33rd Annual International Conference of the IEEE EMBS
Boston, Massachusetts USA, August 30 - September 3, 2011

Regulatory Network Analysis Acceleration with Reconfigurable Hardware

Natasa Miskov-Zivanov, *Member*, *IEEE*, Andrew Bresticker, *Student Member*, *IEEE*, Deepa Krishnaswamy, SreesanVenkatakrishnan, Prashant Kashinkunti, Diana Marculescu, *Senior Member, IEEE*, and James R. Faeder

Abstract— **In medical research it is of great importance to be able to quickly obtain answers to inquiries about system response to different stimuli. Modeling the dynamics of biological regulatory networks is a promising approach to achieve this goal, but existing modeling approaches suffer from complexity issues and become inefficient with large networks. In order to improve the efficiency, we propose the implementation of models of regulatory networks in hardware, which allows for highly parallel simulation of these networks. We find that our FPGA implementation of an example model of peripheral naïve T cell differentiation provides five orders of magnitude speedup when compared to software simulation.**

I. Introduction

THE development of experimental methods and tools, together with the advances in computational power, has greatly improved the process of obtaining and collecting experimental data. However, we have now reached the point where the vast amount of data collected exceeds our capacity for analyzing it. At the same time, predicting the dynamics of complex molecular networks that control living organisms is still an important challenge of systems biology.

Over the past decade, a number of computational approaches have been proposed for the purpose of modeling and studying biological networks. The complexity of these models increases rapidly with the size of the network. Moreover, simulations of such models are computed sequentially on general purpose CPUs, which is in contrast to the highly parallel nature of information flow within biochemical networks. To this end, several hardware-oriented approaches to biological network simulation have been proposed recently [1][2][3]. They have all focused on the implementation of variants of Gillespie's stochastic simulation algorithm (SSA) [4]. The authors in [1][2] implemented the SSA as a single FPGA thread, resulting in speedups of about one order of magnitude compared to a software implementation. More recent work [3] implements multiple simulation threads to achieve greater efficiency. While these implementations do make SSA-based simulations more efficient, applications that use ODE [5][6], rule-based [7][8] and logical modeling [10][11][12] approaches do not benefit.

In this work, we propose a design methodology (shown in Fig. 1 and described in detail in Section II) for a reconfigurable hardware, that is, Field Programmable Gate Arrays (FPGAs). We present our methodology using an example model for peripheral naïve T cell differentiation into regulatory vs. helper T cells, that has been proved to play a critical role in interactions between tumors and immune system. We show that hardware-based emulation of regulatory network models can greatly improve our efficiency of simulating these models, and therefore, produce rapid answers to inquiries about system response to a number of stimuli, pathogens or drugs. The contributions of this work, when compared to other proposed hardware approaches to studying biological networks include:

- Hardware emulation of *dynamic, logical models* of regulatory networks;
- *FPGA design framework* that allows for implementing different logical models and different simulation scenarios;
- *Implementation of a top module* that allows for concurrent simulation of multiple implemented copies of the network;
- *Several orders of magnitude speedup* in network simulation when compared to software-based approaches.

II. Hardware Design Methodology

In this section, we describe the main steps of FPGA design and the specificities of designing for emulation of regulatory networks, using the peripheral naïve T cell

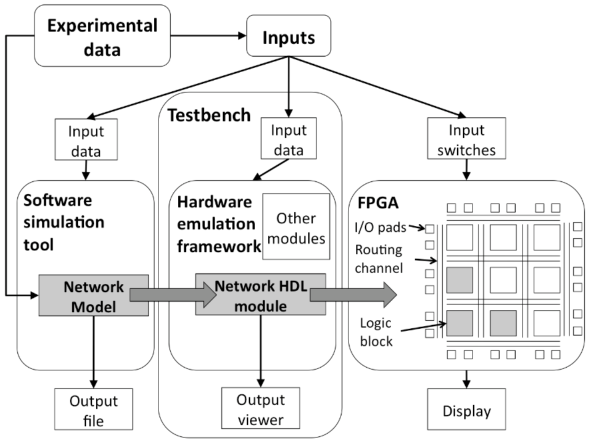
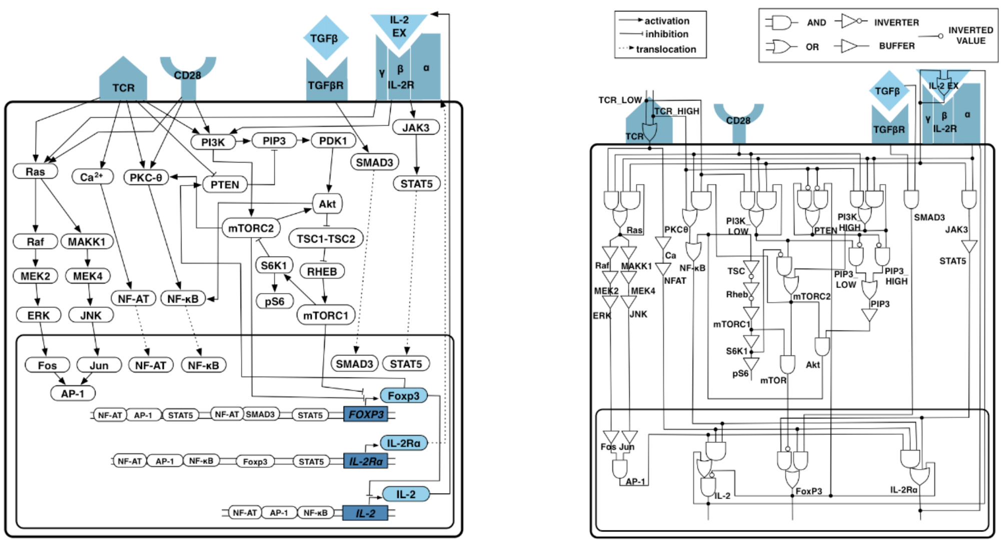


Fig. 1. Design flow: network model (logical, rule-based model), HDL model implementation and actual FPGA implementation.

Manuscript received April 15, 2011. This work was supported in part by NIH grant UL1-RR024153 and an NSF Expeditions in Computing grant (award ID 0926181).

N. Miskov-Zivanov and J. R. Faeder are with the Department of Computational and Systems Biology, School of Medicine, University of Pittsburgh, USA (phone: 412-648-9613; e-mail: nam66@pitt.edu, faeder@pitt.edu).

A. Bresticker, D. Krishnaswamy, S. Venkatakrishnan, P. Kashinkunti and D. Marculescu are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, USA (e-mail: abrestic@ece.cmu.edu, deepakri@ece.cmu.edu, sreesanv@ece.cmu.edu, pkashink@ece.cmu.edu, dianam@cmu.edu).



(a) (b)

Fig. 2. T-cell differentiation network adopted from [12]: (a) Molecular interaction map and (b) Gate-based logic implementing the network.

differentiation example.

The T cell model shown in Fig. 2 is adopted from [12], where the model has been determined through extensive literature survey and discussions with experts. In Fig. 2(a), we present the cell signaling network model with key elements and connections involved in differentiation. As it can be seen from the network, this model includes signaling from receptors (TCR, CD28, TGFβ, IL-2R), subsequent activation of transcription factors (AP-1, NF-AT, NFκB, STAT5, Smad3), gene expression (Foxp3, IL-2Rα, IL-2), as well as the effect of transcribed genes on receptor signaling (IL-2Rα, IL-2) and transcription (Foxp3). T cell subpopulation (regulatory, Treg, vs. helper, Th) ratios have been shown to play an important role in many immune and autoimmune pathologies, but the determinants of differentiation into these two phenotypes are not yet understood. It is known that a marker for Treg cells is Foxp3 and a marker for Th cells is IL-2. It has been suggested in [13] that most of the cells differentiate into Th phenotype for high antigen dose, while a significant population of Treg cells results from stimulation with low antigen dose. In Fig. 2(b), we also show the circuit model that we have developed in [12] and which we implemented in hardware. However, our hardware emulation methodology is general enough to allow implementation of any logical modeling of biological processes.

The steps of FPGA design are presented as follows. We also describe how our approach can be generalized for different models of regulatory networks.

A. Model definition

In order to design a circuit that can emulate a biological network, one needs to consider several information sources or 'inputs' to the design, as shown in Fig. 1. This includes existing experimental data or knowledge about network interactions. Next, it is also necessary to identify the type of a model to be implemented. As described in Section I, previous work focused on implementing the Gillespie's simulation algorithm for the system of differential equations. We present here the implementation of a dynamic, logical model, but do not restrict our approach to logical models only. We plan to create a general-purpose framework that can translate rule-based and reaction-based models into hardware implementation. These models are usually written following existing software-simulation-tool templates and can be simulated using these tools. Finally, once the model is identified, one needs to define a set of inputs and outputs of interest.

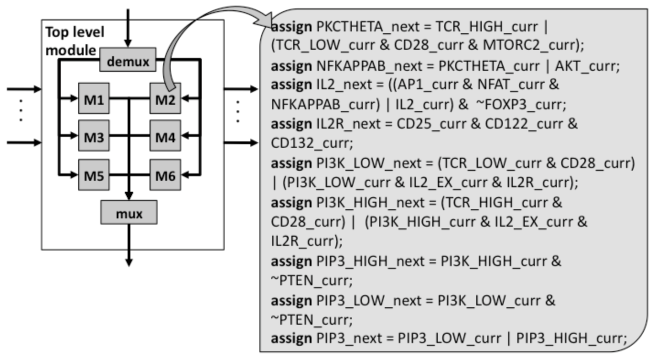
B. HDL framework

Once the model is defined, the next step is the implementation of the model in a hardware description language (HDL). In this work, we use Verilog HDL [14]. Any network model defined as a logical model can be translated into an HDL description in a straightforward way. We translated the T cell model manually, but in the future we anticipate developing an automatic Logical Model → Verilog translator.

The framework that we developed in Verilog consists of several modules necessary to control the simulation of the network. These modules define the simulation setup (*e.g.*, number of rounds of simulation, deterministic vs. stochastic simulation, *etc*.). In this work, we use an asynchronous scheme for simulating the network following one used by the BooleanNet tool [10]. Each simulation round consists of applying the rules for the update of each element in a random order. The order matters because once an element is updated, the new value is used by subsequent update rules.

The HDL framework includes the top-level module, which can be used to run several network copies in parallel. Fig. 3 shows how modules are connected within the top module (left), as well as part of the Verilog code for a single module (right).

The Verilog description of the logical model relies on



```
assign PKCTHETA_next = TCR_HIGH_curr |
(TCR_LOW_curr & CD28_curr & MTORC2_curr);
assign NFKAPPAB_next = PKCTHETA_curr | AKT_curr;
assign IL2_next = ((AP1_curr & NFAT_curr &
NFKAPPAB_curr) | IL2_curr) &  ~FOXP3_curr;
assign IL2R_next = CD25_curr & CD122_curr &
CD132_curr;
assign PI3K_LOW_next = (TCR_LOW_curr & CD28_curr)
| (PI3K_LOW_curr & IL2_EX_curr & IL2R_curr);
assign PI3K_HIGH_next = (TCR_HIGH_curr &
CD28_curr) |  (PI3K_HIGH_curr & IL2_EX_curr &
IL2R_curr);
assign PIP3_HIGH_next = PI3K_HIGH_curr &
~PTEN_curr;
assign PIP3_LOW_next = PI3K_LOW_curr &
~PTEN_curr;
assign PIP3_next = PIP3_LOW_curr | PIP3_HIGH_curr;
```

Fig. 3. Implementation of a top-level control module: figure of the top level module connecting network modules (left) and logical model implementation in Verilog (right).

'assign' statements. This is the simplest way to implement the network although statements such as 'if-else' and 'case' also exist in Verilog and can be suitable for other model implementations. If several copies of the model are instantiated and downloaded on a single FPGA board (modules 'M1'-'M6' in Fig. 3), the 'top′ module can be used to connect wires from input switches (through a demultiplexer) to individual modules, as well as to connect outputs from individual modules to the display (through a multiplexer).

Inputs to the framework are modeled as input wires connected to the 'top' and 'control' modules and include: *Initial state vector*, which initializes all elements of the network to a defined value; *Element inhibitors* and their time of addition, which define if there are elements to be turned off at some point during simulation; *Time of stimulation removal*, which defines if the stimulation that allows for activation of the network is removed during simulation; *Gene knock-out* information, which defines whether there are copies of the model in which some genes are permanently turned off.

Outputs of the framework include signals representing either several elements that need to be measured, or all elements of the model. Tools for the simulation of designs written in HDL exist, thereby also allowing for viewing model outputs, that is, their behavior from the beginning to the end of simulation. More details about using the tools and all steps of specifying HDL design are provided in [15].

Finally, the written HDL implementation can be simulated and tested using a higher-level module, 'Testbench' (as shown in Fig. 1), which encloses the overall design.

C. *FPGA implementation*

When the model is implemented and tested, it can be downloaded onto an FPGA board. An FPGA consists of: 'logic blocks', which can be configured to perform complex combinational functions, or simple logic gates like AND and XOR, a hierarchy of 'reconfigurable interconnects' that allow the blocks to be "wired together" and 'I/O pads', that are connected to switches, buttons and displays.

In this work, buttons are used to start and stop the simulation, while switches are used to select the activator or inhibitor. As an initial way of displaying results, the state of the network is displayed (in hexadecimal digits) on a bank of seven-segment light-emitting diode (LED) displays. Switches are used to select which network module results to display, and which outputs of the chosen network to display. An alternative method would be to use a block RAM (Random Access Memory) to store the values and display them on the monitor; we plan to explore this direction in the future.

III. RESULTS

We compared the runtime and results that were obtained from simulating our Verilog HDL framework (identical to those obtained from the displays on the FPGA board) with results obtained from software simulations in BooleanNet tool, which is implemented in Python [10]. Not only does the hardware emulation accurately reproduce the behavior of all elements of the network, but it is also orders of magnitude faster.

In case of our T cell differentiation example model, we compute the speedup as follows. For each scenario, software simulations were run 200 times, which has been shown to be sufficient to estimate average values across the runs and account for the stochasticity in biological processes. 20 update rounds are executed in each simulation run, where a round includes a single update of each element in the model (in our case, this is 54 update steps within the round). We simulated the network for 20 rounds only, due to the fact that the network reaches steady state within these 20 rounds in all studied scenarios. Simulations are run on an FPGA that has a speed of 1GHz (clock period of 1ns), with a single update step taking approximately five clock cycles. This leads to a single network module evaluation runtime on FPGA of:

$$5 \times 54 \times 20 \times 200 \times 10^{-9} = 1\text{ms}$$

Since an identical simulation setup takes about 60s to run using BooleanNet, a 60,000X speedup can be obtained using our FPGA implementation. Furthermore, we were able to implement six instances of the network on a Spartan 3 FPGA board simultaneously and to execute them in parallel by implementing the top-level module as shown in Fig. 3. This provides an approximate linear speedup increase of 6 × 60,000 or more than five orders of magnitude over BooleanNet.

In Fig. 4, we present the comparison of several software simulation and hardware emulation results. In Fig. 4(a)-(b), we present the Foxp3 and IL-2 results for two different scenarios: stimulation with high and low antigen dose. As it can be seen from those figures, the proposed method is 100% accurate when determining steady-state values. In Fig. 4(c)-(d) we use several additional scenarios for comparison. Behavior of the system with addition of Akt and mTOR inhibitors has been shown in Fig. 4(c), where lines represent software simulation results and error lines show the difference between those results and hardware emulation

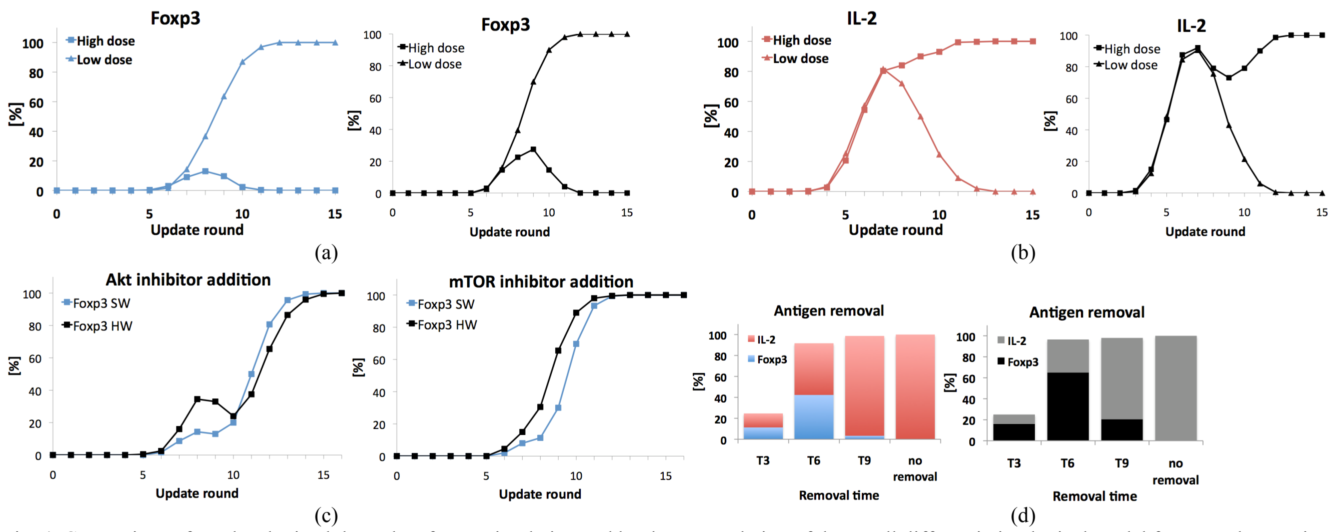


Fig. 4. Comparison of results obtained through software simulation and hardware emulation of the T cell differentiation logical model for several scenarios and initial conditions: Results for high and low antigen dose, simulation (left) and emulation (right) for Foxp3 (a) and IL-2 (b); (c) Foxp3 results (SW – simulation, HW – emulation) for high antigen dose with addition of inhibitors, Akt (left) and mTOR (right); (d) Foxp3 and IL-2 steady state results from simulation (left) and emulation (right) when antigen is removed at round 3,6 or 9 and without antigen removal.

results. Finally, we also implemented the removal of stimuli that occurs at different times during simulation. Fig. 4(d) presents the comparison of steady-state values between software and hardware in that case.

As it can be seen from Fig. 4, some discrepancies are observed when comparing hardware emulation results with software simulations. We do not believe that these differences will have any effect on the timing results, which is supported by their small magnitude, but we need to understand their origin in order to have confidence that the implementation is correct. The two main sources of possible error are noise arising from the relatively small sample sizes used in our comparisons (200) or a subtle difference in execution semantics arising in our FPGA simulations. To rule out the former, we simply need to run a larger number of trajectories and see if the differences disappear. More sophisticated statistical tests can also be applied [16]. If the differences persist and assuming there are no errors in the logic encoded with variable assignments, we will test for differences in execution semantics for fixed (instead of random) update orders.

IV. Conclusion

Modeling of biological regulatory networks is necessary for tackling medical and biological challenges. Moreover, being able to obtain fast answers to inquiries about system's response to different stimuli is of critical importance. We propose an FPGA design methodology that provides an efficient framework for emulating dynamic models of biological networks. Our results show that such a framework can accurately reproduce all results obtained through software simulation, but doing so orders of magnitude faster. Our future work will focus on generalizing the design to allow for emulation of different types of models and the improvement of displaying results to users.

References

[1] J. F. Keane, *et al.*, "A compiled accelerator for biological cell signaling simulations," in *Proc. of ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA)*, 2004.

[2] L. Salwinski and D. Eisenberg, "*In silico* simulation of biological network dynamics," in *Nature Biotechnology*, vol. 22, pp. 1017-1019, 2004.

[3] M. Yoshiini, et al., "FPGA Implementation of a Data-Driven Stochastic Biochemical Simulator with the Next Reaction Method," in *Proc. of International Conference on Field Programmable Logic and Applications (FPL)*, pp. 254-259, 2007.

[4] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," in *Journal of Physical Chemistry*, vol. 81, pp. 2340-2361, 1977.

[5] J. J. Tyson, et al., "The dynamics of cell cycle regulation," in *Bioessays*, vol. 24, pp. 1095-1109, 2002.

[6] M. Novak and J. J. Tyson, "A model for restriction point control of the mammalian cell cycle," in *Journal of Theoretical Biology*, vol. 230, pp. 563-579, 2004.

[7] M. L. Blinov, et al., "BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains," in *Bioinformatics*, vol. 20, pp. 3289-91, 2004.

[8] J. R. Faeder, et al., "Rule-based modeling of biochemical systems with BioNetGen," in *Methods Mol Biol*, vol. 500, pp. 113-67, 2009.

[9] S. Bornholdt, "Boolean network models of cellular regulation: prospects and limitations," in *J R Soc Interface*, vol. 5 Suppl 1, pp. S85-94, 2008.

[10] I. Albert, et al., "Boolean network simulations for life scientists," in *Source Code Biol Med*, vol. 3, p. 16, 2008.

[11] M. Chaves, et al., "Robustness and fragility of Boolean models for genetic regulatory networks," in *Journal of Theoretical Biology*, vol. 235, pp. 431-49, 2005.

[12] N. Miskov-Zivanov et al., "Boolean Modeling and Analysis of Peripheral T Cell Differentiation," under submission.

[13] M. S. Turner, et al., "Dominant role of antigen dose in CD4+Foxp3+ regulatory T cell induction and expansion," in *Journal of Immunology*, vol. 183, pp. 4895-903, 2009.

[14] D. E. Thomas and P. R. Moorby, *The Verilog hardware description language*, 5th ed. Norwell, Mass.: Kluwer Academic Publishers, 2002.

[15] N. Miskov-Zivanov, et al., "Emulation of Biological Networks in Reconfigurable Hardware," in *Proc. of ACM Conference on Bioinformatics, Computational Biology and Biomedicine (ACM-BCB)*, August 2011, to appear.

[16] L. A. Harris and P. Clancy, "A "partitioned leaping" approach for multiscale modeling of chemical reaction dynamics," in Journal of Chemical Physics ,vol. 125, p. 144107, October 2006.