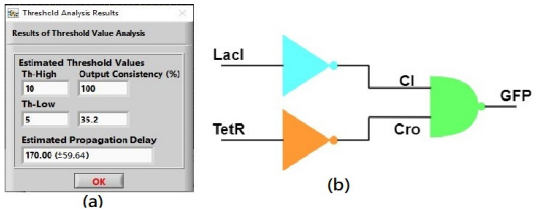
(a)



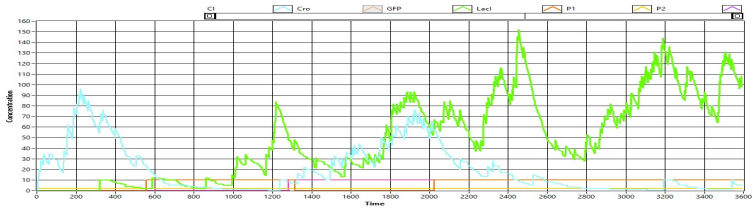
(b)

Figure 12

Ckt 5 - OR Gate The simulation results of a genetic OR gate circuit:



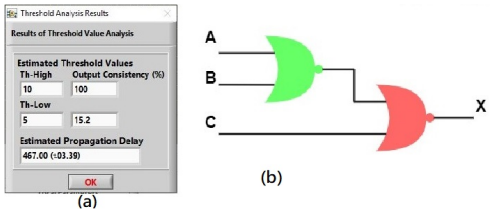
(a)



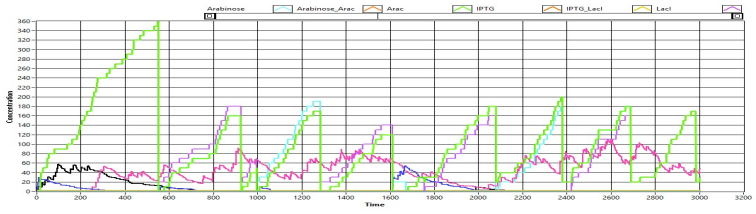
(b)

Figure 13

Cello - Ox70 Circuit The simulation results of a genetic Ox70 circuit:



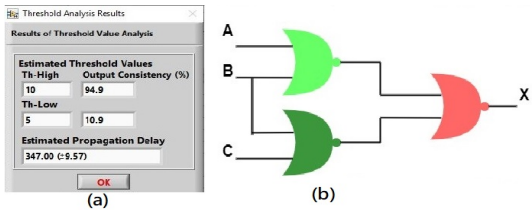
(a)



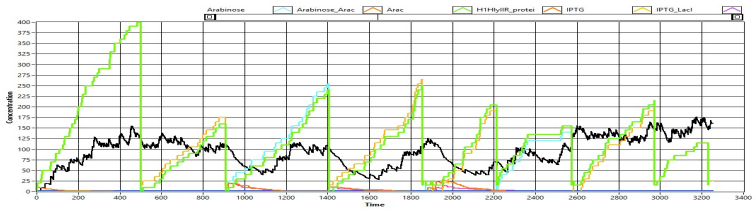
(b)

Figure 14

Cello - OxC8 Circuit The simulation results of a genetic OxC8 circuit:



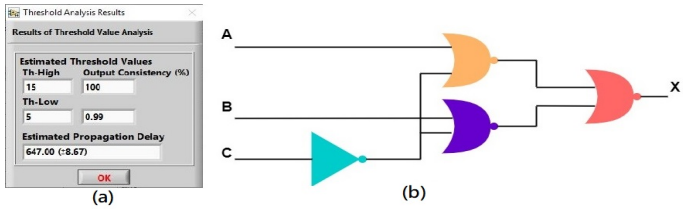
(a)



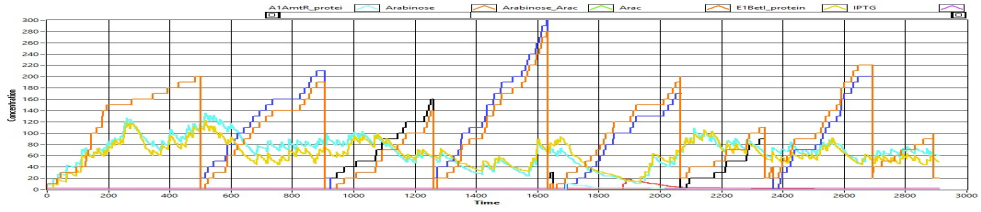
(b)

Figure 15

Cello - OxE Circuit The simulation results of a genetic OxE circuit:



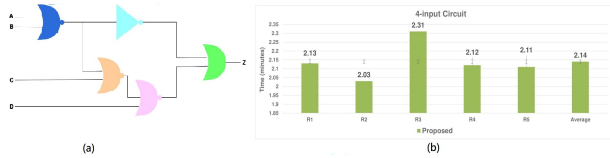
(a)



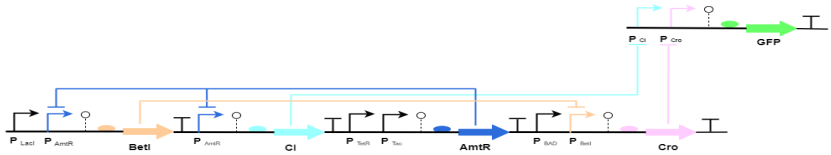
(b)

Figure 16

4-input Circuit The schematic circuit diagram and SBOL representation of the genetic circuit, 4-input are shown below,



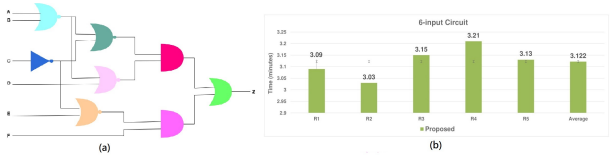
(a)



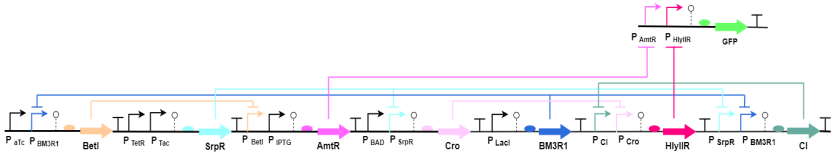
(b)

Figure 17

6-input Circuit The schematic circuit diagram and SBOL representation of the genetic circuit, 6-input are shown below,



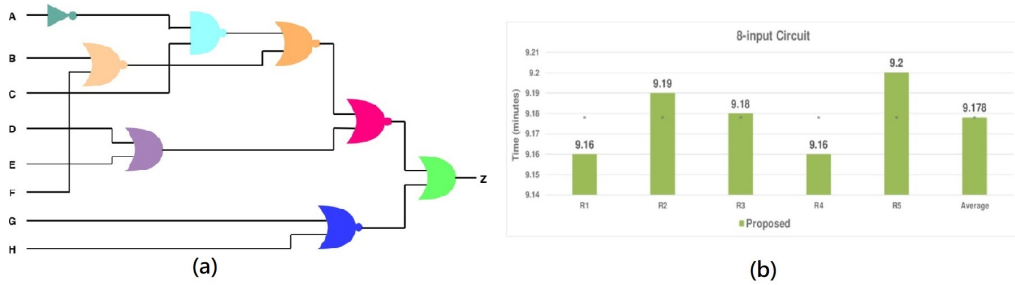
(a)



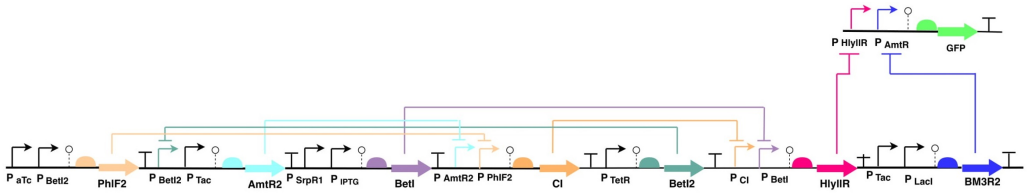
(b)

Figure 18

8-input Circuit The schematic circuit diagram and SBOL representation of the genetic circuit, 8-input are shown below,



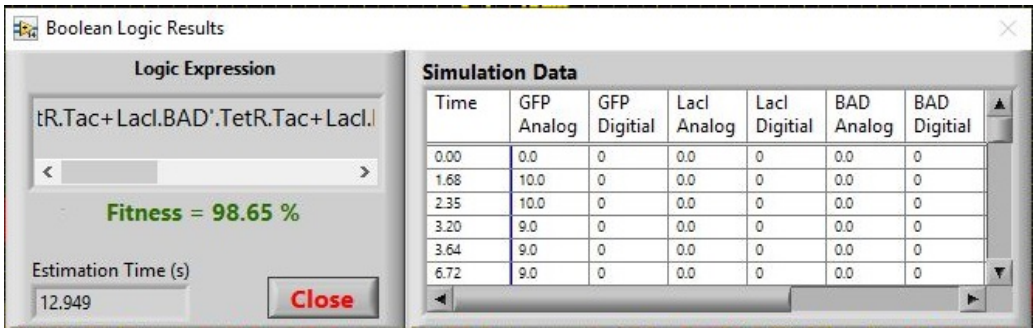
(a)



(b)

Figure 19

4-input Circuit The experimental results of the genetic 4-input circuit are shown in the Figures a and b, respectively.

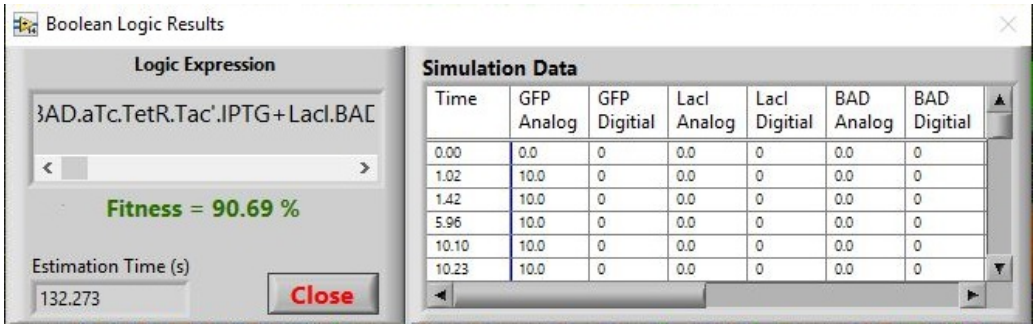


(a)

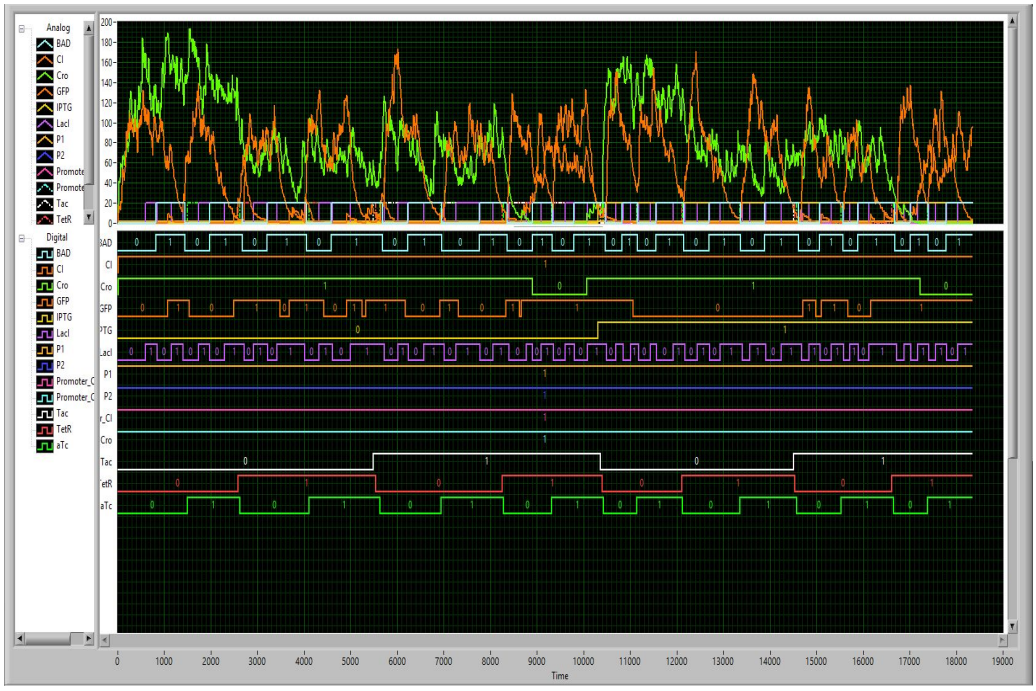
(b)

Figure 20: (a) Boolean expression estimated by the logic analysis algorithm, and (b) Circuit behavior when input is logic 0 and 1.

6-input Circuit The experimental results of the genetic 6-input circuit are shown in the Figures a and b, respectively.



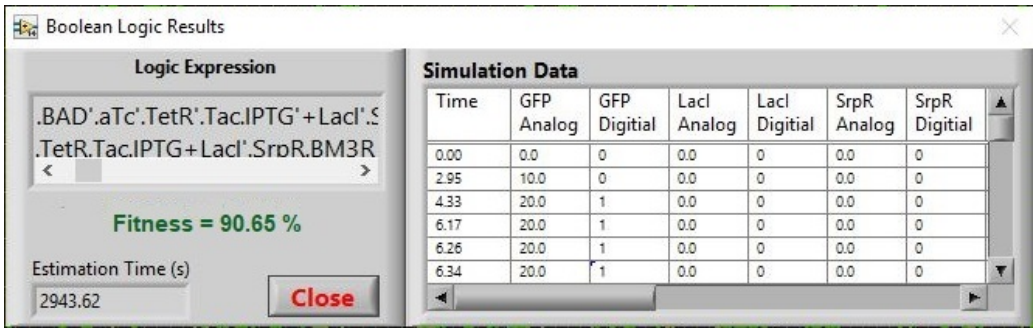
(a)



(b)

Figure 21: (a) Boolean expression estimated by the logic analysis algorithm, and (b) Circuit behavior when input is logic 0 and 1.

8-input Circuit The experimental results of the genetic 8-input circuit are shown in the Figures a and b, respectively.



(a)

(b)

Figure 22: (a) Boolean expression estimated by the logic analysis algorithm, and (b) Circuit behavior when input is logic 0 and 1.

IV. Conclusion

D-VASim provides researchers with a laboratory-like dynamic virtual environment for simulating and analyzing genetic logic circuits. This process includes the estimation of the threshold value of the given circuit, as the threshold concentrations are different for different circuits. In this thesis,

1. We presented an approach towards paralellizing the threshold value and timing analysis and achieved significant reduction in threshold value analysis latency in all of the test cases.

2. We also modify the algorithm to reduce inconsistencies in the output specie concentration. The estimated threshold value is now more accurate which can be seen from experimentation for long simulation runtimes.

3. We introduce further modifications to reduce the deviation in the algorithm runtimes across multiple simulations runs at the same parameter settings.

4. Overall, with the proposed modifications to D-VASim, its usability has considerably improved and the threshold value and timing analysis algorithm is now significantly faster as in the case of NOT gate by up to 16 times with worst-case standard deviation in runtime reduced to 1.841 from 6.637.

Publications

A. Journals

Sanaullah, Hasan Baig, Jan Madsen, and Jeong-A Lee, “A Parallel Approach to Perform Threshold Value and Propagation Delay Analyses of Genetic Logic Circuit Models”, *ACS Synthetic Biology Publications*, (Under Review)

B. Conferences

Sanaullah, Hasan Baig, and Jeong A Lee, “Accelerating the Threshold and Timing Analysis of Genetic Logic Circuit Models”, in *11th International Workshop on Bio Design Automation (IWBDA)*, IWBDA 2019, University of Cambridge, UK, 2019, pp. 64–65.

References

[1] Arkin, Adam, "Setting the Standard in Synthetic Biology", *Nature biotechnology*, vol. 26, no. 7, pp. 771–774, 2008.

[2] Marchisio, Mario A and Stelling, Jörg, "Computational Design Tools for Synthetic Biology", *Current opinion in biotechnology*, vol. 20, no. 4, pp. 479–485, 2009.

[3] Hucka, Michael and Finney, Andrew and Sauro, Herbert M and Bolouri, Hamid and Doyle, John C and Kitano, Hiroaki and Arkin, Adam P and Bornstein, Benjamin J and Bray, Dennis and Cornish-Bowden, Athel and others, "The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models", *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.

[4] Hasan Baig, *Methods and Tools for the Analysis, Verification and Synthesis of Genetic Logic Circuits*, `https://backend.orbit.dtu.dk/ws/portalfiles/portal/138073348/phd456_Baig_H.pdf`.

[5] Crick, Francis, "Central dogma of molecular biology", *Nature*, vol. 227, no. 5258, pp. 561–563, 1970.

[6] Jacob, François and Monod, Jacques, "Genetic Regulatory Mechanisms in the Synthesis of Proteins", *Journal of molecular biology*, vol. 3, no. 3, pp. 318–356, 1961.

[7] Nielsen, Alec AK and Der, Bryan S and Shin, Jonghyeon and Vaidyanathan, Prashant and Paralanov, Vanya and Strychalski, Elizabeth A and Ross, David and Densmore, Douglas and Voigt, Christopher

A, “Genetic Circuit Design Automation”, *Science*, vol. 352, no. 6281, aac7341, 2016.

[8] Myers, Chris J, *Engineering Genetic Circuits*. CRC Press, 2016.

[9] Gardner, Timothy S and Cantor, Charles R and Collins, James J, “Construction of a Genetic Toggle Switch in Escherichia Coli”, *Nature*, vol. 403, no. 6767, pp. 339–342, 2000.

[10] McAdams, Harley H and Shapiro, Lucy, “Circuit Simulation of Genetic Networks”, *Science*, vol. 269, no. 5224, pp. 650–656, 1995.

[11] Bernardi, D and DeJong, JT and Montoya, BM and Martinez, BC, “Bio-bricks: Biologically Cemented Sandstone Bricks”, *Construction and Building Materials*, vol. 55, pp. 462–469, 2014.

[12] *SBML Software Matrix*, `http://sbml.org/SBML_Software_Guide/SBML_Software_Matrix`.

[13] SBML Team and others, *The SBML Software Guide*, 2011.

[14] Myers, Chris J and Barker, Nathan and Jones, Kevin and Kuwahara, Hiroyuki and Madsen, Curtis and Nguyen, Nam-Phuong D, “iBioSim: A Tool for the Analysis and Design of Genetic Circuits”, *Bioinformatics*, vol. 25, no. 21, pp. 2848–2849, 2009.

[15] Funahashi, Akira and Matsuoka, Yukiko and Jouraku, Akiya and Morohashi, Mineo and Kikuchi, Norihiro and Kitano, Hiroaki, “CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks”, *Proceedings of the IEEE*, vol. 96, no. 8, pp. 1254–1265, 2008.

[16] Hoops, Stefan and Sahle, Sven and Gauges, Ralph and Lee, Christine and Pahle, Jürgen and Simus, Natalia and Singhal, Mudita and Xu, Liang and Mendes, Pedro and Kummer, Ursula, "COPASIâ€" A Complex Pathway Simulator", *Bioinformatics*, vol. 22, no. 24, pp. 3067–3074, 2006.

[17] Baig, Hasan and Madsen, Jan, "D-VASim: An Interactive Virtual Laboratory Environment for the Simulation and Analysis of Genetic Circuits", *Bioinformatics*, vol. 33, no. 2, pp. 297–299, 2017.

[18] H. Baig and J. Madsen, "Simulation approach for timing analysis of genetic logic circuits", *ACS synthetic biology*, vol. 6, no. 7, pp. 1169–1179, 2017.

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

[20] Mark B, *Complete Digital Design: A Comprehensive Guide to Digital Electronic and Computer System Architecture.* McGraw-Hill Press, 2003.

[21] Gillespie, Daniel T, "A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions", *Journal of computational physics*, vol. 22, no. 4, pp. 403–434, 1976.

[22] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions", *The journal of physical chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.

[23] C. Dittamo and D. Cangelosi, "Optimized parallel implementation of gillespie's first reaction method on graphics processing units", in *2009*

International Conference on Computer Modeling and Simulation, IEEE, 2009, pp. 156–161.

[24] *iBioSim*, `https://github.com/MyersResearchGroup/iBioSim`.

Acknowledgement

All praise to the Lord of the World alone. This thesis made possible, my heartfelt appreciation for the constant reassurance and prayer of my parents and words cannot describe the level of patience with which they persevere my absence. The exceptional guidance and support from Professor Dr. Jeong-A Lee, my supervisor during the course of this degree. I am also very grateful to Professor Dr. Hasan Baig for his constant advice in capcaity of my co-supervisor, insights and suggestions and I will forever be grateful to him. I also thank my friends and lab members (too many to list here but you know who you are!) for providing support and friendship that I needed. I would like to thank Umar Afzal for being supportive throughout my time and for helping me in each and every part of my life here. I think of him as a brother.

I dedicate this thesis to my family for their constant support and unconditional love.