Stochastic Hazard Analysis of Genetic Circuits in iBioSim and STAMINA

Lukas Buecherl,[*,†] Riley Roberts,[‡] Pedro Fontanarrosa,[¶] Payton J Thomas,[¶] Jeanet Mante,[§] Zhen Zhang,[‡] and Chris J. Myers[*,‖]

†*Department of Biomedical Engineering, University of Colorado Boulder, Boulder, Colorado 80309, USA*

‡*Department of Electrical and Computer Engineering, Utah State University, Logan, Utah 84322, USA*

¶*Department of Biomedical Engineering, University of Utah, Salt Lake City, Utah 84112, USA*

§*Department of Biomedical Engineering, University of Colorado Boulder, Colorado 80309, USA*

‖*Department of Electrical, Computer, and Energy Engineering, University of Colorado Boulder Boulder, Colorado 80309, USA*

E-mail: lukas.buecherl@colorado.edu; chris.myers@colorado.edu

Abstract

In synthetic biology, combinational circuits are used to program cells for various new applications like biosensors, drug delivery systems, and biofuels. Similar to asynchronous electronic circuits, some combinational genetic circuits may show unwanted switching variations (*glitches*) caused by multiple input changes. Depending on the biological circuit, glitches can cause irreversible effects and jeopardize the circuit's functionality. This paper presents a stochastic analysis to predict glitch propensities for

three implementations of a genetic circuit with known glitching behavior. The analysis uses *STochastic Approximate Model-checker for INfinite-state Analysis* (STAMINA), a tool for stochastic verification. The STAMINA results were validated by comparison to stochastic simulation in iBioSim resulting in further improvements of STAMINA. This paper demonstrates that stochastic verification can be utilized by genetic designers to evaluate design choices and input restrictions to achieve a desired reliability of operation.

Keywords

Genetic Regulatory Networks, Genetic Circuits, Hazards and Glitches, Synthetic Biology, Model Verification, Stochastic Simulation

In recent years, *digital logic theory* has been applied to DNA to program living cells as part of the research field of synthetic biology. Digital logic theory is a common and well-established electrical engineering field that is fundamental to the design of digital electronic systems, such as computers, smartphones, etc. Digital logic circuits are based on each signal wire being in one of two *binary* states: a true/high state or a false/low state. A digital circuit calculates the states of its outputs based on the states of its inputs. It is built of different logic gates that perform different mathematical logic operations. For example, an *AND gate* only shows a high output if both of the gate's input signals are high. An *OR gate*, however, shows a high output when either of its inputs is high. Examples of encoded functions in synthetic biology include a genetic toggle switch,[1] a repressilator,[2] and other functions, e.g., those presented by Nielsen et al.[3]

The order of operation of digital logic circuits can be performed either *synchronously* (i.e., using a clock signal to order events) or *asynchronously* (i.e, each event completion triggers the next event). Genetic circuits have generally followed the asynchronous design paradigm, since generating a precise biological timing reference needed by synchronous operation would be quite difficult.[4] Additionally, like their asynchronous electronic counterparts, some input

changes can cause unwanted switching variations (*glitches*).[5] A glitch is a transient, unexpected output from the circuit, for example, a circuit briefly expresses a high output when it is supposed to remain in a stable low output state. If an input transition has the possibility of causing a glitch, the transition is said to have a *hazard*.[6,7] These unwanted variations can have drastic effects if the output produces irreversible effects, e.g., apoptosis or an early drug release in an off-target tissue.

In 2016, Nielsen et al. presented Cello,[3] a *genetic design automation* (GDA) tool that applies principles from *electronic design automation* (EDA) to genetic circuit design to accelerate and simplify the genetic design process. The user specifies the desired circuit function using the Verilog *hardware description language* (HDL), which Cello automatically encodes into a DNA sequence. Cello was used to automate the design of 60 combinational genetic circuits that were tested in *Escherichia coli*. The generated 0x8E circuit showed a glitching behavior in the laboratory under a specific set of input transitions.

Fontanarrosa et al.[5] investigated the cause of this glitching behavior. In this work, the authors demonstrated that an in-depth analysis of hazards is necessary to guide the design towards robust genetic circuits.[5] Using dynamic *ordinary differential equation* (ODE) models, this work identified input transitions that result in glitching behavior for a genetic circuit. Additionally, they presented two modified implementations of the circuit's function to reduce the magnitude of the circuit's typical glitching behavior. Although genetic circuits share similarities with their asynchronous electrical counterparts, they also display unique properties. The behavior of genetic circuits is highly unpredictable compared to the binary behavior of digital circuits. The low molecule counts of genetic circuits lead to stochastic and noisy behavior.[8–12] This stochasticity means that ODE analysis, such as that used in,[5] cannot predict the likelihood of glitching behavior, but rather whether or not a glitch occurs in a typical response of the circuit. Therefore, stochastic analysis is more suitable for predicting the robustness of genetic circuits. The likelihood of a glitch is crucial information that can help designers not only think of the states for each input combination, but also specify

the desired sequence of input and output changes of the circuit, or which input transitions must be avoided. Finally, this analysis can help designers understand what are the risks of applying certain input changes, and decide whether the probability is critical or not for the intended purpose of their designed system.

This paper builds upon this work by performing a stochastic analysis of these circuits to evaluate the robustness of these designs. These analyses include both stochastic simulations performed by the iBioSim GDA tool[13] and stochastic model checking using the STAMINA model checker.[14] These results indicate that stochastic simulation and stochastic model checking are suitable to determine the likelihood of glitches. Although stochastic simulation can return results quickly, stochastic model verification returns a probability range that includes the true probability, and has the potential to analyze more difficult properties.

Results and Discussion

Background. As mentioned in the introduction, genetic circuits follow the asynchronous design paradigm and share their glitching behavior with their electronic counterparts. A digital circuit's behavior is defined by a Boolean function, which can be represented as a truth table. Figure 1(a) shows the truth table of the circuit 0x8E originally presented by Nielsen et al..[3] The circuit reacts to the presence of the three inducer molecules *Arabinose* (Ara, ChEBI = 17535), *Isopropyl-beta-D-thiogalactopyranoside* (IPTG, ChEBI=61448) and *Acetylcholine* (aTc, ChEBI=15355) with four of the eight input combinations producing a high output indicated by the production of *yellow fluorescent protein* (*YFP*). For example, if only *Ara* and *IPTG* are present, as shown in the second to the last row of the truth table, the circuit will produce a high output indicated by *YFP* production, whereas if all three inducers are present no *YFP* will be produced.

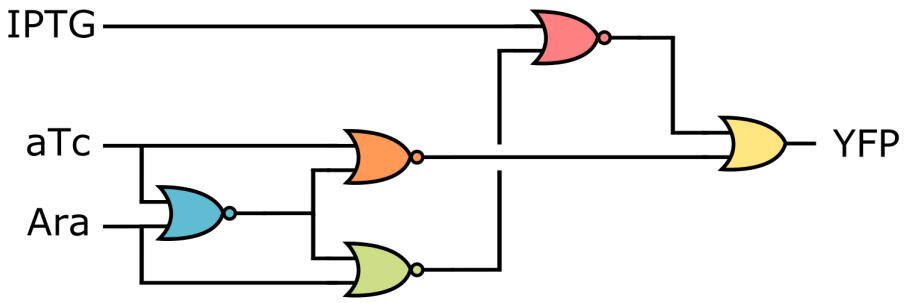
Figure 1(b) shows a Karnaugh map for the circuit 0x8E, a different way to visualize a circuit function that helps identifying hazards. The rows of the table indicate the presence of

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

(a)

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

(b)



(c)

Figure 1: (a) The function of the circuit 0x8E published in[3] shown as a truth table. The output *YFP* is matched to the different inputs of the inducers *Ara*, *IPTG* and *aTc*. A 0 represents a low input/output and a 1 represents a high input/output. (b) Another visualization of the function of circuit 0x8E as a Karnaugh map. The rows indicate the presence of *Ara*, and the columns indicate the presence of *IPTG* and *aTc*, respectively, with the entries indicating a high (1) or low (0) output. (c) The logic of the original circuit 0x8E as published in.[3] The OR gate is represented by  and the NOR gate by  . The three inducer molecules are *IPTG*, *aTc* and *Ara* and the output is *YFP*.

Ara, and the columns the presence of *IPTG* and *aTc*, respectively, with the entries indicating a high (1) or low (0) output. For example, in the upper left corner, if no inducer is present, the circuit produces a high output shown as a one, the same state shown in the first row of the table in Figure 1(a).

The red and blue arrows in Figure 1(b) show the different paths that the system can take when moving from the input transition *Ara*, *IPTG*, $aTc = (0, 0, 0)$ to $(1, 0, 1)$. In the case shown by the red arrow, the system detects the change of concentration of the inducer *Ara* first, transitioning momentarily to the state *Ara*, *IPTG*, $aTc = (1, 0, 0)$ before detecting the change of concentration of *aTc* and ultimately moving to the final state *Ara*, *IPTG*, $aTc = (1, 0, 1)$. Since the transition state *Ara*, *IPTG*, $aTc = (1, 0, 0)$ also produces a high output, the output stays high during the transition. During the transition shown by the blue arrow, the system detects the change of *aTc* before the change of *Ara*. In this case, the system briefly transitions to the state *Ara*, *IPTG*, $aTc = (0, 0, 1)$ before reaching the final state *Ara*, *IPTG*, $aTc = (1, 0, 1)$. However, the state *Ara*, *IPTG*, $aTc = (0, 0, 1)$ produces a low output. If the system transitions in this way, the output briefly drops from a high output to a low output before returning back to high when reaching the final state. Therefore, this input transition has a hazard. The likelihood of the hazard is determined by the likelihood of the system taking the different paths. This hazard is called a *static 1-hazard*, since the output should remain stable high throughout the transition. On the other hand, a hazard where the output should remain low but briefly glitches high is called a *static 0-hazard*. In their experiments on this circuit, Nielsen et al. detected the circuits glitching behavior.[3]

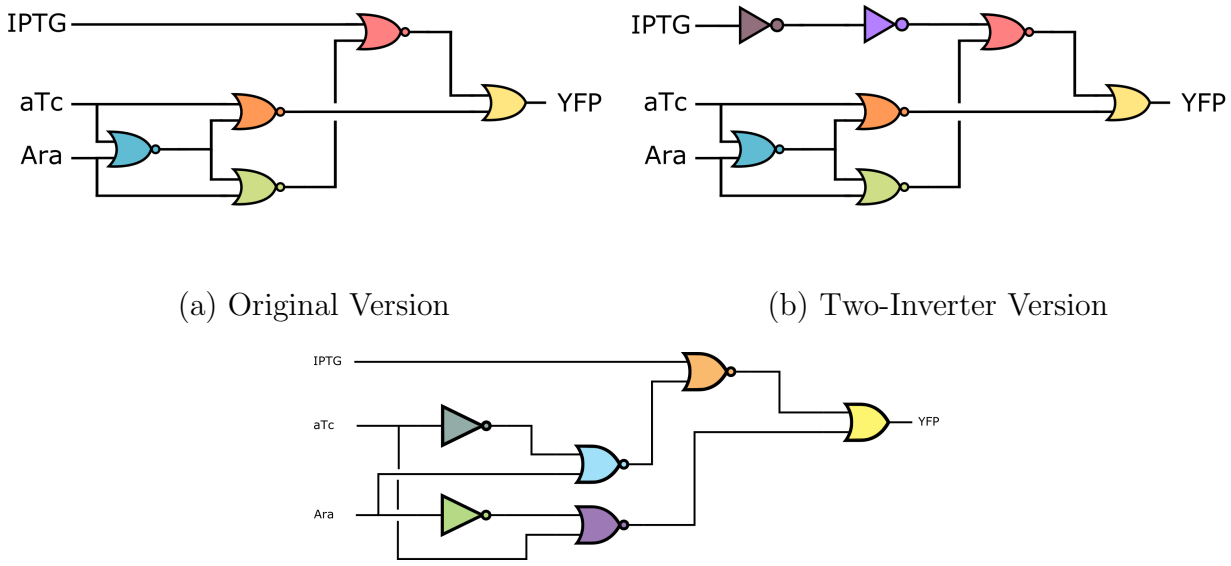
Finally, Figure 1(c) shows the logic or layout of the circuit as originally implemented by Nielsen et al..[3] A circuit function is implemented using a combination of different logic gates, called the logic implementation of the circuit. As shown later in the paper, a function can be constructed in multiple ways, using alternative logic implementations composed of different gate combinations. A hazard can be due to the function of a circuit, called a *function hazard*, or due to its logic implementation, called a *logic hazard*. Function hazards are a property of

the circuit's mathematical function and, therefore, unavoidable. Logic hazards, however, can be avoided by redesigning the circuit's logic implementation using *hazard-free* logic design methods.[7]

Results. The analysis in this paper is run on three implementations of one of the Cello circuits published in[3] labeled 0x8E. Figure 2(a) shows the original implementation of the circuit.[3] Figure 2(b) and (c) show two additional implementations of the circuit presented in.[5] All three circuits implement the same logic function represented by the truth table shown in Figure 1(a), but use different networks of logic gates. The circuit shown in Figure 2(b) has two added NOT gates that add an extra delay to the IPTG path. The added delay is supposed to impact the glitching behavior of the circuit positively. The third implementation of the circuit shown in Figure 2(c) is a logic hazard-free version of the circuit. As mentioned in the background, logic hazards are based on the circuit's logic and not its function. Unlike function hazards, logic hazards can be avoided using hazard preserving optimizations while simplifying the circuit's function. More details about the circuit and its different implementations can be found in.[5]

The original implementation and the two modified versions from Fontanarrosa et al. were modeled using the GDA tool iBioSim.[13] iBioSim is a software tool for designing, modeling, and analyzing genetic circuits. It is driven by community standards, not limited to genetic logic circuits, and allows for sharing the designs via data repositories like SynBioHub.[15] iBioSim has many analysis methods, including stochastic simulation methods based on Gillespie's algorithm.[16] Our initial analysis results were obtained using iBioSim's stochastic simulator.

The designs were then exported as *Systems Biology Markup Language* (SBML) models[17] and translated into PRISM models[18] using PRISM's SBML-to-PRISM translator. Further analysis was carried out using the *STochastic Approximate Model-checker for INfinite-state Analysis* (STAMINA).[14] STAMINA is an infinite-state stochastic model checking tool. Stochastic model checking is a formal verification technique for modelling and analyzing



(a) Original Version

(b) Two-Inverter Version

(c) Logic-Hazard-Free Version

Figure 2: Three different logic layouts for the circuit 0x8E. The three inducer molecules are *IPTG*, *aTc* and *Ara* and the output is *YFP*. The OR gate is represented by  and the NOR gate by  . (a) is the original layout as published in.[3] Version (b) has two added NOT gates that add an extra delay to the IPTG pathway. The NOT gate is represented by  . Version (c) is a logic-hazard-free implementation of the circuit's function. More details on the implementations (b) and (c) can be found in.[5]

systems that exhibit stochastic behaviors. Specifically, it provides provable guarantees for probabilistic properties of a system model. The underlying stochastic models for the genetic circuits analyzed in this paper are *continuous time Markov chain* (CTMC) models.[19] Although stochastic model checking can establish the correctness of probabilistic properties, real-world applications usually have a vast or infinite state space resulting in a scalability problem. STAMINA reduces these infinite state CTMC models to finite-state representations using state-space approximation methods. These reduced models are then manageable by existing stochastic model checkers like PRISM[18] and STORM.[20]

Fontanarrosa et al. identified twelve input transitions of the circuit that have a function hazard.[5] Using iBioSim for simulation, Figure 3(b) shows the probability of a glitch over time for 100,000 Monte Carlo runs for four selected input transitions with known hazards for the original layout of the circuit. For the analysis, the molecular counts are defined as follows: a high input signal equals 60 molecules. The output constraint is chosen to be 10 molecules for *0-hazards*. This means that if the output is supposed to stay low during an input transition, the run counts as a glitch if *YFP* surpasses the threshold of 10 molecules. For example, the red curve in Figure 3(b) shows the inducer transition from *IPTG*, *aTc*, *Ara* = (1, 1, 1) to (1, 0, 0). The output signal is supposed to remain low during this input change, but after 1000 seconds, in 92 percent of the runs the circuits output *YFP* glitches to a high state (i.e., exceeds 10 molecules). For input transitions where the output is supposed to remain high, the run is counted as a glitch if *YFP* drops below a molecule count of 30. For example, the cyan curve shows the inducer transition from *IPTG*, *aTc*, *Ara* = (0, 1, 1) to (0, 0, 0). In 86 percent of the runs, *YFP* drops below the threshold of 30 molecules, even though it is supposed to remain high. More information about the selection of the thresholds can be found in the methods section. It should be noted that circuits can also have a steady-state failure over time that is unrelated to the hazard. This is simply due to noise where a random set of reactions leads to an erroneous change in the *YFP* level. For example, consider the green graph in Figure 3(b), the graph shows a sharp increase to

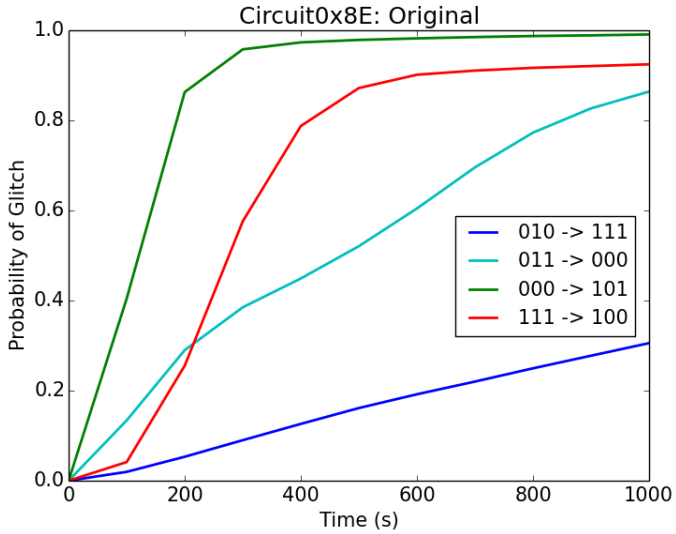
over 90 percent in the first 300 seconds. After that, the probability increases linearly with a small slope. If the simulation is run indefinitely, the probability will reach 100 percent due to the noisy behavior, which eventually will let the system fail. This analysis was limited to the first 1000 seconds to ensure that we separated these types of failures from those due to hazards.

The table in Figure 3(a) shows the glitch probabilities of all input transitions with known glitching behavior for the circuit's original implementation. The three columns of the table are the input transition, the probability of the glitch simulated in iBioSim, and the probability of the glitch calculated with STAMINA. For example, for the input transition $IPTG$, aTc, Ara = (1, 1, 1) to (1, 0, 0), iBioSim predicts a 92.5 percent chance of a glitch, while STAMINA verifies that the likelihood of a glitch in this case is between 91.1 and 99.7 percent.

Figure 3(d) shows the probability of a glitch over time for 100,000 Monte Carlo runs for the two-inverter version of the circuit shown in Figure 2(b) for the same four input transitions presented in Figure 3(b) for the original implementation. The results of the stochastic analysis using STAMINA for this version of the the circuit are shown in Table 3(c). The two-inverters were added to positively influence the input transition (0,0,0) to (1,0,1) shown as the green curve in Figure 3(b) and 3(d). Comparing the iBioSim simulation results from the original implementation in Table 3(a) with the modified model results in Table 3(c), it shows that the probability for this input transition was improved from 99 to 81 percent. Therefore, adding the two-inverters to the circuit results in an improvement of 18 percent while also adding stress to the host cell due to the larger size of the circuit.[21] On the contrary, the input transition (0,1,0) to (1,1,1) shown as the blue curve in Figure 3(b) and (d) has a probability of less than 30.5 percent for the original circuit compared to almost 44 percent for the two-inverter circuit, an increase of 13.5 percent. While adding two inverters results in a reduction in glitching for the green curve, the trade-off is an increase for the blue curve. The influence of the two inverters on the probabilities for the input transitions (0,1,1) to (0,0,0) (cyan curve) and (1,1,1) to (1,0,0) (red curve) are negligible.

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

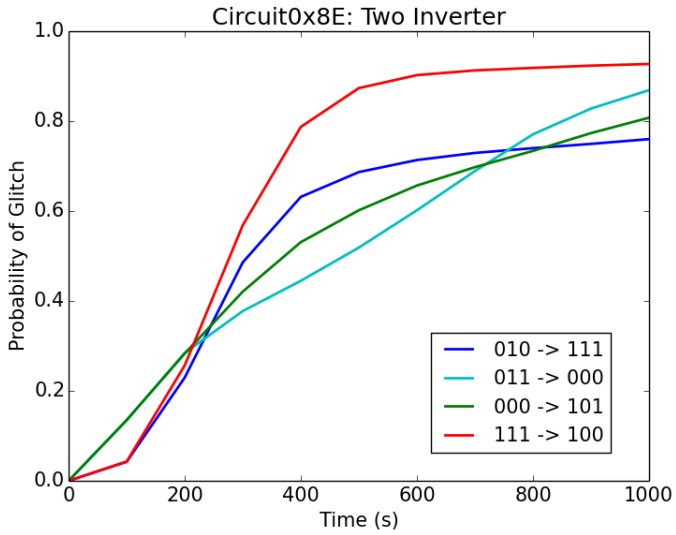
(a) Original Version



(b)

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

(c) Two-Inverter Version

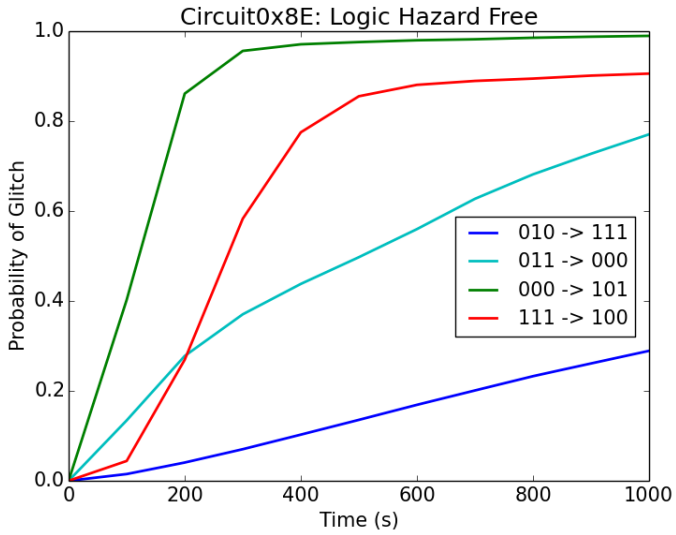


(d)

Figure 3: (a), (c), (e): Glitch probabilities of input transitions of the circuit simulated with iBioSim[13] and verified in STAMINA.[14] The order of the inputs is *IPTG*, *aTc*, *Ara*. The stochastic simulation in iBioSim was based on 100,000 stochastic runs. STAMINA was run with a target probability window width of 0.1 or 0.5(†). Models marked with the * were unable to achieve their target probability bound due to memory constraints. Out-of-Memory (OOM) indicates that STAMINA was not able to achieve probability window width of less than 0.8 before the host machine ran out of memory, while Max-Iterations-Reached (MIR) indicates the the maximum iterations (default 10) were reached, an indication that the convergence was too slow. (b), (d), (f): Probability of a glitch occurring over time for input changes known to have hazards calculated from 100,000 stochastic simulation runs in iBioSim. The legend shows the state transition of the three inputs *IPTG*, *aTc*, *Ara*.

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

(e) Logic Hazard Free Version



(f)

Figure 3: Cont.

Finally, Figure 3(f) shows the analysis on the same four input transitions for the logic hazard-free version of the circuit shown in 2(c). Table 3(e) shows iBioSim and STAMINA's results for the glitch probability of all input transitions with known glitching behavior for the logic hazard-free implementation. This analysis focuses on function hazards exclusively. Therefore, the results show that using hazard preserving optimizations may eliminate logic hazards, while influencing the probability of function hazards, in this case positively. Comparing the results of this version of the circuit with the results from the original implementation shows that redesigning the circuit using hazard preserving optimization also influences function hazards, highlighting the importance of this type of analysis.

Comparing iBioSim's results with STAMINA's shows that in some cases the estimated probability of iBioSim lies within or close to the bounds given by STAMINA. For example, for the results of the input transition $IPTG$, aTc, Ara = (0, 0, 0) to (0, 1, 1) from the original circuit (Table 3(a)), iBioSim simulates a probability of 82.713 percent. STAMINA calculates for the same input transition a probability window of 82.26 percent to 83.12 percent. However, in a few other cases STAMINA fails to provide a reasonably tight bound for the calculated probabilities. STAMINA's result for the input transition $IPTG$, aTc, Ara

= (0, 1, 0) to (1, 1, 1) for the two-inverter circuit (Table 3(c)) is 27 to 100 percent when it runs out-of-memory, providing no helpful information to the designer. In other cases, STAMINA provides large bounds like for the input transition *IPTG*, *aTc*, *Ara* = (1, 0, 1) to (0, 1, 1) for the two-inverter circuit (Figure 3(c)) with a bound of 91.59 percent to 100 percent. Although the bound is not very tight, it still provides enough information to inform the designer that the glitch probability is likely too high.

Comparing the iBioSim stochastic simulation results and the STAMINA verification results demonstrates that stochastic model checking is appropriate to analyze genetic circuits and produces guaranteed probability bounds that are consistent with simulation. However, compared to running stochastic simulations, stochastic model checking is memory and run-time intensive. All stochastic simulation and the model checking tasks were carried out on a computer with an AMD Ryzen Threadripper 12-Core 3.5 GHz Processor and 132 GB of RAM, running Ubuntu Linux (v18.04.3). The run-times for the stochastic simulation and model verification for all input transitions and circuit versions are given in Table 1.

The stochastic simulation of the transition *IPTG*, *aTc*, *Ara* = (1, 0, 0) to (1, 1, 1) on the two-inverter circuit had the longest run-time with a duration of 10 minutes and 12 seconds for 100,000 simulation runs. STAMINA was not able to compute an adequately tight probability window for the same input transition. The transitions *IPTG*, *aTc*, *Ara* = (1, 1, 1) to (1, 0, 0) and *IPTG*, *aTc*, *Ara* = (0, 0, 0) to (1, 0, 1) on the original circuit were the fastest stochastic simulations taking 1 minute and 31 seconds each. The same transition checked by STAMINA took only 49 seconds which is 54 percent of the time it took iBioSim. The longest run-time for stochastic model checking (not including MIR or OOM models) was for the input transition *IPTG*, *aTc*, *Ara* = (0, 1, 1) to (1, 0, 1) of the logic hazard free circuit, which was 6 hours 58 minutes and 33 seconds. State space generation took 1 hour 11 minutes and 1 second and the analysis took 5 hours 47 minutes and 32 seconds. All stochastic model checking runs marked with an asterisk in Figure 3(a),(c), and (e) report the run-time to obtain the provided result, and do not include the run-time between obtaining

Table 1: Run-time comparison of the stochastic simulations in iBioSim[13] and model verification in STAMINA.[14] The first column shows the input transition with the order of inputs being *IPTG*, *aTc*, *Ara*. The stochastic simulation in iBioSim was based on 100,000 Gillespie runs. The columns labeld iBioSim and STAMINA are split into three colums containing the run-times for the (OG) original layout (Figure 2 (a)), for the (TI) two-inverter layout (Figure 2 (b)), and the (LHF) logic hazard free layout (Figure 2 (c)) of the circuit. All stochastic simulation and the model checking tasks were carried out on a computer with an AMD Ryzen Threadripper 12-Core 3.5 GHz Processor and 132 GB of RAM, running Ubuntu Linux (v18.04.3).

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

this result and running out of memory in the next iteration, as this is not reported by the tool.

Discussion. As mentioned in the introduction, the inherent noisy and stochastic behavior of genetic circuits requires stochastic analysis. This paper presents hazard analysis results using both stochastic simulation with iBioSim and infinite state stochastic model checking using STAMINA. These results show that both simulation and model checking provide consistent results on the likelihood of static function hazards. While stochastic simulation can be faster depending on the number of runs required to obtain a high confidence result, model checking provides probability bounds that are guaranteed to include the actual probability of the event. This work only considers static function hazards. Model checking however, has the potential to verify more complicated error conditions, such as *dynamic hazards*.[6] In dynamic hazards, the output switches from one state to the other in a non-monotonic manner.

The predicted glitch probabilities discovered by the analysis methods presented in this paper can be applied to restrict the behavior of genetic circuits to produce more robust operation. In particular, a designer can avoid input transitions that exceed a minimum glitch probability threshold. For example, considering the original circuit results shown in Table 3(a), we can group the input transitions by their probability to glitch. For example, the transition *IPTG*, *aTc*, *Ara* = (0, 0, 0) to (1, 0, 1) has a high probability to glitch (over 98 percent) suggesting that this input transition must be avoided. On the other hand, the input transition *IPTG*, *aTc*, *Ara* = (1, 0, 0) to (1, 1, 1) has a lower probability to glitch of 30 percent, which might be acceptable for the intended use.

The predicted glitch probabilities can also be applied to design more robust genetic circuits. For example, the results for the original circuit shown in Table 3(a) indicate that the input transition *IPTG*, *aTc*, *Ara* = (1, 1, 1) to (0, 1, 0) glitches about 91 percent of the time. Contrary, the results for the two-inverter circuit shown in Table 3(c) indicate that this input transition only results in a glitch about 54 percent of the time. On the other

hand, for the input transition *IPTG*, *aTc*, *Ara* = (0, 1, 0) to (1, 1, 1) the original circuit glitches 30 percent of the time compared to the 76 percent of the time for the two-inverter circuit. Therefore, there is a trade-off to consider depending on which input transition is more critical for the application. Using the analysis presented in this paper allows a genetic designer to evaluate alternative designs *in silico* before building and testing them *in vivo*.

While STAMINA can sometimes produce wide probability bounds or take too much time or memory to produce a tight bound, precise bounds are not always necessary to guide design decisions. Namely, it may only be necessary to know if the probability of glitch is high or low. For example, determining if a glitch probability is higher than some threshold is often much easier than calculating the exact probability.

There are several additional areas of future work. While this paper has focused on the analysis of static functions hazards, we plan to perform further analysis of other hazards, such as dynamic and logic hazards. Also, this work uses a generic model with default parameters produced as described in,[4] and we are planning to extend this work to use a characterized dynamic model[22] in the future. This would enable the user to not only predict glitch probabilities, but also guide the selection of different specific parts to reduce these probabilities. Finally, we would like to verify these glitch probabilities *in vivo*.

Methods

The design of the genetic circuit and the generation of its model was achieved using the software tool iBioSim.[13,23,24] iBioSim is open-source *genetic design automation* (GDA) tool for the modeling, analysis, and design of genetic circuits that is being actively developed. This tool is intended to promote model-based design of genetic circuits using community-developed data standards such as the *synthetic biology open language* (SBOL),[25–28] the *systems biology markup language* (SBML),[29] and the *simulation experiment description markup language* (SED-ML).[30] iBioSim also allows for designs, models, and analysis results to be

shared via the SynBioHub data repository.[15] The following is a high-level description of the key features of iBioSim used in this project:

- **Genetic Circuit Design**

    1. The genetic circuits described in this paper were constructed using the sequence editor GDA tool SBOLDesigner.[31] This tool allows for the creation and editing of hierarchical genetic designs represented using the SBOL data standard and depicted using the SBOL Visual standard.[32–34]

    2. Parts for these designs were fetched from a collection stored in the Living Computing instance of SynBioHub[15] (`https://synbiohub.programmingbiology.org`).

- **Model Generation**

    1. The *Virtual Parts Repository* (VPR) model generator[35,36] is used to enrich the SBOL representation with non-DNA components and interactions between the components as described in.[4] For example, VPR adds the proteins produced by the genetic circuits and the interactions that indicate which coding sequence they are produced from, as well as the inhibition interactions between these proteins and promoters within the design. Finally, the small molecule used as inputs and their interactions are added.

    2. The computational model is produced using the SBOL to SBML converter integrated into iBioSim.[37] This converter is used to translate structural and functional information in SBOL to create a quantitative model expressed represented in SBML using generic or user-defined parameters.

    3. Finally, the iBioSim *graphical user interface* (GUI) was used to edit and refine the model.

- **Analysis**

1. iBioSim includes a variety of simulation methods to analyze SBML models such as *ordinary differential equations* (ODEs) and stochastic simulation. The simulations performed are encoded using SED-ML to allow exchange to other simulators. This project utilized the stochastic simulation written in the C-language.

2. Finally, iBioSim was used to view simulation results plotted in a graphical form.

For this paper, the models were further refined to reduce their vast state space. Instead of modeling the input molecules *IPTG*, *aTc*, and *Ara*, their corresponding internal molecules and complexes were used to reduce the number of species in the circuit. As an example, instead of modeling *Ara* and its complex formation to *AraAraC* that regulates the circuit, just the complex *AraAraC* was modeled. Additionally, instead of allowing protein production and degradation in steps of one, the model was adjusted to only allow molecules to be produced or degraded in steps of ten with a ten times reduction in reaction propensity.

For the stochastic simulation in this work, 60 molecules were chosen as a high input. This value is arbitrary since any input value above 60 molecules yields the same output molecular count. To identify thresholds that can be used to determine 0-hazards and 1-hazards, ODE analysis to characterize the sensitivity of a NOT gate was run. Figure 4 shows the number of molecules of the output over a sweep of the input molecular count.

Figure 4 shows that an input of ten molecules results in a decrease of over 40 molecules in the output. This suggests that a *0-hazard*, a hazard where the output is supposed to stay low, can already be critical if the output briefly passes a molecular count of 10. Therefore, the threshold for a *0-hazard* was set to 10. On the other hand, the threshold for a *1-hazard* was set to 30, since even a high signal of 30 molecules still results in a low output of under 10 molecules. This method is a simplified version of the more sophisticated approach by Baig and Madsen in.[38] Future work could utilize the tool D-VASim[38,39] to increase precision of the threshold analysis.

The stochastic simulation is based on 100,000 runs of the Gillespie's algorithm.[16] The number of runs was chosen to achieve reasonable confidence in the results. In statistics, the

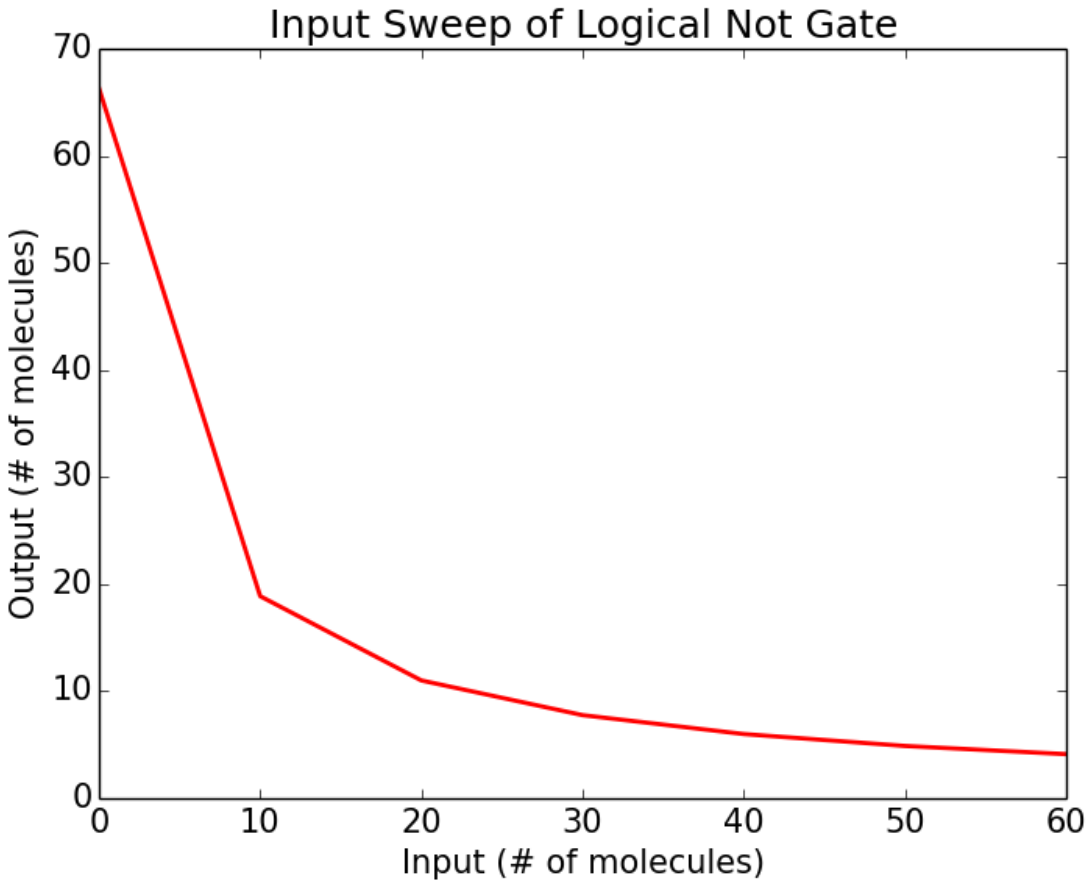


Figure 4: The output in molecular counts of a logical NOT gate over the number of molecules selected as input. The analysis helps selecting the threshold for *0-hazards* and *1-hazards*. The graph shows that an input signal of 10 molecules results in a decrease of the output of over 40 molecules. Contrary, a high input signal of 30 molecules and above results in a low output of under 10 molecules. The threshold to detect a *0-hazards* was selected to be 10 and the threshold for a *1-hazards* to be 30.

95 percent confidence interval is defined by $\bar{X} \pm 1.96\frac{\sigma}{\sqrt{n}}$ with $\bar{X}$ being the mean of the random variable, σ being the standard deviation of the population, and n being the number of runs. Therefore, the confidence interval tightens with an increasing number of runs. After running the stochastic simulations in iBioSim, the model was exported as an SBML file and converted to a PRISM model using the SBML-to-PRISM translator implemented in PRISM.[18] Finally, the PRISM model was passed to STAMINA for stochastic model checking.

While simulation can usually provide accurate results for complex systems, it cannot provide provable guarantees as to the true probabilities in question. Genetic circuit models, such as the ones used in this paper, are *continuous-time Markov chains* (CTMCs), which can be analyzed by probabilistic model checking tools such as PRISM.[18] Using these tools, the true probability associated with the CTMC model can be obtained. Such probabilistic analysis often suffers from state-space explosion due to the exponential increase in the state space needed to represent a system model's behavior over time. In fact, state explosion is guaranteed to occur in unbounded models, such as a genetic circuit without arbitrarily enforced molecule count limits. STAMINA[14] is a tool designed to use on-the-fly predictions to reduce very large or infinite state-spaces to a subset that is more tractable and can be analyzed by PRISM. STAMINA preserves the true probability by outputting a probability window with lower- and upper-bound probabilities that enclose the true probability. The true probability is guaranteed to be within the outputted bound.[40] Thus, STAMINA can be used to model check genetic circuits with unbounded molecule counts.

STAMINA truncates the state space in the following way: as each state is explored, it estimates the *state reachability probability*, which is the likelihood of reaching a state from the initial state of a model. States whose reachability probability falls below some small threshold κ are not explored, i.e., truncating the state space; instead, outgoing transitions of these states are directed toward an artificially created absorbing state A. Note that transitions to an existing state are preserved rather than directed toward A. The state reachability probability is estimated as follows. From a state s, each next state s' is obtained

by executing an enabled transition in s. The transition probability of each such transition is obtained from $r/E(s)$, where r is the reaction rate, a function of reactants, and $E(s)$ is the sum of all outgoing reaction rates from s. Let n be the number of incoming transitions into a state s', p_i be the transition probability of the i-th incoming transition from state s_i into s', and $\pi(s_i)$ be the state reachability probability for s_i. Then, $\pi(s') = \sum_{i=0}^{n} p_i \cdot \pi(s_i)$.

Once STAMINA completes the state space generation and truncation, it passes it to PRISM for Markov chain analysis. When the PRISM analysis is complete, the probability accumulated in the absorbing state A, P_A, is the reachability probability for states not explored by STAMINA. It follows that the true probability of a the property ϕ under verification lies between the probability of ϕ being satisfied in the existing state space, P_ϕ, and $P_\phi + P_A$. Thus the probability bound is $[P_\phi, P_\phi + P_A]$.

Originally, the version of STAMINA described in earlier papers,[14] was used to attempt to gather results. This version of STAMINA operated by running with a chosen κ value, model checking the state space in PRISM, and if the calculated bounds did not meet a desired level of precision, then reducing κ by some reduction factor and continuing state expansion. The genetic circuit models presented in this paper represent the most intensive models STAMINA has been tested on thus far, and it quickly became obvious that STAMINA could not obtain adequately tight probability bounds given the memory constraints of the machine used to run it. As a result, STAMINA was evaluated for possible optimizations, during which evaluation a number of issues and inefficiencies were discovered. One of these issues affected the accuracy of the bound output, while the remaining did not affect accuracy, but heavily affected efficiency and success at finding the most reachable states. In addition to fixing these issues, the main STAMINA algorithm was also restructured to allow for a more accurate estimation of the most reachable states. A full explanation of the new STAMINA algorithm and various fixes will be given in future work. For the scope of this work, it is adequate to note that the new algorithm starts with a fixed κ of 1, and repeatedly halves κ, starting from the initial state each iteration in order to re-calculate reachability probabilities,

but not passing to PRISM for checking each time. The repeated re-calculation of reachable states allows for a more accurate representation of how reachability behaves in the presence of cycles within the state graph. This process is repeated until the estimated reachability of the absorbing state drops below a proportion p of the desired probability window width. The state space is then passed to PRISM to be checked. If the obtained bounds are not tight enough, p is reduced and state-generation continues. Results shown in this paper were obtained with this newer algorithm implemented in STAMINA.

Associated Content

Additional Information

The latest version of iBioSim with the dynamic model generator, including source code, instructions, and related files, can be found online at `https://github.com/MyersResearchGroup/iBioSim`.

The current version of the software tool STAMINA used for state space reduction can be found online at `https://github.com/fluentverification/stamina` and the model PRISM checker used for the analysis at `https://www.prismmodelchecker.org`.

Information about the SBML-to-PRISM translator can be found at the following web page: `https://www.prismmodelchecker.org/manual/RunningPRISM/SupportForSBML`

All models created in this work are accessible at `https://github.com/fluentverification/stamina/tree/master/case-studies/HazardCct`. It includes the model designs encoded in both SBML and PRISM.

Special Issue Paper

Invited contribution from the 12th International Workshop on Bio-Design Automation.

Author Information

Corresponding Author

Lukas Buecherl - *Department of Biomedical Engineering, University of Colorado Boulder, Boulder, Colorado 80309, USA*; Email:lukas.buecherl@colorado.edu

Chris J. Myers - *Department of Electrical, Computer, and Energy Engineering, University of Colorado Boulder, Boulder, Colorado 80309, USA*; Email:chris.myers@colorado.edu

Authors

Riley Roberts - *Department of Electrical and Computer Engineering, Utah State University, Logan, Utah 84322, USA*

Pedro Fontanarrosa - *Department of Biomedical Engineering, University of Utah, Salt Lake City, Utah 84112, USA*

Payton J Thomas - *Department of Biomedical Engineering, University of Utah, Salt Lake City, Utah 84112, USA*

Jeanet Mante - *Department of Biomedical Engineering, University of Colorado Boulder, Boulder, Colorado 80309, USA*

Zhen Zhang - *Department of Electrical and Computer Engineering, Utah State University, Logan, Utah 84322, USA*

Author Contribution

L.B. and C.M. created the SBML models of the circuits, based on preliminary work by P.F. and J.M., ran the stochastic simulations in iBioSim, and build the PRISM models. P.T. provided a dockerized version of iBioSim to facilitate the stochastic simulation. R.R. and Z.Z. improved the software tool STAMINA and assisted in running the model verification. L.B. created all figures in their entirety and all images used in the TOC graphic. All authors contributed to the writing of the paper.

Notes

The authors declare no competing financial interest.

Acknowledgement

The authors thank Professor Hao Zheng of the University of South Florida and Professor Chris J. Winstead of Utah State University and their students for their feedback as members of the FLUENT Verification Project (`https://fluentverification.github.io`).

Funding Sources

The authors of this work are supported by the National Science Foundation under Grant Nos. 1856740, 1939892, and 1856733, DARPA FA8750-17-C-0229, Dean's Graduate Assistantship at the University of Colorado Boulder, and the University of Colorado Boulder Palmer Chair funds. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

(1) Gardner, T. S.; Cantor, C. R.; Collins, J. J. Construction of a genetic toggle switch in Escherichia coli. *Nature* **2000**, *403*, 339–342.

(2) Elowitz, M. B.; Leibler, S. A synthetic oscillatory network of transcriptional regulators. *Nature* **2000**, *403*, 335–338.

(3) Nielsen, A. A. K.; Der, B. S.; Shin, J.; Vaidyanathan, P.; Paralanov, V.; Strychalski, E. A.; Ross, D.; Densmore, D.; Voigt, C. A. Genetic circuit design automation. *Science* **2016**, *352*.

(4) Nguyen, T.; Jones, T. S.; Fontanarrosa, P.; Mante, J. V.; Zundel, Z.; Densmore, D.; Myers, C. J. Design of asynchronous genetic circuits. *Proc. IEEE* **2019**, 1–13.

(5) Fontanarrosa, P.; Doosthosseini, H.; Borujeni, A. E.; Dorfan, Y.; Voigt, C. A.; Myers, C. Genetic Circuit Dynamics: Hazard and Glitch Analysis. *ACS Synthetic Biology* **2020**, *9*, 2324–2338.

(6) Huffman, D. A. The Design and Use of Hazard-Free Switching Networks. *Journal of the ACM* **1957**, *4*, 47–62.

(7) Myers, C. J. *Asynchronous circuit design*; John Wiley & Sons, 2001.

(8) Elowitz, M. B. Stochastic Gene Expression in a Single Cell. *Science* **2002**, *297*, 1183–1186.

(9) Thattai, M.; van Oudenaarden, A. Intrinsic noise in gene regulatory networks. *Proceedings of the National Academy of Sciences* **2001**, *98*, 8614–8619.

(10) Raser, J. M. Noise in Gene Expression: Origins, Consequences, and Control. *Science* **2005**, *309*, 2010–2013.

(11) Lestas, I.; Paulsson, J.; Ross, N. E.; Vinnicombe, G. Noise in Gene Regulatory Networks. *IEEE Transactions on Automatic Control* **2008**, *53*, 189–200.

(12) Sanchez, A.; Choubey, S.; Kondev, J. Stochastic models of transcription: From single molecules to single cells. **2013**, *62*, 13–25.

(13) Watanabe, L.; Nguyen, T.; Zhang, M.; Zundel, Z.; Zhang, Z.; Madsen, C.; Roehner, N.; Myers, C. iBioSim 3: A Tool for Model-Based Genetic Circuit Design. *ACS Synthetic Biology* **2019**, *8*, 1560–1563.

(14) Neupane, T.; Myers, C. J.; Madsen, C.; Zheng, H.; Zhang, Z. STAMINA: STochastic Approximate Model-Checker for INfinite-State Analysis. 2019; pp 540–549.

(15) McLaughlin, J. A.; Myers, C. J.; Zundel, Z.; Mısırlı, G.; Zhang, M.; Ofiteru, I. D.; Goñi-Moreno, A.; Wipat, A. SynBioHub: A Standards-Enabled Design Repository for Synthetic Biology. *ACS Synthetic Biology* **2018**, *7*, 682–688.

(16) Gillespie, D. T. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry* **1977**, *81*, 2340–2361.

(17) Keating, S. M.; Waltemath, D.; König, M.; Zhang, F.; Dräger, A.; Chaouiya, C.; Bergmann, F. T.; Finney, A.; Gillespie, C. S.; Helikar, T. et al. SBML Level 3: an extensible format for the exchange and reuse of biological models. *Molecular Systems Biology* **2020**, *16*.

(18) Kwiatkowska, M.; Norman, G.; Parker, D. PRISM 4.0: Verification of Probabilistic Real-Time Systems. Computer Aided Verification. 2011; pp 585–591.

(19) Stewart, W. Introduction to the Numerical Solution of Markov Chains. *IEEE Computational Science and Engineering* **1996**, *3*, 93–98, Publisher: IEEE.

(20) Dehnert, C.; Junges, S.; Katoen, J.-P.; Volk, M. A Storm is Coming: A Modern Probabilistic Model Checker. Computer Aided Verification. 2017; pp 592–600.

(21) Borkowski, O.; Ceroni, F.; Stan, G.-B.; Ellis, T. Overloaded and stressed: whole-cell considerations for bacterial synthetic biology. *Current Opinion in Microbiology* **2016**, *33*, 123–130.

(22) Shin, J.; Zhang, S.; Der, B. S.; Nielsen, A. A.; Voigt, C. A. Programming Escherichia coli to function as a digital display. *Molecular Systems Biology* **2020**, *16*.

(23) Myers, C. J.; Barker, N.; Jones, K.; Kuwahara, H.; Madsen, C.; Nguyen, N.-P. D. iBioSim: a tool for the analysis and design of genetic circuits. *Bioinformatics* **2009**, *25*, 2848–2849.

(24) Madsen, C.; Myers, C. J.; Patterson, T.; Roehner, N.; Stevens, J. T.; Winstead, C. Design and Test of Genetic Circuits Using iBioSim. *IEEE Design & Test of Computers* **2012**, *29*, 32–39.

(25) Galdzicki, M.; Clancy, K. P.; Oberortner, E.; Pocock, M.; Quinn, J. Y.; Rodriguez, C. A.; Roehner, N.; Wilson, M. L.; Adam, L.; Anderson, J. C. et al. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature Biotechnology* **2014**, *32*, 545–550.

(26) Roehner, N.; Beal, J.; Clancy, K.; Bartley, B.; Misirli, G.; Grünberg, R.; Oberortner, E.; Pocock, M.; Bissell, M.; Madsen, C. et al. Sharing Structure and Function in Biological Design with SBOL 2.0. *ACS Synthetic Biology* **2016**, *5*, 498–506.

(27) McLaughlin, J. A.; Beal, J.; Mısırlı, G.; Grünberg, R.; Bartley, B. A.; Scott-Brown, J.; Vaidyanathan, P.; Fontanarrosa, P.; Oberortner, E.; Wipat, A. et al. The Synthetic Biology Open Language (SBOL) Version 3: Simplified Data Exchange for Bioengineering. *Frontiers in Bioengineering and Biotechnology* **2020**, *8*, 1009.

(28) Madsen, C.; Goñi Moreno, A.; P, U.; Palchick, Z.; Roehner, N.; Atallah, C.; Bartley, B.; Choi, K.; Cox, R. S.; Gorochowski, T. et al. Synthetic Biology Open Language (SBOL) Version 2.3. *Journal of Integrative Bioinformatics* **2019**, *16*.

(29) Hucka, M.; Finney, A.; Sauro, H. M.; Bolouri, H.; Doyle, J. C.; Kitano, H.; and the rest of the SBML Forum:,; Arkin, A. P.; Bornstein, B. J.; Bray, D. et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* **2003**, *19*, 524–531.

(30) Waltemath, D.; Adams, R.; Bergmann, F. T.; Hucka, M.; Kolpakov, F.; Miller, A. K.; Moraru, I. I.; Nickerson, D.; Sahle, S.; Snoep, J. L. et al. Reproducible computational biology experiments with SED-ML - The Simulation Experiment Description Markup Language. *BMC Systems Biology* **2011**, *5*, 198.

(31) Zhang, M.; McLaughlin, J. A.; Wipat, A.; Myers, C. J. SBOLDesigner 2: An Intuitive Tool for Structural Genetic Design. *ACS Synthetic Biology* **2017**, *6*, 1150–1160.

(32) Quinn, J. Y.; Cox, R. S.; Adler, A.; Beal, J.; Bhatia, S.; Cai, Y.; Chen, J.; Clancy, K.; Galdzicki, M.; Hillson, N. J. et al. SBOL Visual: A Graphical Language for Genetic Designs. *PLOS Biology* **2015**, *13*.

(33) Beal, J.; Nguyen, T.; Gorochowski, T. E.; Goñi-Moreno, A.; Scott-Brown, J.; McLaughlin, J. A.; Madsen, C.; Aleritsch, B.; Bartley, B.; Bhakta, S. et al. Communicating Structure and Function in Synthetic Biology Diagrams. *ACS Synthetic Biology* **2019**, *8*, 1818–1825.

(34) Baig, H.; Fontanarossa, P.; Kulkarni, V.; McLaughlin, J.; Vaidyanathan, P.; Bartley, B.; Bhakta, S.; Bhatia, S.; Bissell, M.; Clancy, K. et al. Synthetic biology open language visual (SBOL Visual) version 2.3. *Journal of Integrative Bioinformatics* **2021**,

(35) Misirli, G.; Hallinan, J.; Wipat, A. Composable Modular Models for Synthetic Biology. *ACM Journal on Emerging Technologies in Computing Systems* **2014**, *11*, 1–19.

(36) Cooling, M.; Rouilly, V.; Misirli, G.; Lawson, J.; Yu, T.; Hallinan, J.; Wipat, A. Standard Virtual Biological Parts: A Repository of Modular Modeling Components for Synthetic Biology. *Bioinformatics (Oxford, England)* **2010**, *26*, 925–31.

(37) Roehner, N.; Zhang, Z.; Nguyen, T.; Myers, C. J. Generating Systems Biology Markup Language Models from the Synthetic Biology Open Language. *ACS Synthetic Biology* **2015**, *4*, 873–879.

(38) Baig, H.; Madsen, J. Simulation Approach for Timing Analysis of Genetic Logic Circuits. *ACS Synthetic Biology* **2017**, *6*, 1169–1179.

(39) Sanaullah,; Baig, H.; Madsen, J.; Lee, J.-A. A Parallel Approach to Perform Threshold Value and Propagation Delay Analyses of Genetic Logic Circuit Models. *ACS Synthetic Biology* **2020**, *9*, 3422–3428.

(40) Liò, P.; Zuliani, P. *Automated Reasoning for Systems Biology and Medicine*; Springer, 2019.

Graphical TOC Entry