A Faster DiSH: Hardware Implementation of a Discrete Cell Signaling Simulator

Kevin Gilboy, Khaled Sayed, Niteesh Sundaram, Kara N. Bocan, and Natasa Miskov-Zivanov, *Member, IEEE*

Abstract—**Development of fast methods to conduct *in silico* experiments using computational models of cellular signaling is a promising approach toward advances in personalized medicine. However, software-based cellular network simulation has runtimes plagued by wasted CPU cycles and unnecessary processes. Hardware emulation affords substantial speedup, but prior attempts at hardware implementation of biological simulators have been limited in scope and have suffered from inaccuracies due to poor random number generation. In this work, we propose several simulation schemes utilizing novel random update index generation techniques for step-based and round-based stochastic simulations of cellular networks. Our results show improved runtimes while maintaining simulation accuracy compared to software implementations.**

I. INTRODUCTION

Biological systems are stochastic in nature; biochemical reactions have a certain probability of occurring, even when the concentrations of reactants are known, and the environmental conditions are controlled. This is in direct contrast to a deterministic system, in which a given set of inputs will always produce the same set of outputs. Computational models can mimic the randomness of biological systems to gain insight into the nature of particular biological mechanisms, such as the role a specific ligand plays in a signaling network, or the substance concentration needed to initiate a reaction. In addition, computational models have been shown to be invaluable as researchers can run a large number of *in silico* experiments that would be inefficient or impractical if attempted *in vivo* or *in vitro*. Such models allow for predicting wet-lab outcomes and for shedding insights into the nature of biological systems.

As computational models are versatile and can be constantly updated via input from the user, they have the potential to be used in clinical applications such as personalized medicine. A model of a physiological system could be created and then modified in real time to more accurately reflect the unique characteristics of a specific person. The model could then be used to predict patient outcomes and inform treatments.

K. Gilboy, K. Sayed, and K. N. Bocan are with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15213. E-mail: {kevingilboy, k.sayed, knb12}@pitt.edu

N. Sundaram is with the School of Medicine, University of Pittsburgh, Pittsburgh, PA 15213. E-mail: nis101@pitt.edu

N. Miskov-Zivanov is with the Department of Electrical and Computer Engineering, Department of Bioengineering, and the Department of Computational and Systems Biology, University of Pittsburgh, Pittsburgh, PA 15213. E-mail: nmzivanov@pitt.edu

Although many software (SW) modeling and analysis tools for personalized medicine are being developed, they are slower and less efficient than hardware (HW) emulator systems designed for the same purpose [1-4]. Specifically, an implementation in a Field Programmable Gate Array (FPGA) not only allows for significant speedup of the same algorithms, when compared to execution of a SW code, it is also a commonly used HW platform for parallel processing, and thus, well suited for concurrently conducting many *in silico* experiments. In this paper, we propose a circuit design, and a HW framework called Fast-DiSH (Discrete, Stochastic, Heterogeneous), written in SystemVerilog [11], for emulating cellular signaling networks. We compare the accuracy and runtime of our HW framework with SW simulation for (i) different simulation schemes that capture the stochasticity of cellular signaling networks, and (ii) different scenarios that represent network conditions or external influences. Implementing stochasticity in HW is a non-trivial task, and therefore, we also propose novel random index generation techniques for stochastic simulation schemes.

II. BACKGROUND

To demonstrate the functionality of our HW emulator design, we use the T cell differentiation model described and analyzed in [5, 6]. For both HW and SW simulations, we use an executable discrete T cell model consisting of elements and their associated update rules listed in [7, 8], and we compare the simulations conducted in Faster-DiSH with those conducted using SW simulator DiSH described in [7].

A. T Cell Differentiation Model

In [5], the authors presented a logical model of the circuitry controlling T cell differentiation. They focused on effects of antigen dose and timing on CD4+ naïve T cell differentiation into two main subtypes: helper T cells (Th), which promote immune response, and are characterized by the secretion of a cytokine interleukin 2 (IL2) [9]; and regulatory T cells (Treg), which suppress the immune response, and are characterized by the expression of transcription factor forkhead box P3 (Foxp3) [10]. The model in [5] recapitulates experimental results suggesting that a low antigen dose induces differentiation into Treg cells, while a high dose induces differentiation into Th cells, and showed that timing of antigen dose removal plays a crucial role in determining the differentiation outcome.

B. DiSH Simulator

DiSH simulator, previously described in [7], is a SW simulator that enables simulation with several different schemes to capture features of transient and steady-state behavior of cellular signaling networks. These schemes offer control over

978-1-5386-1311-5/19/$31.00 ©2019 IEEE

timing and order of updating elements in a model according to their update rules.

We implemented within our HW simulation framework the following update schemes from DiSH: *simultaneous* (SMLN); *random-order sequential* (RSQ), both *round-based* (RB-RSQ) and *step-based* (SB-RSQ); and *grouped random-order sequential* (RSQ-g), both *round-based* (RB-RSQ-g) and *step-based* (SB-RSQ-g). Additional details on these schemes and their implementations are provided in the Methods section.

III. METHODS

A. Faster-DiSH Circuit Design

We designed the HW emulator circuit using SystemVerilog. A schematic of our HW framework that includes the circuitry representing elements and connections of the cellular network, and the circuitry that conducts network simulation, is shown in Fig. 1. The simulation schemes are implemented and selected within the **`Rule Selector`** block, and each simulation is run for a specified number of steps or rounds. In addition, the network can be initialized with different element values at the beginning of each simulation.

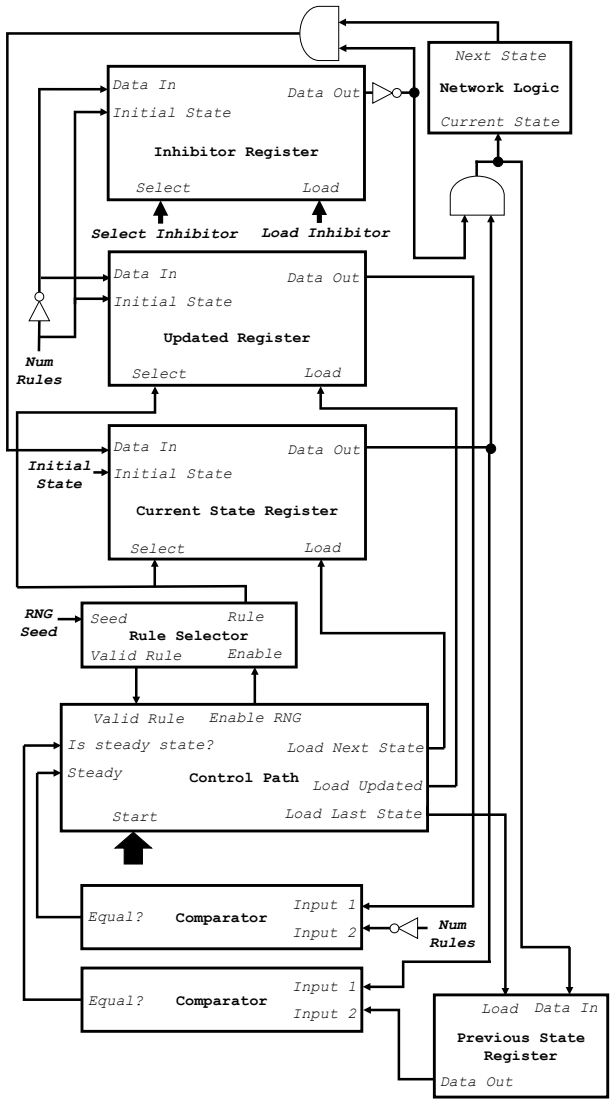

Figure 1. Hardware framework for simulating cellular networks with different simulation schemes and random update index generation.

A `Start` signal to the **`Control Path`** module prompts the **`Rule Selector`** to randomly generate an index via the `Enable` pin. If the `Valid Rule` pin indicates a valid index, **`Control Path`** asserts a high `Load` signal to the **`Inhibitor`**, **`Updated`**, and **`Current State Registers`**, allowing their values to be modified on the next clock cycle. The index is used as a `Select` input for the **`Current State Register`**, which keeps track of the state of each element in the model. The output (`Data Out`) of the **`Current State Register`** is then bit masked with output (`Data Out`) from the **`Inhibitor Register`**, where the inhibitor is chosen by the initial conditions (***`Select Inhibitor`***). The result of the combined current state and inhibition is stored in the **`Previous State Register`** and passed into the **`Network Logic`** module, which contains the update rules of the model. The current state is passed on the `Current State` pin, and the **`Network Logic`** module outputs the next state of the system on the `Next State` pin. The result is stored in the **`Current State Register`**. The circuit is "steady" once all rules have been run, as tracked by the **`Updated Register`**. The upper **`Comparator`** module compares the updated elements/groups with the complete list of elements/groups, asserting the `Steady` pin on the **`Control Path`** if they are equal. The circuit is said to be in "steady state" if the current state is equal to the previous state. The lower **`Comparator`** module compares the output of the **`Current State Register`** to that of the **`Previous State Register`** and asserts the `Is steady state?` pin on the **`Control Path`** if they are equal.

B. RNG Algorithms

While the SMLN scheme is deterministic, all other schemes require random number generation (RNG) to generate a random rule index that selects elements/groups and executes their update rules in each simulation round/step. We describe here two novel algorithms for HW-based RNG that we created and implemented within the **`Rule Selector`** block in Fig. 1.

1) Round-based RNG:

In the round-based simulation schemes, each element/group is updated at least once within a given round; therefore, each round consists of a number of steps equal to the number of elements/groups in the model, which have associated update rules. Our round-based RNG algorithm allows linear runtime with respect to the number of elements/groups in the model by eliminating the possibility of duplicate or invalid rule index generation. The operation is illustrated in Fig. 2. Two parallel register arrays, A and B, are utilized as stacks for storing items consisting of a rule index ("Priority") and randomly generated number ("Value"). Stack A always starts empty and grows upward while Stack B starts out full and recedes downward. With each step, a new item is pushed up Stack A. If the new item's Value is greater than an existing Value, the Priority of the new item is increased by one. Otherwise, the Priority of the existing item is increased by one. If there are no other Values on the stack, the rule takes a default priority of 0. Simultaneously to the generation of Stack A, an item is popped off of B, and its Priority determines which element/group is updated in the current step. When a round is

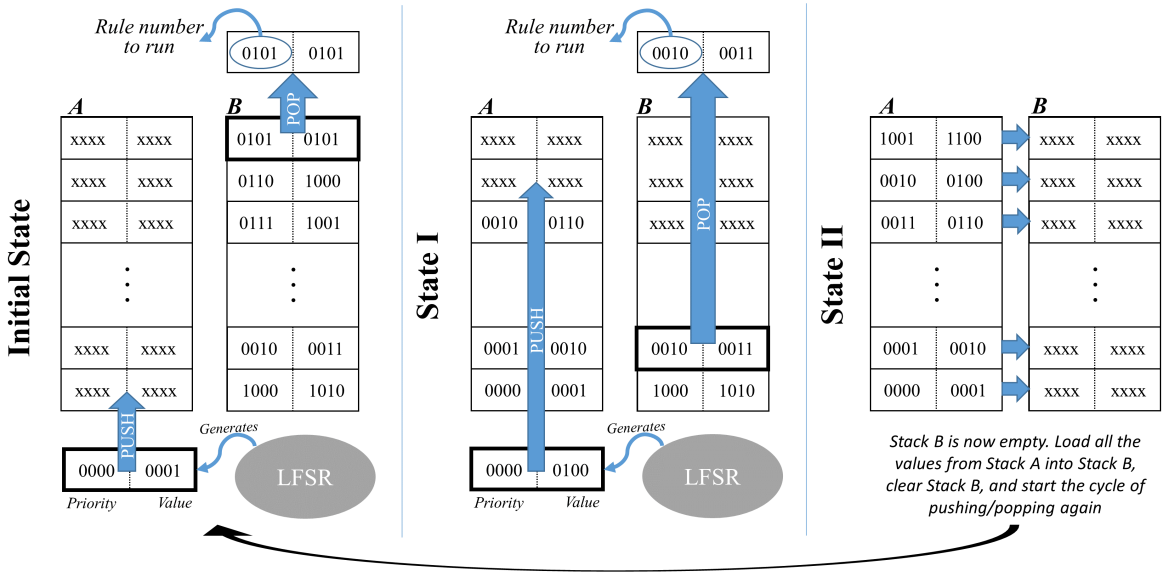

Figure 2. The RB-RSQ rule selection scheme implementation for a sample 16-rule system, utilizing a linear feedback shift register (LFSR). A 16-element/group system as outlined here would be $2 \cdot \log_2 16 = 8$ bits wide and 16 registers tall. Each register in the array could be thought of as being logically divided in half, with the least significant bits corresponding to a Value, and the most significant bits indicating the Priority.

completed, Stack B is empty, and Stack A is full. The items from A are then loaded into B, Stack A is flushed, and the parallel generation and popping of items repeats. At the start of the simulation, where both stacks are empty, items must be generated to fill Stack B before a round can begin. After the initial generation of Stack B, the coincident generation and popping of items allows linear runtime without duplicates or invalid rules.

2) Step-based RNG

We also developed an RNG algorithm for the step-based simulation scheme where, in each step, one element/group is randomly selected for update. Therefore, the same element/group can potentially be updated multiple times in a row. Directly using the numbers generated with the RNG as rule indices is highly inefficient for a model that has a number of elements/groups not equal to a power of two: a model with 37 elements would need 6 bits to represent each of the potential element updates, leading to $2^6 - 37 = 27$ outputs (42%) counting as a costly "miss". Here, we determine the rule index (I) using Equation 1, where X is the RNG number and E is the number of elements/groups. We also use 10 RNG bits ($n = 10$) to approach uniform probability of generating any index, calculated by setting a threshold of $(2^n \bmod E)/2^n < 3\%$

$$I = X \bmod E \qquad (1)$$

TABLE I. SIMULATION SCENARIOS AND INITIAL VALUES

Scenario	TCR_high	TCR_low	TGFbeta	AKT off	Toggle
1	1	0	0	0	-
2	0	1	0	0	-
3	1	0	1	0	-
4	0	1	1	0	-
5	1	0	0	1	-
6	1^a	0	0	0	20.00%
7	1^a	0	0	0	26.67%
8	1^a	0	0	0	33.33%

a. Initial value before toggle

TABLE II. RMSE AVERAGED (%) ACROSS ALL SIMULATION SCENARIOS

Element	Scheme				
	RB-RSQ	SB-RSQ	RB-RSQ-g	SB-RSQ-g	SMLN
FOXP3	1.13	11.18	10.86	6.22	14.99
IL2	0.99	10.79	13.00	5.41	6.23

C. HW and SW Comparison

To compare the simulators written in SW, and those implemented directly in HW circuit, we studied eight scenarios representing different input value configurations. To evaluate the accuracy of the HW framework, we compared the responses of proteins Foxp3 and IL2 (used as markers of Treg and Th cell phenotypes, respectively [5]). Table I shows the initial values for input signals that were varied for the eight scenarios, antigen dose affecting T cell receptor (TCR) signal strength, the value of transforming growth factor beta ligand (TGFβ), and the value of inhibitor of protein kinase B (AKT off). The percentages listed for the Toggle scenarios in Table I represent the percentage of simulation steps/rounds that were completed before the protein's value was toggled. Of the variables not listed in Table I, CD28, PTEN, TSC, CD122, and CD132 were initialized to 1, and all the other variables were initialized to 0, to mimic the naïve T cell phenotype at the beginning of each simulation. We ran all round-based simulations for 30 rounds, and all step-based and SMLN simulations for 2000 steps [7]. We ran each simulation scheme 200 times from the initial to steady state and calculated average trajectories, according to the methodology from [7].

IV. RESULTS

Fig. 3 and Fig. 4 show the difference between average trajectories obtained with our HW framework and the SW

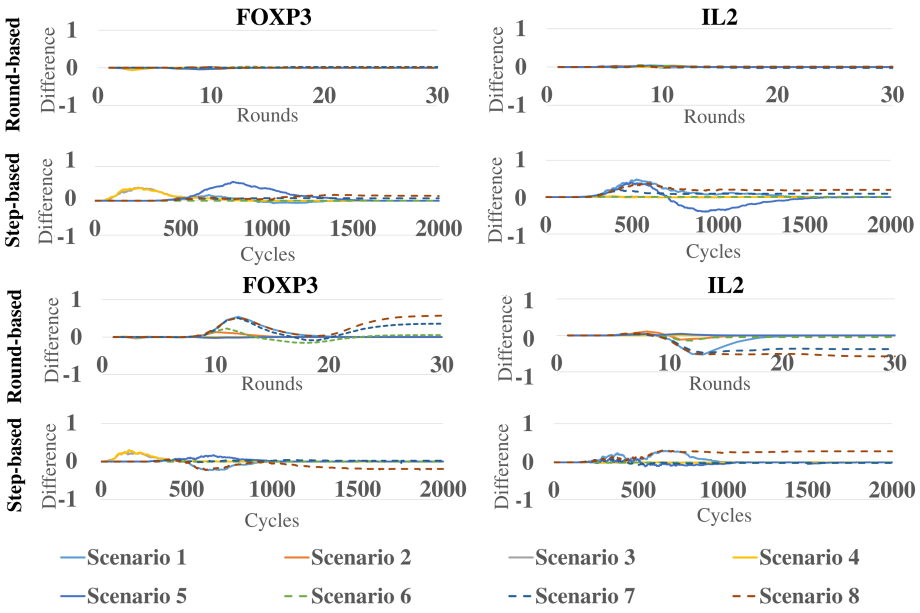

Figure 3. Difference between HW and SW simulator output for FOXP3 and IL2 with the RSQ scheme (top), and the RSQ-g scheme (bottom).

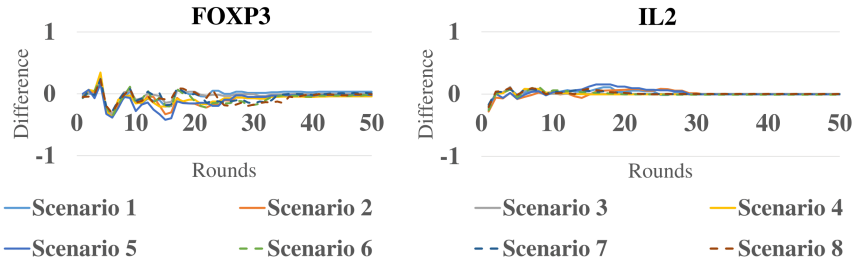

Figure 4. Difference between HW and SW simulator output for FOXP3 and IL2 with the SMLN scheme.

simulator for Foxp3 and IL2 in each simulation scheme. Average root-mean-square error (RMSE) across all scenarios for five trials of each simulation scheme is shown in Table II.

Runtime comparisons are shown in Table III for each simulation scheme. HW wall-clock runtimes were calculated by executing the framework in ModelSim [12] for each scheme and dividing the reported number of simulated clock cycles by 50 MHz, a common and conservative FPGA base clock frequency. This result could then be directly compared to SW runtime as performed on a 2015 Apple MacBook Pro (3.1 GHz Intel i7 processor). The HW circuit design had a median speedup of 54.4X for round-based simulations, and 426.7X for step-based simulations. The greatest speedup was provided for the SMLN scheme, followed by RSQ-g and then RSQ.

TABLE III. RUNTIME (MS) COMPARISON FOR EACH SIMULATION SCHEME.

	Steps/Rounds	SW	HW	Speedup
Round-Based Schemes				
RB-RSQ	1929500	1062.5	38.6	27.5X
RB-RSQ-g	1216500	1253.6	24.3	51.5X
SMLN	11300	645.8	0.2	2857.5X
RB-RSQ Toggle	1929500	1027.6	38.6	26.6X
RB-RSQ-g Toggle	1216500	1394.7	24.3	57.3X
SMLN Toggle	11300	688.9	0.2	3048.3X
Step-Based Schemes				
SB-RSQ	401300	3225.1	8.0	401.8X
SB-RSQ-g	303537	2745.0	6.1	452.2X
SB-RSQ Toggle	401300	3036.9	8.0	378.3X
SB-RSQ-g Toggle	303537	2741.7	6.1	451.6X

V. DISCUSSION

Our results show that all five simulation schemes, when written directly into a HW circuit produced results comparable to the SW code, with all RMSEs under 15%. The round-based implementation had the smallest RMSE, but with the smallest runtime speedup, making it a Las Vegas type algorithm. Conversely, the SMLN scheme showed the greatest runtime speedup but greatest RMSE, making it a Monte Carlo type algorithm. The SB-RSQ-g implementation had the second smallest RMSE and the fastest runtime speedup for all the RSQ simulation schemes. The step-based HW implementation was expected to provide greater speedup than the round-based implementation because the round-based RNG relies on pushing numbers onto a stack at linear runtime. Similarly, the SMLN schemes show much greater speedup than the other

schemes, because every element's value is updated in each step without the need for RNG. However, the RSQ schemes are more desirable for modeling stochasticity in biological networks [1], [2], [3], [4]. The speedups for RSQ simulation schemes were similar to previous stochastic simulation work by Yoshimi et al. [2], and HW runtimes were similar to those reported in [4], despite differences in HW design architectures.

All non-Toggle HW simulations approached zero difference from the SW simulator at steady state. For the Toggle scenarios, a large deviation occurred subsequent to the toggle of the antigen dose, and the steady state values were different between the SW simulation and the HW framework, particularly for the grouped schemes. This is likely due to the fact that inputs in the logic circuit of HW framework change before the output has stabilized.

The most likely source of differences between HW and SW simulation with RSQ schemes lies in the randomness of the HW RNG and the randomized SW update rule selection. This is evidenced by the greater differences in transient behavior compared to steady state. Comparing the HW emulator design with a SW simulator using the same sequence of element/group updates could provide more similar transient results, and will be investigated in future work.

The SMLN scheme implementations showed differences at the start of the simulation, but reached zero difference at steady state. In the SMLN scheme, each element not governed by a forced initial condition was given a random initial state, leading to 252 possible initial states. Since only 200 of these initial states were tested, and the 200 states used in the HW model most likely differed from the 200 states used in the SW model, deviations are expected especially at the start of the simulation due to different initial conditions. In other words, whenever the initial state of the model (initial values of all model elements) is exactly the same in HW and SW simulations, the results of these two implementations for the SMLN scheme will be the same as well.

VI. Conclusion

In this work, we propose a design for HW emulation of cellular signaling networks. The design of our HW framework, Faster-DiSH, includes five simulation schemes and two random update index generation techniques. Our design has been shown to accurately predict the phenotype decision in T cells while providing orders of magnitude runtime speedup compared to SW simulation. Specifically, in the case of the T cell differentiation control network, the round-based implementation was most accurate when compared to its SW counterpart, however, it provided the smallest runtime speedup. On the other hand, the SMLN implementation led to the largest speedup although with lower accuracy, when compared to SW. Future work includes FPGA synthesis to evaluate resource utilization, parallelization and simulation of multiple model versions, as well as improving the random update index generation to increase the accuracy without the cost of runtime.

Acknowledgment

This work is supported in part by DARPA Big Mechanism award W911NF-17-1-0135 and DARPA World Modelers award W911NF-18-1-0017.

References

[1] L. Salwinski and D. Eisenberg, "In silico simulation of biological network dynamics," *Nature Biotechnology*, vol. 22, no. 8, pp. 1017–1019, 2004. [Online]. Available: http://dx.doi.org/10.1038/nbt991

[2] M. Yoshimi, Y. Iwaoka, Y. Nishikawa, T. Kojima, Y. Osana, A. Funahashi, N. Hiroi, Y. Shibata, N. Iwanaga, and H. Yamada, "FPGA implementation of a data-driven stochastic biochemical simulator with the next reaction method," in *2007 International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2007, Conference Proceedings, pp. 254– 259.

[3] N. Miskov-Zivanov, A. Bresticker, D. Krishnaswamy, S. Venkatakrishnan, P. Kashinkunti, D. Marculescu, and J. R. Faeder, "Regulatory network analysis acceleration with reconfigurable hardware," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2011, Conference Proceedings, pp. 149–152.

[4] N. Miskov-Zivanov, A. Bresticker, D. Krishnaswamy, S. Venkatakrishnan, D. Marculescu, and J. R. Faeder, "Emulation of biological networks in reconfigurable hardware," in *Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine*. 2147893: ACM, 2011, Conference Proceedings, pp. 536–540.

[5] N. Miskov-Zivanov, M. S. Turner, L. P. Kane, P. A. Morel, and J. R. Faeder, "The duration of T cell stimulation is a critical determinant of cell fate and plasticity," *Science Signaling*, vol. 6, no. 300, p. ra97, 2013.

[6] I. Albert, J. Thakar, S. Li, R. Zhang, and R. Albert, "Boolean network simulations for life scientists," *Source Code for Biology and Medicine*, vol. 3, p. 16, 2008.

[7] K. Sayed, Y. H. Kuo, A. Kulkarni, and N. Miskov-Zivanov, "DiSH simulator: Capturing dynamics of cellular signaling with heterogeneous knowledge," *in 2017 Winter Simulation Conference (WSC),* Dec 2017, pp. 896–907.

[8] K. Sayed, C. A. Telmer, and N. Miskov-Zivanov, "Motif modeling for cell signaling networks," *in 8th Cairo International Biomedical Engineering Conference (CIBEC)*, 2016, Conference Proceedings, pp. 114–117.

[9] T. Hofer, O. Krichevsky, and G. Altan-Bonnet, "Competition for IL-2 between regulatory and effector T cells to chisel immune responses," *Frontiers in Immunology*, vol. 3, p. 268, 2012.

[10] S. Sakaguchi, K. Wing, and M. Miyara, "Regulatory t cells - a brief history and perspective," *European Journal of Immunology*, vol. 37 Suppl 1, pp. S116–S123, 2007.

[11] D. Thomas, "Logic Design and Verification using SystemVerilog (Revised)". *CreateSpace Independent Publishing Platform*, 2016.

[12] Mentor Graphics, "ModelSim® Reference Manual Software Version 10.1c" (2012) [online]. Available: https://www.cc.gatech.edu/~hadi/teaching/cs3220/doc/modelsim/ModelSim_Reference_Manual_v10.1c.pdf