Accurate Prediction of Random Telegraph Noise Effects in SRAMs and DRAMs

Karthik V. Aadithya, Alper Demir, *Fellow, IEEE,* Sriramkumar Venugopalan, and Jaijeet Roychowdhury, *Fellow, IEEE*

***Abstract*—With aggressive technology scaling and heightened variability, circuits such as SRAMs and DRAMs have become vulnerable to random telegraph noise (RTN). The bias dependence (i.e., non-stationarity), bi-directional coupling, and high inter-device variability of RTN present significant challenges to understanding its circuit-level effects. In this paper, we present two computer-aided design (CAD) tools, SAMURAI and MUSTARD, for accurately estimating the impact of non-stationary RTN on SRAMs and DRAMs. While traditional (stationary) analysis is often overly pessimistic (e.g., it overestimates RTN-induced SRAM failure rates), the predictions made by SAMURAI and MUSTARD are more reliable by virtue of non-stationary analysis.**

***Index Terms*—1/f noise, circuit noise, circuit simulation, computational modeling, computer-aided analysis, DRAM chips, error probability, failure analysis, SRAM chips.**

I. Introduction

RANDOM TELEGRAPH noise (RTN) has become an important challenge associated with designing circuits in deep submicron technologies. Indeed, with aggressive CMOS scaling and increased parameter variability, RTN has emerged as a critical limiting factor producing transient failures in SRAMs, DRAMs, oscillators, PLLs, and many radio frequency (RF) circuits [1]–[5].

In SRAMs, RTN has been shaving away design margins for a while. This is seen from Fig. 1 [1], which quantifies, in supply voltage terms, the adverse effects of various non-idealities on SRAM design margins, as technology has progressed from 90 to 22 nm. In the figure, each CMOS technology is represented by a stacked bar, onto which the design margin impacts of different non-idealities (including static noise, local and global parameter variation, NBTI, and RTN) are successively added.[1] The downward sloping dashed line depicts supply voltage scaling.

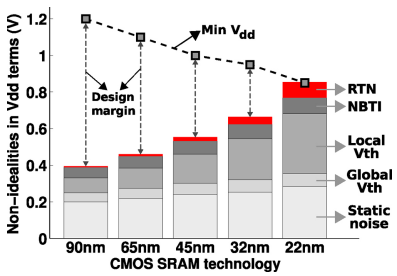
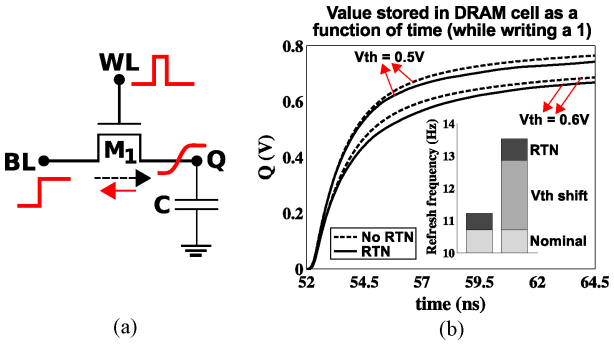


Fig. 1. Impact of RTN on SRAM design margins (data courtesy Yasumasa Tsukamoto, Renesas Electronics Corporation, Japan).



(a) (b)

Fig. 2. (a) Writing the bit 1 to a 1T DRAM cell. The dashed arrow indicates the direction of drain current, while the solid arrow indicates the direction of the noise current due to RTN. (b) Effect of RTN on the stored DRAM value for two different threshold voltages.

From the figure, it is seen that the impact of RTN has been steadily increasing under continued CMOS scaling. Indeed, of all the variability or noise sources depicted in Fig. 1, RTN is the fastest growing contributor to design margin degradation. At the 22-nm node, RTN (coming on top of parameter variability) is large enough to push the stacked bar above the minimum supply voltage, thereby driving design margins negative. In fact, RTN-induced SRAM failures, leading to transient read or write bit errors, have already been experimentally reported [1], [4].

In addition to the unfavorable impact on SRAMs, RTN is also considered responsible for variable retention times in DRAMs [6]. Fig. 2(a) shows a standard DRAM cell to which the bit "1" is written. Fig. 2(b) depicts the stored value Q of the DRAM cell over time, both in the absence and in the presence of RTN, for two different threshold voltages. To compensate for RTN, it is necessary to refresh the cell more often, which in turn increases power consumption and reduces speed of operation. The bar chart in Fig. 2 shows the increase in refresh

Manuscript received March 9, 2011; revised January 10, 2012 and April 3, 2012; accepted May 25, 2012. Date of current version December 19, 2012. This paper was recommended by Associate Editor A. Elfadel.

K. V. Aadithya, S. Venugopalan, and J. Roychowdhury are with the University of California, Berkeley, CA 94770 USA (e-mail: aadithya@berkeley.edu; sriram@eecs.berkeley.edu; jr@eecs.berkeley.edu).

A. Demir is with the Department of Electrical and Electronics Engineering, Koç University, Istanbul 34450, Turkey (e-mail: aldemir@ku.edu.tr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2012.2212897

[1]We note that the noise effects of Fig. 1 do not add up linearly toward design margin degradation. However, Fig. 1 is useful as a rule of thumb for variability or noise-aware SRAM design.

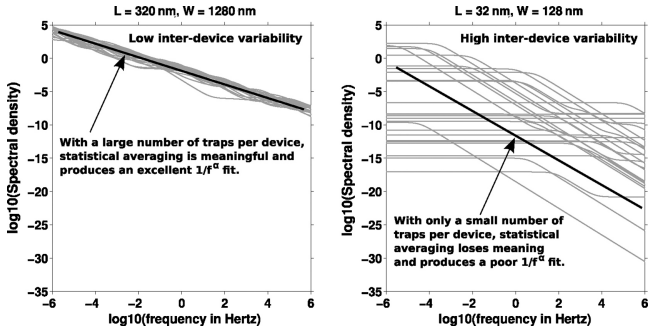


Fig. 3. Spectral density plots for 25 randomly sampled devices in two CMOS technologies.

frequency necessitated by RTN, both in the presence and in the absence of the threshold voltage shift.

The magnitude of RTN grows as $1/WL$, a rate much faster than many other sources of variability[2] (e.g., random dopant fluctuation grows only as $1/\sqrt{WL}$) [7]. In the future, therefore, it is expected that RTN will affect more circuits adversely, and will affect them more adversely. Thus, for continued CMOS scaling, it is necessary to develop new CAD techniques that enable RTN-aware circuit design, while simultaneously accounting for other sources of variability.

However, the very nature of RTN, from how it originates at the device level to how it impacts performance at the circuit level, poses several challenges for CAD tools.

Non-stationarity: At the device level, RTN is produced by random processes, involving the capture and release of charge carriers by dangling bonds known as "traps" [2], [3], [8]. The statistics of these processes are strongly dependent on time-varying bias conditions (e.g., gate voltages). Therefore, there is an inherent non-stationarity associated with RTN, i.e., the statistics of RTN change with time rapidly and continuously. As a result, most existing noise analysis approaches (which are valid only in the stationary domain) do not apply to RTN; it is necessary to develop more powerful techniques.

High inter-device variability: Analytical approaches to RTN typically assume a large number of traps, leading to the classic $1/f$ stationary characteristic for RTN [8], [9], with minimal inter-device variation. For today's small devices, however, this assumption is invalid [2], [10], [11]. Detailed trap profile models (corroborated by measured data) suggest that, in deeply scaled technologies, only about one to two traps are active at any given bias [2], [10], [12], [13], causing significant inter-device variation. For example, Fig. 3 shows the spectral density plots for 25 devices (randomly generated using the trap profiling model of [2], and held at a constant bias) from two CMOS technologies. While inter-device variation is negligible in the older technology (left), it is significant in the newer technology (right), creating additional challenges for CAD tools.

Bi-directional coupling: As noted above, the statistics of RTN exhibit a complicated bias dependence. This leads to a bi-directionally coupled interaction between RTN and the rest of the circuit, i.e., the time-varying biases in the circuit affect the statistical parameters of RTN, while RTN simultaneously produces changes in these very biases (illustrated in Fig. 6). Thus, RTN cannot be considered in isolation from the rest of the circuit; instead, the circuit and its RTN evolve together as a coupled system—a feature that makes RTN characterization especially difficult.

The challenges imposed by the above three innate characteristics of RTN, namely, non-stationarity, high inter-device variability, and bi-directional coupling, have hampered the development of CAD tools for RTN characterization. Indeed, even though detailed, trap-level equations for RTN generation have been available for decades [2], [8], there is still no CAD tool that incorporates these models to achieve circuit-level characterization of non-stationary RTN. Instead, existing CAD techniques are limited in scope. For example, Tian and El Gamal [3] have derived analytical RTN expressions for constant and switched gate bias, and Roy and Enz [14] have extended these to periodic gate bias. However, such results do not apply to circuits such as SRAMs and DRAMs, which are subject to large, rapid, and non-periodic bias swings. For such applications, Ye *et al.* [15] recently proposed a 2-stage comparator topology, driven by white noise, for generating RTN. However, this applies only to stationary RTN at constant bias; it cannot perform non-stationary analysis. Moreover, this method is time-consuming (Section VII) because it requires time-domain white noise simulation.

Against this background, in this paper, we perform the following.

1) We present SAMURAI,[3] a computational technique for trap-level, non-stationary analysis of RTN in SRAMs/DRAMs under time-varying biases and inter-device variability. SAMURAI has been implemented as a stand-alone module interoperable with any existing circuit simulator (without modifying the simulator). The only aspect that SAMURAI does not address is bi-directional coupling; for this, we have developed MUSTARD.
2) We present MUSTARD,[4] a second technique to predict the impact of RTN on SRAMs and DRAMs. In addition to all of SAMURAI's features, MUSTARD offers the added capability of bi-directionally coupled RTN analysis. Thus, MUSTARD enables accurate, non-stationary, bi-directionally coupled, discrete stochastic RTN trace generation seamlessly integrated with deterministic, continuous circuit simulation. Unlike SAMURAI, however, MUSTARD requires modifying the underlying circuit simulator, the payoff being improved accuracy over SAMURAI. Thus, MUSTARD and SAMURAI each have their own strengths, and together they overcome the challenges associated with circuit-level RTN analysis.
3) We validate SAMURAI against analytical expressions known for stationary RTN (Section V), by showing that the statistical properties of SAMURAI-generated RTN

[2] W and L denote the width and length, respectively, of minimum-sized devices in the latest generation CMOS technology.

[3] SAMURAI stands for SRAM analysis via Markov uniformization with RTN awareness incorporated, and was presented [16] at DATE 2011. This paper elaborates on [16], providing additional results and comparisons against a previously published method [15].

[4] MUSTARD stands for Markov uniformization based simulation of trap activity for RTN aware design, and was presented [17] at DAC 2011. This paper provides greater detail, validation, comparisons against SAMURAI, and algorithmic extensions.

traces closely match analytical expressions. In effect, this also validates MUSTARD because, for the stationary case, MUSTARD reduces to SAMURAI.

4) We highlight the differences between stationary RTN analysis and SAMURAI, using a 22-nm 6T SRAM cell (Section VI). The results indicate that stationary analysis, being overly pessimistic, can predict RTN-induced SRAM failures even when there are none. By contrast, SAMURAI and MUSTARD do not make such overly pessimistic predictions.
5) We compare SAMURAI with the 2-stage method recently proposed by Ye *et al.* [15], from both an accuracy and an efficiency perspective (Section VII). We demonstrate that SAMURAI is not only more accurate, but also significantly more efficient than 2-stage RTN generation.
6) We examine the key differences between SAMURAI and MUSTARD, using as example a 22-nm 6T SRAM cell (Section IX). Our results indicate that, until the first RTN-induced failure, both SAMURAI and MUSTARD make similar predictions. However, after that, SAMURAI's predictions tend to be less reliable than MUSTARD's.
7) We apply MUSTARD to duplicate experimentally observed RTN-induced SRAM failures. We also generate statistical characterizations of SRAM bit errors due to RTN, in the presence of parameter variation.
8) We present MUSTARD-generated results showing the effect of RTN on DRAM retention times.

II. Existing Methods for RTN Characterization

Three approaches to RTN characterization can be distinguished from the literature: 1) analytical expressions at the device level; 2) measurement-based characterizations at the circuit level; and 3) simulations at both levels.

Analytical expressions: Much work has been done on the stochastic modeling of trap activity, the principal mechanism for RTN generation. Physics-based equations [2], [8] have been developed to describe the bias-dependent statistics of trap activity. Detailed device models [14], [18] are also available for translating trap activity to RTN noise currents. Several models [2] have also been proposed for obtaining realistic trap profiles for today's deep submicron technologies. Therefore, at the device level, it is possible to construct realistic, non-stationary models for RTN from first principles.

However, at the circuit level, analytical expressions are available only for the constant-bias (stationary) case [2], [8] and the periodic-bias (cyclo-stationary) case [3], [14], neither of which applies to circuits such as SRAMs/DRAMs.

Measurement-based characterizations: These involve subjecting post-fabrication SRAM/DRAM arrays to a large number of measurement tests, in the hope of detecting vulnerabilities to RTN [1]. This is the primary mode of RTN analysis available today for SRAMs and DRAMs. In this context, accelerated testing techniques have also been developed [4].

However, it is costly to correct errors discovered during post-Silicon measurement. In addition, measurements usually do not provide insight into why circuit failures occur. For instance, whereas a measurement can indicate that an SRAM cell is vulnerable to RTN, it cannot pinpoint which trap-level events can trigger a failure. Moreover, unlike other sources of variability, RTN is a temporal effect that cannot be completely characterized by measurement data alone. For example, identical tests carried out on the same SRAM/DRAM chip at different times can yield completely different success or failure outcomes [1]. Thus, even if a large number of measurements indicate that a chip is robust to RTN, it may in fact not be so.

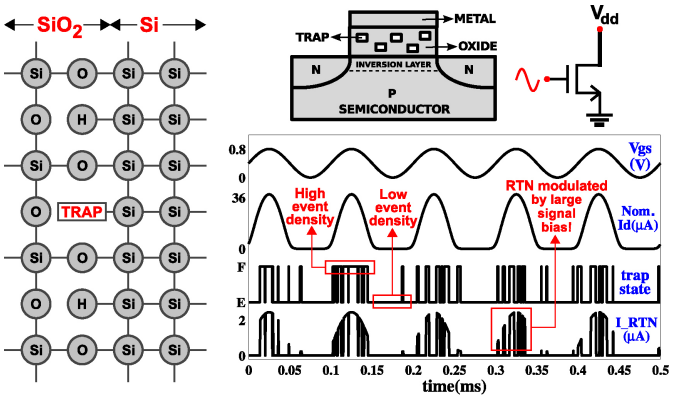


Fig. 4. (a) Dangling bonds at the Si/SiO_2 interface. (b) Trap activity under sinusoidal gate bias, and its modulation by large signal drain current.

Simulation-based characterizations: These are recent developments in RTN analysis. The main idea is that, because the number of active traps in today's devices is small (averaging about 1 to 2 [10], [12], [13]), each trap can be stochastically simulated (for example, alongside a SPICE run). Whenever such a simulation predicts an RTN-induced failure, it is possible to examine the simulation trace to investigate the RTN events that triggered the failure. Moreover, a simulation-driven methodology accounts for RTN in the design phase itself, which greatly reduces the risk of unpleasant findings during the (much later) measurement or testing phase.

The state-of-the-art simulation-driven strategy (excluding SAMURAI and MUSTARD) is the one proposed by Ye *et al.* [15], which uses a 2-stage comparator circuit to generate stationary RTN under constant bias, starting from an independent white noise source for each trap. However, this method cannot truly reproduce non-stationary RTN in circuits such as SRAMs and DRAMs. So, we believe that the predictive utility of this method in the context of non-stationary RTN in SRAMs/DRAMs is not clear. Moreover, as Section VII shows, this method can be very time-consuming.

III. Unidirectionally and Bi-Directionally Coupled Models for RTN

As mentioned in Section I, RTN is produced by the random capture and release of electrons by dangling bonds in the device oxide layer [8]. As Fig. 4(a) depicts, the oxide layer and the oxide-semiconductor interface contain silicon (Si) atoms with unsatisfied valences, called dangling bonds or "traps." When the device is on, each trap can randomly 1) capture an electron from the inversion layer, and 2) release the captured electron back into the inversion layer [8].

Thus, at any given time, each trap has two possible states: *filled* (with an electron) and *empty*. An *empty* trap can become

filled by capturing an electron, whereas a *filled* trap can become *empty* by releasing its captured electron.

Also, whenever a trap becomes *filled*, its captured electron modifies 1) the electric field and electron mobility in the inversion layer, and 2) the number density of charge carriers contributing to the transistor current [19]. Therefore, every capture or release event by a trap brings about a change in the device current, which is observed as a random waveform $I_{RTN}(t)$ opposing the nominal drain current $I_d(t)$.

Furthermore, the propensity of a trap to capture or release an electron is not constant, but depends on the instantaneous gate bias $V_{gs}(t)$ [2], [3], [14]. Mathematically, given that a trap tr is *empty* (*filled*) at time t, the probability that it changes state to become *filled* (*empty*), within a short time interval dt, is given by $\lambda_{c,tr}(t)dt$ ($\lambda_{e,tr}(t)dt$), where $\lambda_{c,tr}(t)$ ($\lambda_{e,tr}(t)$) is called the capture (emission) propensity of the trap tr at time t.[5] Physics-based models are available for the bias-dependent capture and emission propensities [2], [8], [20].

For simplicity, we first develop our RTN analysis techniques (Algorithms 1 and 2) assuming that, for each trap, the sum $\lambda_c + \lambda_e$ is constant (independent of bias) with time, while the ratio λ_e/λ_c can vary with time, depending on the instantaneous gate bias $V_{gs}|_t$. It is this bias dependence that causes the electron capture or release process to be non-stationary. In Algorithm 3, we relax this assumption and allow both the sum $\lambda_c + \lambda_e$ and the ratio λ_e/λ_c to be arbitrary functions of (time-varying) bias conditions.

Finally, we need a formula that relates individual trap states (i.e., *filled* or *empty*) to the collective noise current $I_{RTN}(t)$. One model for this is the following equation [18]:

$$I_{\rm RTN}(t) = \frac{N_{\rm filled}(t)\ q}{WL\ C_{ox}(V_{\rm gs}(t) - V_{\rm th})}\ I_d(t) \qquad (1)$$

where $N_{\rm filled}$ denotes the number of *filled* device traps, q is the electronic charge ($\sim 1.6 \times 10^{-19}$ Coulomb), W and L are the device dimensions, $C_{\rm ox}$ is the oxide capacitance per unit area, and $V_{\rm th}$ is the threshold voltage. More elaborate models have also been suggested [20].

Thus, the net effect $I_{RTN}(t)$ is that of a non-stationary electron capture or release process, whose resulting trap occupancy function $N_{\rm filled}(t)$ is modulated by a bias-dependent large signal waveform. Fig. 4(b) illustrates this for a trap in an NMOS device whose gate is driven by a sinusoidal voltage source. The figure depicts the nominal biases $V_{gs}(t)$ and $I_d(t)$, in addition to the trap occupancy function and the noise current $I_{RTN}(t)$. Note that, because the gate bias is time-varying, the density of capture or release events is non-uniform, i.e., some time intervals have a low event density, while others have a high density, which provides a visual cue that the underlying capture/release process is non-stationary.

To incorporate all the above aspects of RTN without loss of generality, SAMURAI and MUSTARD use the mathematical abstraction of a time-inhomogeneous Markov chain with a hypercube state transition graph (described below).

[5]Thus, given the present state of each trap, its future statistics are completely independent of the past. This is a fundamental characteristic of RTN, known as the Markovian property. It is supported by decades of measured data [2], [3], [5], [8], [9], [18].

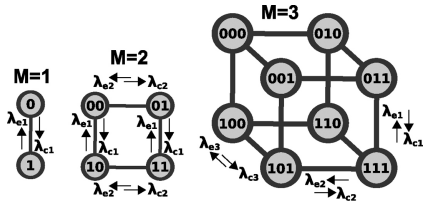


Fig. 5. Hypercube Markov state transition graphs for one, two, and three trap systems.

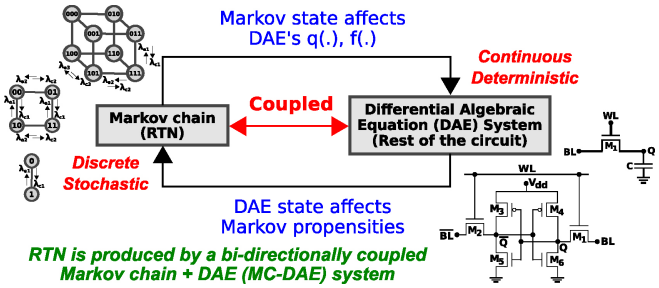


Fig. 6. Bi-directionally coupled model for RTN. A non-stationary Markov chain that drives, and is, in turn, driven by a DAE system.

Consider a circuit with M traps, where each trap contributes to the I_{RTN} of a specific device (multiple traps can belong to the same device). Each trap has two possible states; so the M-trap system has 2^M possible states, which can be encoded using M bits [with one bit per trap, where 0 (1) denotes *empty* (*filled*)]. Mathematically, this corresponds to a state transition graph that is an M-dimensional hypercube (Fig. 5 shows this for $M = 1,\ 2,$ and 3). Each dimension represents a unique trap, i.e., all hypercube edges along a given dimension denote capture or release events involving a unique trap. So, for every trap tr, all edges along the tr-dimension are annotated with the propensities $\lambda_{c,tr}(t)$ and $\lambda_{e,tr}(t)$ (Fig. 5), resulting in a time-inhomogeneous Markov chain.

In a circuit, however, the M-trap system is not isolated; rather, its (discrete) state evolves simultaneously with the underlying circuit, which has its own (continuous) state vector $\vec{x}$ of voltages and currents, represented by a differential algebraic equation (DAE) system D [21], given by

$$D: \quad \frac{d}{dt}\vec{q}(\vec{x}) + \vec{f}\,(\vec{x}) + \vec{b}(t) = \vec{0}.$$

Thus, RTN is produced by an M-dimensional hypercube Markov chain, whose time-varying propensities are determined by a state vector $\vec{x}$, which itself evolves according to a DAE system D, whose $\vec{q}$ and $\vec{f}$ functions are, in turn, determined by the Markov chain's state. This bi-directionally coupled RTN model is summarized by Fig. 6.

We note that commercial SPICE simulators typically simulate only deterministic DAEs (circuit equations), not the Markov/DAE systems above. To circumvent this, one can implement a non-stationary Markov process within a device model such as BSIM. However, this may involve significantly modifying the simulation engine, the device model, and the interaction between the two. Instead, we have implemented MUSTARD as an independent module (separate from the device model), so that the modifications are largely confined to the simulation engine. To use MUSTARD, one can either modify one's existing circuit simulator to interface with the MUSTARD module, or switch to the circuit simulator that

Algorithm 1: Non-stationary RTN generation in SAMURAI

```
   Input: Trap profile, Bias {V_gs(t), I_d(t) ...}, t_0, t_f
   Output: Realistic I_RTN(t) trace in time interval [t_0, t_f]
1  foreach trap tr in the device do
2      compute λ_c(t), λ_e(t), t ∈ [t_0, t_f], for tr (e.g., use the model in [2]);
3      λ* = λ_c(t_0) + λ_e(t_0);
4      curr_time = t_0; curr_state = tr.init_state;
5      times = [curr_time]; states = [curr_state];
6      while curr_time < t_f do
7          next_cand_time = curr_time + exprand(1/λ*);
8          curr_time = next_cand_time;
9          if curr_time > t_f then break;
10         if curr_state == 1 then
11             λ_next = λ_e(curr_time)
12         else
13             λ_next = λ_c(curr_time)
14         end
15         bool change_the_state = rand() < λ_next/λ*;
16         if change_the_state then
17             times.append(curr_time);
18             states.append(curr_state);
19             curr_state = (curr_state == 1) ? 0 : 1;
20             times.append(curr_time);
21             states.append(curr_state);
22         end
23     end
24     trap_occupancy[tr] = [times, states];
25 end
26 compute I_RTN(t) from trap_occupancy[tr] (use, e.g., Eq. (1))
```

is built into MUSTARD. Sometimes, however, neither choice may be acceptable, and one may require the RTN analysis module to work with existing simulators as they are. For this, we have developed SAMURAI.

SAMURAI approximates the above bi-directionally coupled Markov or DAE system with a unidirectionally coupled Markov or DAE system. In SAMURAI, the time-varying effect of the DAE state on the Markov propensities (i.e., non-stationarity) is fully taken into account. However, the DAE itself is not changed as and when individual RTN events occur. Instead, the effects of RTN are incorporated in a new DAE constructed at the end of the simulation (after an entire train of RTN events). Thus, during a SAMURAI run, the influence of individual RTN events, through the DAE, on the statistics of future RTN events, is not taken into account. We call this approximation the unidirectionally coupled RTN model.

IV. SAMURAI: A CAD Tool for Unidirectionally Coupled RTN Simulation by Markov Uniformization

We now discuss how to simulate the non-stationary, unidirectionally coupled RTN model using Markov uniformization, the core technique behind SAMURAI. Given a circuit netlist (e.g., an SRAM/DRAM), and a time interval $[t_0, t_f]$ during which the circuit's voltages and currents evolve continuously, the goal is to construct a realistic, non-stationary RTN trace for each device in the circuit, over $[t_0, t_f]$. For a single device, assuming that $\lambda_c + \lambda_e$ is constant for each trap [2], this is achieved by Algorithm 1.

Algorithm 1 takes as input: 1) the trap profile of the device (i.e., the position y_{tr} and energy E_{tr} of each trap), and 2) the time-varying bias conditions (e.g., $V_{gs}(t)$). The former is obtained either from measured data or from technology-specific trap profile models (e.g., [2]). The latter is obtained by SPICE simulating the circuit over $[t_0, t_f]$. Algorithm 1 outputs an $I_{RTN}(t)$ trace for the device, whose (time-varying) statistics are guaranteed to be identical to those of the unidirectionally coupled, non-stationary model of Section III.

Algorithm 1 works by generating more trap activity than necessary, and then discarding some of this activity to preserve the time-varying RTN statistics exactly. Line 3 computes λ^*, an upper bound on the sum of the outward propensities from every hypercube state. In each iteration of the while loop (line 6), a candidate capture or release event is generated (line 7) corresponding to a stationary two-state Markov chain with both propensities set to λ^*. Thus, the original non-stationary Markov chain is first uniformized into an easy-to-simulate, stationary (but high-rate) Markov chain. However, not all events in the high-rate chain correspond to the original Markov chain: some high-rate events are spurious, and need to be discarded. Algorithm 1 does this probabilistically (line 15), by making a randomized decision to either keep or discard each high-rate event. This exactly restores the non-stationarity of the original Markov chain (as proved in [22]–[24]).

For circuits with multiple transistors (e.g., SRAMs), we apply Algorithm 1 individually to each transistor, to obtain one noise current source $I_{RTN}(t)$ per transistor. We now construct a new circuit by including these noise sources between the source and drain of the corresponding devices. Thus, SAMURAI's non-stationary analysis involves two SPICE simulations: 1) a simulation of the original circuit to obtain the time-varying propensities, and 2) a simulation of a new circuit derived by augmenting the original circuit with RTN sources. These simulations can be performed by any SPICE simulator, without modifying the simulator. Our paper [16] illustrates these steps in detail, using a 90 nm SRAM cell and SpiceOPUS [25] as the simulator.

V. SAMURAI: Validation Against Analytical Expressions

As mentioned before, SAMURAI can generate non-stationary RTN traces under arbitrarily time-varying bias conditions. Although analytical expressions are not available for such a general case, they are known for the restricted case of constant gate bias [3], [8]. We now validate SAMURAI against these expressions, as follows.

1) We run three validation experiments, using typical values for the parameters V_{gs}, E_{tr}, and y_{tr}. In each experiment, we fix two of these, and sweep the third. We simulate these trap configurations under constant gate bias using Algorithm 1.
2) Algorithm 1 returns a trace $I_{RTN}(t)$, from which we estimate the autocorrelation [26] $R(\tau) = E[I_{RTN}(t)I_{RTN}(t+\tau)]$.
3) We translate the above results into the frequency domain, by computing the power spectral density (PSD) [26] $S(f)$ numerically from $R(\tau)$.
4) We plot $R(\tau)$ and $S(f)$ alongside analytical expressions obtained from [3] and [8]. To understand the relative importance of RTN, we also plot the PSD of thermal noise in the device.

The results in Fig. 7(a)–(f) show that the RTN traces predicted by SAMURAI closely match analytical expressions in both the time domain [autocorrelation plots (a)–(c)] and the frequency domain [spectral density plots (d)–(f)].

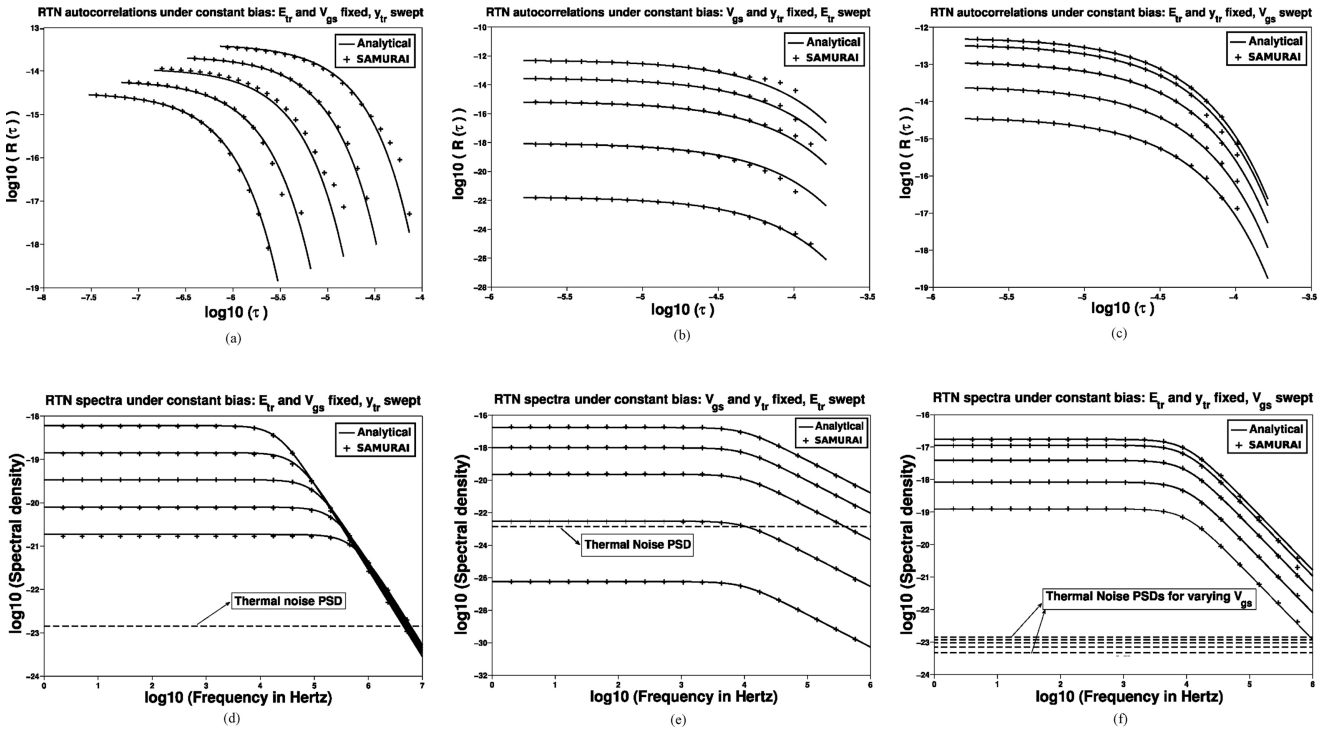


Fig. 7. Plots showing that the RTN traces generated by SAMURAI closely match analytical predictions in both the [autocorrelation plots (a)–(c)] time domain and the [spectral density plots (d)–(f)] frequency domain. For (a) and (d), y_{tr} is swept uniformly in $[0.2T_{ox}, 0.8T_{ox}]$, where T_{ox} is the device oxide thickness. For (b) and (e), E_{tr} is swept uniformly in $[E_{\min}, E_{\max}]$, where $E_{\min}$ and $E_{\max}$ have been defined in the trap profiling model of [2]. For (c) and (f), V_{gs} has been swept uniformly in [0.2 V, 0.8 V]. Here, τ is measured in seconds, $R(\tau)$ in A^2, all frequencies are in Hz, and all spectral densities are in A^2/Hz.

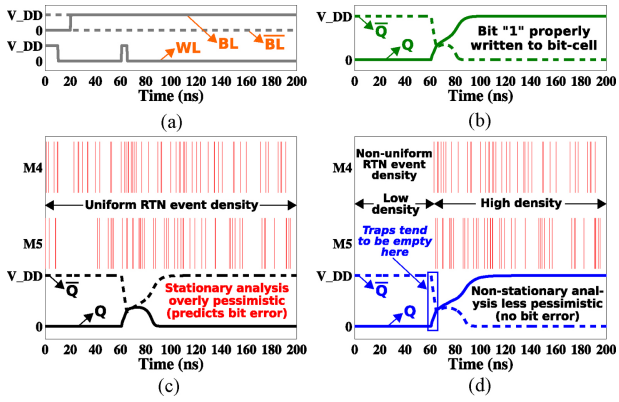


Fig. 8. SAMURAI versus stationary analysis, for a 22-nm 6T SRAM cell. Stationary analysis, being overly pessimistic, predicts an RTN-induced SRAM write failure even when there is none. SAMURAI, under the same conditions, does not predict any such failure. (a) SRAM cell inputs. (b) SRAM cell outputs: no RTN. (c) SRAM cell outputs: stationary RTN. (d) SRAM cell outputs: SAMURAI.

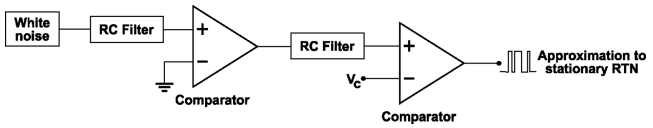


Fig. 9. Schematic proposed by Ye *et al.* [15] for producing traces that approximate stationary (constant bias) RTN.

VI. SAMURAI Versus Stationary RTN Analysis

It is well known [3], [5], [14] that stationary analysis often overestimates the impact of RTN, especially if circuit operation involves rapidly switching devices [3]. In particular, for SRAMs, we expect stationary methods to result in overly pessimistic predictions compared to a non-stationary technique such as SAMURAI. This is illustrated as follows.

Fig. 8 compares SAMURAI against stationary analysis, for a 22-nm (BSIM3) 6T SRAM bit-cell. Consider writing the bit "1" to this cell (which initially stores a "0"). Fig. 8(a) shows the input waveforms BL, $\overline{\text{BL}}$, and WL applied to write the "1." In the absence of RTN, the SPICE simulation of Fig. 8(b) shows that the bit is properly written (i.e., Q and $\overline{Q}$ settle to their desired values at the end of the write).

We now inject stationary RTN into the simulation, using traps in devices M4 and M5 [see Fig. 13(a)]. To model stationary traps, we set the capture and emission propensities to constant values (independent of gate bias). This leads to a uniform density of RTN events in each device, as seen from Fig. 8(c), which depicts each RTN event by a vertical bar. When this train of RTN events is included in the SPICE simulation, it results in a bit error [the waveforms of Fig. 8(c)]. This is largely because, under the stationary assumption, near $t = 60$ ns when the write operation begins, the traps in devices M4 and M5 have a reasonably good chance of being filled. The RTN produced by such filled traps slows down the devices, causing a write failure.

Fig. 8(d) applies SAMURAI to the same SRAM cell for the same inputs. In this case, the capture and emission propensities do depend on gate bias (the positions, energy levels and other parameters are the same as the previous stationary simulation). This produces a train of non-stationary RTN events (i.e., with a non-uniform event density) in each device, as seen from the vertical bars of Fig. 8(d). In particular, the time point near $t = 60$ ns, when the write operation begins, falls in a low density region. At this instant, the traps in M4 and M5 have a much greater chance of being empty than filled, which

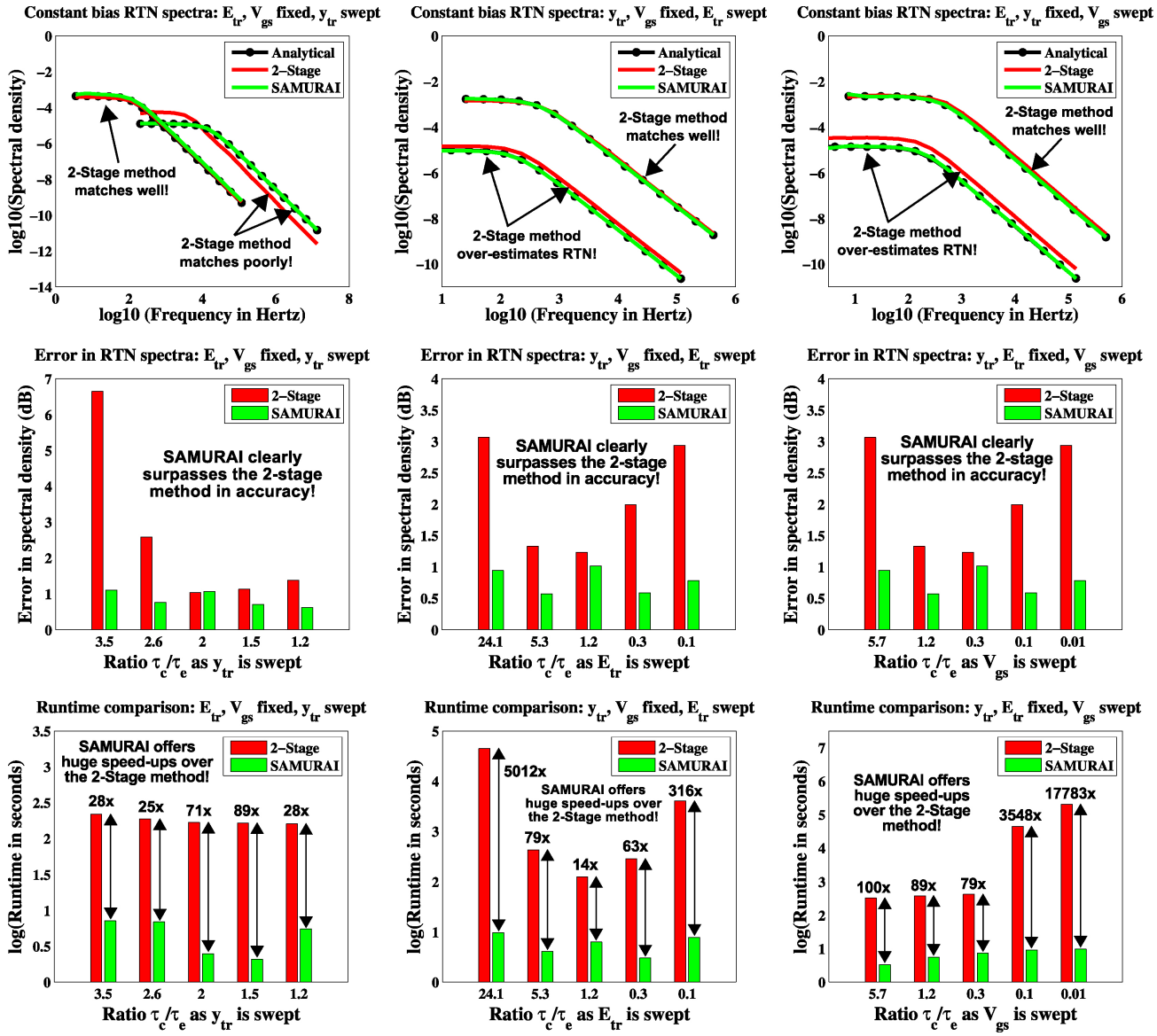


Fig. 10. Comparing SAMURAI against 2-stage RTN generation. The top three plots demonstrate that, even for stationary RTN, the 2-stage method, unlike SAMURAI, does not closely match analytical expressions. The middle three plots show that the 2-stage method can mispredict RTN by several dB, whereas SAMURAI is always within 1 dB of the true spectral density. The bottom three plots show that SAMURAI is much faster than 2-stage RTN generation (note that runtimes are plotted on a logarithmic scale).

makes an RTN-induced write failure very unlikely. This point is completely ignored by stationary analysis, but is captured by SAMURAI (for example, the waveforms of Fig. 8(d), which includes non-stationary RTN, do not show a bit error).

We note that the trap time constants used to generate Fig. 8 are in the nanosecond range, which is physically unrealistic (typical time constants tend to be much longer). We have used such small time constants mainly for convenient visual depiction, and to illustrate that SAMURAI makes more realistic predictions than stationary analysis under similar time constants [5]. Please see Section XII for a longer discussion.

VII. SAMURAI: Comparison Against 2-Stage RTN Generation

We now compare SAMURAI against a recently proposed approach [15] for RTN generation. This approach is based on the observation that white noise, on being RC-filtered, exhibits a spectrum similar to stationary RTN. From this, Ye *et al.* proposed a schematic (Fig. 9) to generate approximate RTN traces for a single trap under constant gate bias.

As Fig. 9 shows, each trap is simulated by a 2-stage "RC filter followed by comparator" structure driven by white noise (with the noise sources of different traps being independent of one another). The above circuit outputs a two-level signal, which is taken to be the trap occupancy function. The filter parameters, and the threshold V_c of the second comparator, are chosen depending on the stationary trap parameters τ_c and τ_e (reciprocals of λ_c and λ_e, respectively).

To compare SAMURAI against the above method, we perform the following steps.

1) We generate several trap configurations by varying V_{gs}, E_{tr}, and y_{tr} (as before, we fix two parameters at a time, and sweep the third). We compute τ_c and τ_e for each trap [2].

2) We start the timer. For each trap, we compute the parameters R, C, and V_c using [15], and SPICE-simulate the corresponding 2-stage circuit over time interval $[0, 1000(\tau_c+\tau_e)]$, with time step $0.1(\tau_c+\tau_e)$. We stop the timer and denote the elapsed time by $t_{2\text{-stage}}$.
3) We simulate each trap under constant bias, for the same time interval $[0, 1000(\tau_c+\tau_e)]$, using SAMURAI. We also time SAMURAI, letting $t_{SAMURAI}$ denote the elapsed time.
4) From the RTN traces returned by both methods, we numerically estimate the autocorrelations $R(\tau)$ and the PSDs $S(f)$ (similar to Section V).
5) We plot the above PSDs alongside analytical expressions (the top three plots of Fig. 10). We also plot the average dB error in the PSD, $|S_{predicted} - S_{analytical}|$ (the middle three plots of Fig. 10). Finally, we plot the runtimes $t_{2-stage}$ and $t_{SAMURAI}$ (the bottom three plots of Fig. 10).

While carrying out the above, we observed that for all trap configurations, SAMURAI closely matched the analytical expression, whereas the 2-stage method did not. Therefore, in each of the top three plots of Fig. 10, we have presented one scenario where the 2-stage method matched the analytical PSD well, and one where it does not match so well.

The middle three plots of Fig. 10 show that SAMURAI surpasses the 2-stage method in accuracy; whereas the 2-stage method can significantly mis-predict even stationary RTN (at times, by as much as 7 dB), the error in SAMURAI is always within about 1 dB (the theory of uniformization guarantees that SAMURAI is stochastically exact; however, a small error is expected because of numerical truncations, both in the simulation and in the autocorrelation/PSD estimation).

The bottom three plots of Fig. 10 highlight the speedups that SAMURAI offers over the 2-stage method. For every trap configuration that we tested, SAMURAI was many times faster than the 2-stage method; indeed, a logarithmic scale is needed for convenient visual depiction. In Fig. 10, the speed-ups are indicated alongside the log(runtime) bars for each trap. As the figure shows, SAMURAI can be four orders of magnitude faster than 2-stage RTN generation.

We note that the 2-stage method [15] spends the bulk of its time simulating white noise sources (one for each trap) in the time domain. This requires SPICE to take very small time-steps, even if the underlying circuit can tolerate much larger steps (as is indeed the case for the above experiments, since the gate voltages are constant). By contrast, SAMURAI does not require time-consuming time-domain white noise simulation, which makes it much faster than the 2-stage method.

VIII. MUSTARD: A CAD Tool for Bi-Directionally Coupled RTN Simulation

Having presented and validated an algorithm for unidirectional RTN, we now extend it to the bi-directionally coupled case, which forms the core of MUSTARD.

A. Stochastic Simulation of Bi-Directionally Coupled Markov/DAE Systems by Uniformization

Algorithm 2 describes MUSTARD's strategy for generating non-stationary RTN at the circuit level (assuming [2] that the sum $\lambda_c+\lambda_e$ is constant for each device trap). Similar to SAMURAI, the algorithm first uniformizes the time-inhomogeneous RTN Markov chain into a stationary high-rate chain, which is then simulated using Gillespie's algorithm [27]. Again like SAMURAI, later in the simulation, a probabilistic decision discards some of the generated RTN events, which restores the non-stationarity of the original RTN hypercube.

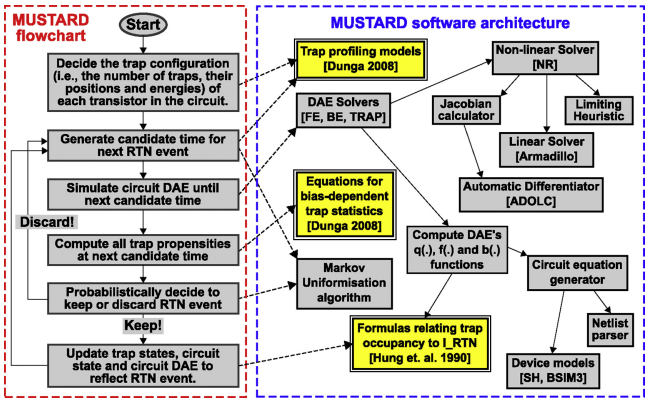


Fig. 11. (a) Flowchart describes Algorithm 2, while the (b) dependence graph illustrates its implementation in MUSTARD. An arrow from u to v indicates that the module u depends on the module v.

The crucial difference between MUSTARD (Algorithm 2) and SAMURAI (Algorithm 1) is that: MUSTARD ensures that the effects of individual RTN events on the circuit's DAE are incorporated in a bi-directionally coupled manner, i.e., the DAE is updated as soon as the RTN events occur (lines 19 and 20). This enables MUSTARD to overcome the main limitation of SAMURAI. Moreover, MUSTARD and SAMURAI both have the same runtime, so MUSTARD's improved accuracy is achieved at no extra computational cost.

B. MUSTARD's Software Architecture

We intend to release MUSTARD as an open-source tool for RTN analysis. To encourage early adoption, we have implemented a modular, easily extensible, easily maintainable architecture for MUSTARD (Fig. 11). To integrate RTN simulation more efficiently with circuit simulation, we have implemented much of the circuit simulation functionality from scratch. Also, the RTN-related modules (indicated using double-bordered boxes) are maintained separately from the rest of the simulator, making it easy to experiment with different trap configurations, statistical parameters, I_{RTN} equations, etc.

IX. MUSTARD Versus SAMURAI

We now illustrate the differences between SAMURAI and MUSTARD, in terms of bit-error predictions using a 22-nm (BSIM3) 6T SRAM cell. Because SAMURAI does not consider "second-order effects" (i.e., how each RTN event affects the statistics of subsequent events), its predictions are valid only as long as the circuit's voltages and currents, in the presence of RTN, are approximately equal to the nominal voltages and currents. For SRAMs, this holds only until the first read or write failure. After that, the voltages in the presence of RTN differ greatly from the nominal voltages, which may render SAMURAI's predictions invalid. MUSTARD, however, continually updates the circuit

Algorithm 2: Circuit simulation with non-stationary RTN in MUSTARD

Input: Circuit DAE D, initial circuit and trap states at time t_0, final time t_f
Output: Circuit simulation trace with realistic, non-stationary RTN in time $[t_0, t_f]$

```
// uniformise the RTN Markov chain to a high-rate λ*
λ* = 0;
foreach trap tr in the circuit do λ*+ = λ_{c,tr}(t0) + λ_{e,tr}(t0);
t_curr = t0; x_curr = x0; tr_curr = tr0;
while t_curr < tf do
    // generate a candidate time for the next RTN event
    t_next_RTN = t_curr + exprand(1/λ*);
    // simulate the circuit's DAE D until t_next_RTN
    while t_curr < min(tf, t_next_RTN) do
        record(t_curr, x_curr, tr_curr);
        t_next = min(t_curr + t_step, t_next_RTN, tf);
        x_next = LMSSolve(D, t_curr, x_curr, t_next);
        // LMSSolve can be any standard DAE solution
           method
        // e.g., Forward Euler, Backward Euler,
           Trapezoidal
        t_curr = t_next; x_curr = x_next;
    end
    // time to make a probabilistic decision
    // to either keep or discard the candidate RTN event
    if t_curr < tf then
        u = rand(); // u is uniformly distributed in [0, 1]
        sum = 0;
        foreach trap tr in the circuit do
            // compute propensity of trap tr to change
               state
            // detailed stochastic models available for
               this
            // by default, MUSTARD uses [2]
            λ[tr] = tr_curr[tr] ? λ_{e,tr}(x_curr) : λ_{c,tr}(x_curr);
            if u ≥ sum/λ* AND u < (sum + λ[tr])/λ* then
                // Keep the RTN event: trap tr changes
                   state!
                tr_curr[tr] = !(tr_curr[tr]);
                // update circuit's DAE to reflect RTN
                   event
                // many literature models available for
                   this
                // by default, MUSTARD uses [19]
                x_curr = new_ckt_state_after_RTN_event_at_tr;
                D = new_ckt_DAE_after_RTN_event_at_tr;
                break;
            end
            sum += λ[tr];
        end
    end
end
```

equations to reflect the higher-order effects of bi-directionally coupled RTN, so its predictions are always valid.

Fig. 12 illustrates the above, for a 22-nm 6T SRAM cell. Fig. 12(a) depicts the inputs to the bit-cell (to write the bit sequence "011"). In the absence of RTN, the SPICE simulation of Fig. 12(b) shows that this sequence is written properly.

We now use SAMURAI to generate (non-stationary) RTN, using nominal voltages/currents from the simulation of Fig. 12(b). This produces a train of RTN events, with a non-uniform event density, in each device [the vertical bars of Fig. 12(c)]. However, when these events are included in the simulation, a bit error results at $t \approx 96$ ns [Fig. 12(c)]. After this, the nominal Q is close to V_{DD}, but the Q in the presence of RTN is close to 0 V. Therefore, SAMURAI's predictions are not valid after $t = 96$ ns. MUSTARD, however, fully takes the first bit error into account for generating subsequent RTN events. So MUSTARD's predictions eventually differ materially from those of SAMURAI. For example, in Fig. 12(c),

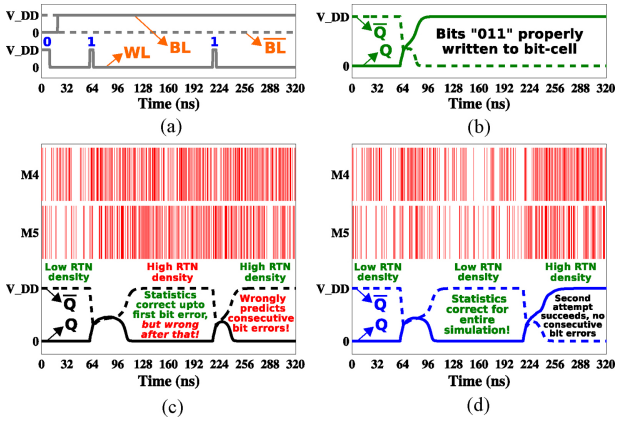


Fig. 12. MUSTARD versus SAMURAI. Until the first bit error, SAMURAI and MUSTARD make similar predictions. But after that, SAMURAI makes less reliable predictions than MUSTARD. (a) SRAM cell inputs. (b) SRAM cell outputs: no RTN. (c) SRAM cell outputs: SAMURAI. (d) SRAM cell outputs: MUSTARD.

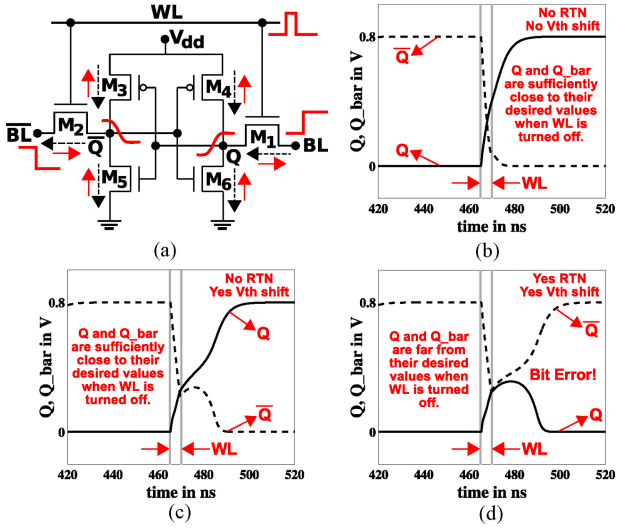


Fig. 13. (a) Writing a "1" to the 6T SRAM cell. Dashed (solid) arrows indicate the direction of I_d (I_{RTN}) in each transistor. (b)–(d) RTN, coming on the top of a 100 mV V_{th} shift due to parameter variations, can produce an SRAM write error.

SAMURAI wrongly predicts that the second attempt at writing a "1" would also fail, whereas MUSTARD correctly predicts a successful second write attempt [Fig. 12(d)].

We note that the trap time constants used to generate Fig. 12 are in the nanosecond range, which is physically unrealistic because typical time-constants tend to be much longer. We have used such small time-constants mainly for convenient visual depiction, and to illustrate that MUSTARD makes more reliable predictions than SAMURAI under similar assumptions about time constants. Please see Section XII for a longer discussion.

X. Applications to SRAM Design

We now apply Algorithm 2 to conduct bi-directionally coupled, non-stationary analysis of RTN in 22-nm SRAMs.

A. Prediction of RTN-Induced SRAM Write Failures

RTN-induced write failures have been experimentally observed in deep submicron SRAMs [4]. To reproduce these, we

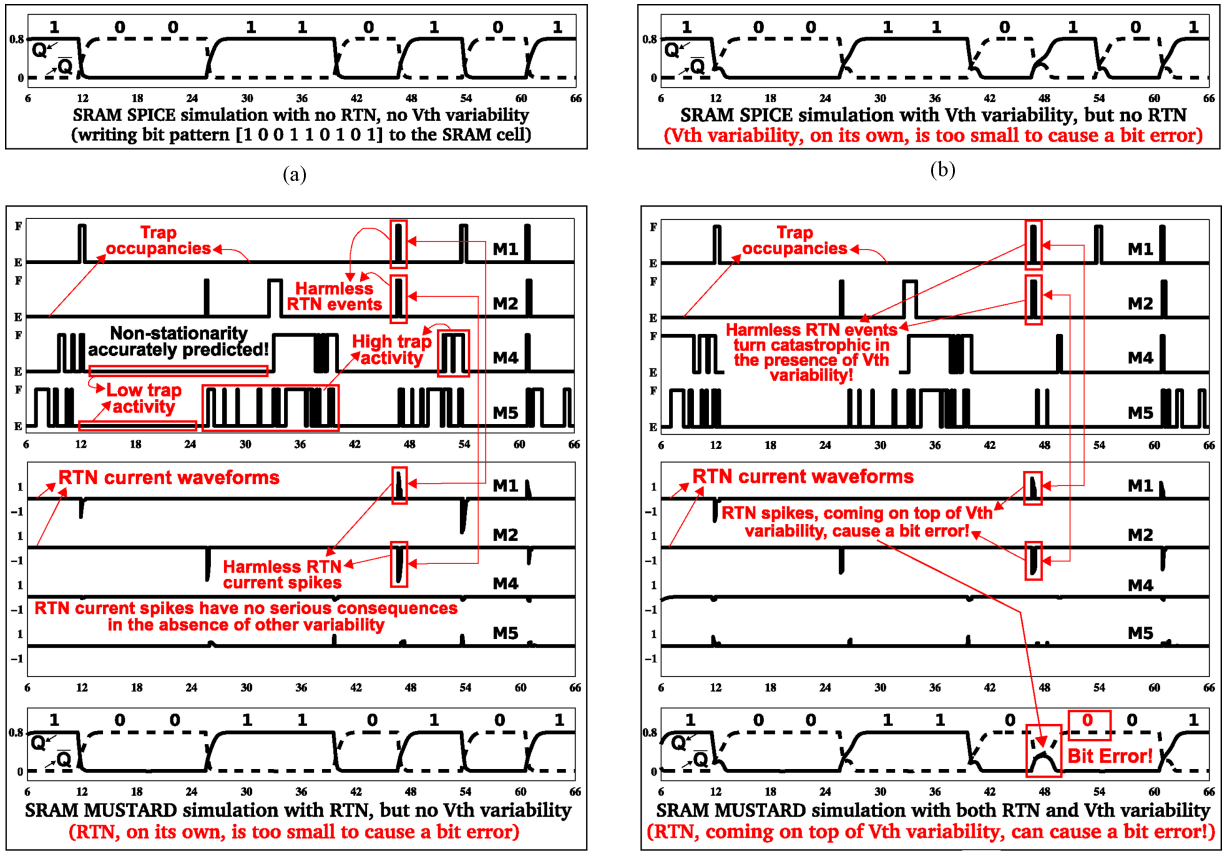


Fig. 14. Examining MUSTARD's simulation trace (including trap occupancies and RTN currents) during the clock cycles prior to the write failure of Fig. 13. Parts (a) and (b) show that, in the absence of RTN, the bit-cell is robust to write failure (even with significant V_{th} variability). Part (c) shows that RTN events are harmless in the absence of a V_{th} shift, while part (d) pinpoints the specific RTN events that were responsible for the write failure depicted in Fig. 13. In all plots, the x-axis denotes time (in ns). For the $Q/\overline{Q}$ plots, the y-axis denotes voltage (in volts). For trap occupancy plots, the y-axis is discrete, with E meaning *empty* and F meaning *filled*. In the I_{RTN} plots, the y-axis denotes current in μA.

designed a 22-nm 6T SRAM cell (using the BSIM3 model, with parameters obtained from [28]) and studied its non-stationary RTN using Algorithm 2.

Fig. 13 illustrates an RTN-induced failure predicted by MUSTARD for the above bit-cell. Part (a) shows the biases under which this failure occurs. The expected directions of I_d (dashed arrows) and I_{RTN} (solid arrows) are also indicated next to each transistor. If there is no RTN and no V_{th} shift, we see from Fig. 13(b) that Q properly settles to logical 1 (or V_{dd}).

We now introduce a 100 mV shift (to model parameter variability) to the V_{th} of pass transistors M1 and M2. Even so, the SRAM cell, in the absence of RTN, latches on to the correct value of Q by the end of the clock cycle [Fig. 13(c)].

Now we bring in RTN, by injecting one trap each into devices M1, M2, M4 and M5. As a result, the SRAM cell is no longer able to respond by the end of the clock cycle [Fig. 13(d)]. This is because RTN currents in M1 and M4 (M2 and M5) oppose the nominal currents driving Q ($\overline{Q}$) to 1 (0), sufficient to cause a bit error. Thus, we have used MUSTARD to reproduce results previously obtainable only by measurement.

In addition, MUSTARD offers significant "debuggability" advantages over pure measurement. For example, now that a bit error has been discovered, the entire simulation trace of RTN events leading up to the bit error can be examined (Fig. 14). From this, it is possible to precisely pinpoint which RTN events triggered the failure [Fig. 14(d)].

Fig. 14(a) and (b) shows simulations, without RTN, of the SRAM cell with and without V_{th} shifts. Fig. 14(c) shows that RTN current spikes that occur in the absence of V_{th} shifts are unable to cause bit errors. However, in the presence of V_{th} shifts, similar spikes do produce bit errors [Fig. 14(d)]. Indeed, it is clear that the RTN events of M1 and M2 (pointed out in the figure), which produce RTN spikes (also pointed out in the figure) just as the SRAM cell is switching from 0 to 1, must have been responsible for this particular bit error.

As before, we note that the trap time-constants used to generate Fig. 13 are physically unrealistic, and that we have used them mainly for convenient visual depiction, and to illustrate the kinds of predictions that our tools are capable of making. For more realistic results, it is necessary to use technology-specific trap time-constants, which are likely to be much larger, and as a result, require longer transient simulations. Please see Section XII for a longer discussion.

B. Statistical Inferences About RTN-Induced SRAM Failures

SRAM arrays typically contain thousands of cells, spanning a wide range of V_{th} values and trap populations. To ascertain the impact of RTN on such circuits, we performed a large number of MUSTARD simulations.

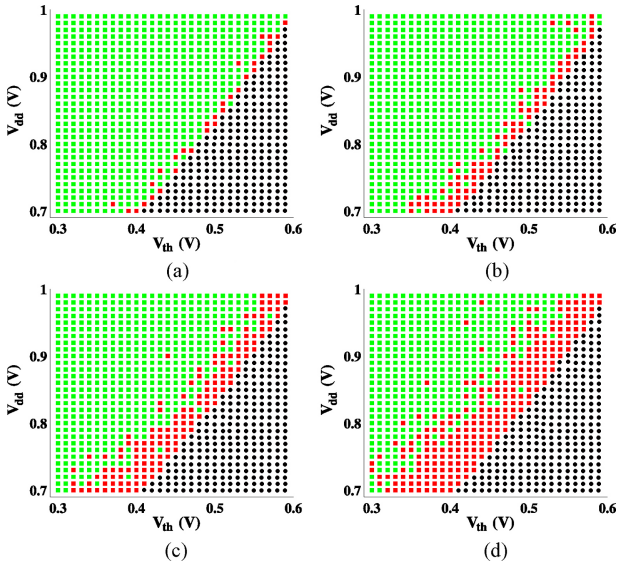


Fig. 15. MUSTARD simulations of RTN in 22-nm SRAM cells. For each ($V_{\rm th}$, V_{dd}, N) triple, a few hundred random N-traps-per-transistor configurations are sampled and MUSTARD-simulated over a random bit pattern. The circles indicate a bit error even without RTN. The dark squares indicate a bit error with RTN, which would not have occurred if RTN had been absent. The light squares indicate robust bit cells (no bit errors even with RTN). (a) N = 2. (b) N = 4. (c) N = 6. (d) N = 8.

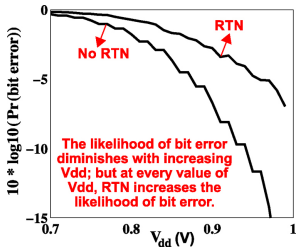


Fig. 16. Bit error probabilities as a function of V_{dd}. For each V_{dd}, several SRAM cells, with randomly distributed trap configurations and $V_{\rm th}$ values, are sampled and MUSTARD-simulated. $V_{\rm th}$ is sampled from a normal distribution, while trap configurations are sampled using the model proposed in [2].

Fig. 15 shows plots generated by sweeping the pass transistors' V_{th} and the supply voltage V_{dd}, for various RTN strengths (i.e., two, four, six, and eight traps per device). As the figure shows, the (V_{th}, V_{dd}) space is roughly banded into three regions: 1) a region with circles, containing bit errors even without RTN; 2) a dark-squared region containing bit errors with RTN, which would have been absent without RTN; and 3) a light-squared region containing no bit errors even with RTN. The dark-squared region, therefore, isolates the contribution of RTN toward reducing the SRAM design margin. As expected, this region becomes bigger as the number of traps increases. On average, the effect of RTN seems equivalent to a V_{th} shift of about 0.02V to 0.06V, which tallies with measured data [4].[6]

Fig. 16 quantifies the bit-error impact of RTN on SRAM arrays. Using a normal distribution for V_{th} and the trap profiling model of [2], we have computed the probability of write failure as V_{dd} is increased from 0.7 V to 1.0 V. As expected, the failure probability decreases with increasing V_{dd}, whether or not RTN is present. However, in the presence of RTN, the bit error probability diminishes with V_{dd} at a reduced rate, leading to a higher failure probability at *every* V_{dd}.

We note that, even though the number of active device traps may average only 1 to 2 [10], [12], [13], the actual number is often a Poisson random variable [2], [12], which occasionally assumes a value much greater than average. For example, consider the Intel Core i7 processors that have 8 MB of shared L3 (SRAM) cache memory. Of the 8 million bit-cells in this SRAM, at least a few hundred cells are likely to have much higher trap counts than average.[7] Such "high-trap-count" bit-cells are particularly vulnerable to RTN, and can break the error correcting code of the SRAM. Thus, it is necessary to simulate a large number of such vulnerable bit-cells (analogous to importance sampling), to determine the process/trap corners at which RTN-induced failures can occur (we have done this, using MUSTARD, in Fig. 15). Although such process/trap corners seem unlikely, they must be accurately accounted for, because they can degrade the 5σ to 6σ yield that SRAMs typically need. For example, the failure rates in Fig. 16 correspond to yields much lower than 5σ, indicating that significant re-design is required.

XI. Applications to DRAM Design

We now apply MUSTARD to study the impact of RTN on DRAM refresh time. Fig. 17(a) shows how the stored value Q of a 22-nm DRAM cell evolves with time as the bit "1" is written to it.[8] The figure shows that, whether or not a V_{th} shift is present, RTN always has some impact on the stored *analog* value Q of a DRAM cell. For the same simulation, Fig. 17(b) shows the number of filled traps as a function of time, while Fig. 17(c) shows the RTN currents $I_{RTN}(t)$ [whose directions are along the solid arrow below device M1 of Fig. 17(a)]. In all four cases, it is seen that $I_{RTN}(t)$ starts at 0, attains a peak value and tapers off towards the end of the write. This can be explained as follows. In the beginning, the transistor is off, so there is no RTN and the traps are likely to be empty [3] (because their emission propensities are much higher than their capture propensities). As the gate voltage is increased, the traps start demonstrating activity (because their capture and emission propensities are now comparable), which leads to increased RTN current. As Q rises further, the propensities are still comparable, but the nominal current I_d becomes smaller and smaller. Therefore, even though trap activity continues [as seen from Fig. 17(b)], the waveform modulating the trap activity becomes small, thereby causing RTN to taper off.

Again, we note that the trap time-constants used to generate Fig. 17 are physically unrealistic, and that we have used them mainly for convenient visual depiction, and to illustrate the kinds of predictions that our tools are capable of making. In particular, while the explanations above help to understand the

[6]Please note that N in Fig. 15 is the total number of device traps, not the number of active traps. For example, even though eight traps may exist per device (for $N = 8$), only one to two traps may be active at each bias point, consistent with measured data. As the simulation progresses, SAMURAI and MUSTARD both automatically keep track of which traps are active at each bias.

[7]For instance, assuming a Poisson mean of 1, about 1 in 16 000 bit cells is expected to have at least one device with ≥ 8 active traps.

[8]Please note that, while writing a "1" to a DRAM cell, WL has to be held at $V_{DD} + V_{\rm th}$ for Q to approach V_{DD}; otherwise, if both WL and BL are held at V_{DD}, then Q will only reach $V_{DD} - V_{th}$, before the device gets switched off.

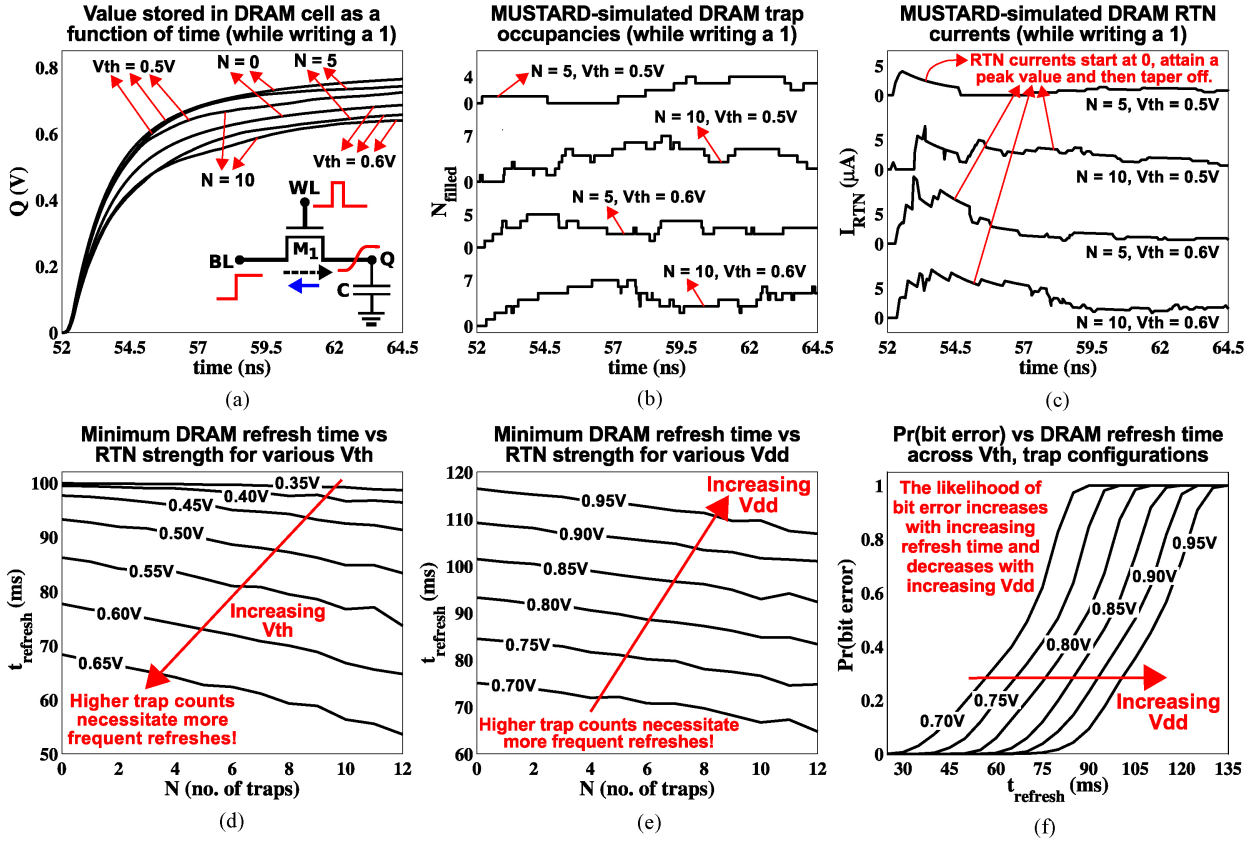


Fig. 17. RTN analysis of DRAMs using MUSTARD. (a)–(c) Plots showing that MUSTARD is able to simulate non-stationary, coupled RTN within a DRAM cell [circuit shown in (a)]. (d)–(f) Plots of statistical results obtained by MUSTARD-simulating hundreds of DRAM cells, with different threshold voltages and trap configurations.

predictions made by MUSTARD under the small time-constant assumption, it is necessary to use (much longer) technology-specific time-constants for a more realistic RTN analysis.

Fig. 17(d)–(f) shows the impact of RTN on DRAM refresh time, an important figure-of-merit that characterizes how long a DRAM cell can retain a stored value (before leakage currents eventually corrupt it). Fig. 17(d) shows that a DRAM cell with higher V_{th} needs to be refreshed more often. For two DRAM cells with the same V_{th}, the one with higher trap count needs to be refreshed more often (because, on average, the increased RTN would weaken its stored value to a greater extent). Fig. 17(e) shows that the refresh frequency can be reduced as V_{dd} increases. However, more traps necessitate more frequent refreshes. Fig. 17(f) plots the probability of a DRAM bit-error against refresh frequency; this plot was generated by MUSTARD-simulating hundreds of DRAM cells.

XII. Discussion

Here, we discuss two important issues: 1) the unusually fast traps used in this paper, and 2) using SAMURAI/MUSTARD when the sum $\lambda_c + \lambda_e$ is bias dependent.

The unusually fast traps: We note that most RTN simulations in this paper use traps with time constants smaller than experimentally reported data. This enabled us to generate SRAM/DRAM failures within relatively short simulation runs that could be visually depicted in this paper. Also, the shortened time constants illustrate SAMURAI/MUSTARD's ability to perform accelerated RTN testing, a much needed feature [1], [4]. The motivation is that, if traps transition only once every millisecond or so, it would take prohibitively long measurements or simulations to produce statistical failure estimates for circuits clocked in the nanosecond range, like SRAMs. We overcome this difficulty by speeding up the traps, to make their time constants comparable to the circuit's operating frequency. This ensures that all trap states can be covered during shorter simulation runs, from which circuit failure probabilities can be quickly estimated.[9]

An adaptive λ^* extension for SAMURAI/MUSTARD: Algorithms 1 and 2 both require a λ^* that is an upper bound on the Markov propensities (Section III) for all time. While the tightness of the upper bound does not affect the correctness of either SAMURAI or MUSTARD, the simulation becomes more efficient as the bound's tightness improves (since fewer events will be discarded). For the RTN model of [2] (which assumes a constant $\lambda_c + \lambda_e$), it is straightforward to compute λ^* at time 0, and prove its validity for all time (Algorithms 1 and 2). However, such a "pre-computable upper bound" is not necessary for SAMURAI/MUSTARD to work. For example, an anonymous reviewer brought [29] to our attention, which reports trap behaviors that do not obey the assumptions of [2]. For such traps, it may be difficult to compute a valid λ^* at time 0. So, we have developed an extension of MUSTARD (Algorithm 3) that learns λ^* adaptively. The extended algorithm eliminates the need to select λ^* at the beginning; instead, λ^* is changed "on the fly" during the course of the simulation.

[9]Of course, both SAMURAI and MUSTARD can handle arbitrary time constants, and can run without acceleration if desired. The resulting simulations, however, would require more time.

Algorithm 3: Extending MUSTARD's core technique to adaptively learn the upper bound λ^*

```
Input: Circuit DAE D, initial circuit and trap states at time t_0, final time t_f
Output: Circuit simulation trace with realistic, non-stationary RTN in time [t_0, t_f]
t_curr = t_0; x_curr = x_0; tr_curr = tr_0;
while t_curr < t_f do
    // guess a value for λ*
    λ*_guess = guess_lambda_star(set of all traps tr in the circuit, x_curr, tr_curr);
    // take snapshot of current state
    // to revert back in case λ*_guess is found to be invalid
    t_prev = t_curr; x_prev = x_curr;
    // generate a candidate time for the next RTN event
    t_next_RTN = t_curr + exprand(1/λ*_guess);
    // simulate the circuit's DAE D until t_next_RTN
    while t_curr < min(t_f, t_next_RTN) do
        record(t_curr, x_curr, tr_curr);
        t_next = min(t_curr + t_step, t_next_RTN, t_f);
        x_next = LMSSolve(D, t_curr, x_curr, t_next);
        // LMSSolve can be any standard DAE solution method
        // e.g., Forward Euler, Backward Euler, Trapezoidal
        t_curr = t_next; x_curr = x_next;
        // check if λ*_guess is valid
        λ*_check = 0;
        foreach trap tr in the circuit do λ*_check += (tr_curr[tr] ? λ_e,tr(x_curr) : λ_c,tr(x_curr));
        if λ*_check > λ*_guess then
            // λ*_guess is too small, increase it
            λ*_guess *= 1 + η(x_curr, tr_curr);
            // start again from t_prev with new candidate RTN event
            clear_records(t_prev);
            t_curr = t_prev; x_curr = x_prev;
            t_next_RTN = t_curr + exprand(1/λ*_guess);
        end
    end
    // time to make a probabilistic decision
    // to either keep or discard the candidate RTN event
    if t_curr < t_f then
        u = rand(); // u is uniformly distributed in [0, 1]
        sum = 0;
        foreach trap tr in the circuit do
            // compute propensity of trap tr to change state
            λ[tr] = tr_curr[tr] ? λ_e,tr(x_curr) : λ_c,tr(x_curr);
            if u ≥ sum/λ*_guess AND u < (sum + λ[tr])/λ*_guess then
                // Keep the RTN event: trap tr changes state!
                tr_curr[tr] = !(tr_curr[tr]);
                // update circuit's DAE to reflect RTN event
                x_curr = new_ckt_state_after_RTN_event_at_tr;
                D = new_ckt_DAE_after_RTN_event_at_tr;
                break;
            end
            sum += λ[tr];
        end
    end
end
```

Algorithm 3 works by guessing an upper bound λ^*_{guess}, which needs to be valid only until the next candidate RTN event. As the simulation progresses, the algorithm keeps track of the time-varying propensities, and at each time-step, checks whether the guessed λ^* is valid. If, at any time, the guess becomes invalid, the algorithm reverts back to the time of the previous event, and restarts the simulation from there with an improved guess. We note that an initial guess that is too high would considerably impair the algorithm's efficiency. To avoid this, it is important to have the guess_lambda_star() routine make an aggressive (low) guess. Another solution would be to continuously monitor the fraction of discarded RTN events, and reduce λ^*_{guess} if necessary.

XIII. Conclusions and Future Work

In this paper, we presented two techniques, SAMURAI and MUSTARD, for circuit-level non-stationary RTN analysis. While SAMURAI is interoperable with existing SPICE simulators, MUSTARD sacrifices interoperability for the ability to analyze bi-directionally coupled RTN. Both tools are highly generic; they work with 1) any circuit design (e.g., 6T/9T SRAMs, 1T/3T DRAMs), 2) any device model (e.g., BSIM, PSP), and 3) any RTN model (e.g., number/mobility fluctuation, with any number of traps per device, any number of them being active). We used MUSTARD to duplicate experimentally observed RTN-induced SRAM failures, and variable DRAM retention times. We were also able to generate statistical characterizations of RTN-induced SRAM/DRAM failures, in the presence of variability.

References

[1] Y. Tsukamoto, S. O. Toh, C. Shin, A. Mairena, T. J. K. Liu, and B. Nikolic, "Analysis of the relationship between random telegraph signal and negative bias temperature instability," in *Proc. IEEE Intl. Reliability Phys. Symp.*, May 2010, pp. 1117–1121.

[2] M. V. Dunga, "Nanoscale CMOS modeling," Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Univ. California, Berkeley, 2008 [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-20.html

[3] H. Tian and A. El Gamal, "Analysis of 1/f noise in switched MOSFET circuits," *IEEE Trans. Circuits Systems II Analog Digital Signal Process.*, vol. 48, no. 2, pp. 151–157, Feb. 2001.

[4] S. O. Toh, Y. Tsukamoto, Z. Guo, L. Jones, T. J. K. Liu, and B. Nikolic, "Impact of random telegraph signals on Vmin in 45 nm SRAM," in *Proc. IEEE IEDM*, Dec. 2009, pp. 767–770.

[5] J. S. Kolhatkar, "Steady-state and cyclo-stationary RTS noise in MOSFETS," Ph.D. dissertation, Dept. Electr. Eng., Math. Comput. Sci., Univ. Twente, Twente, The Netherlands, 2005.

[6] T. Umeda, K. Okonogi, K. Ohyu, S. Tsukada, K. Hamada, S. Fujieda, and Y. Mochizuki, "Single silicon vacancy-oxygen complex defect and variable retention time phenomenon in DRAMs," *Appl. Phys. Lett.*, vol. 88, no. 25, p. 253504, 2006.

[7] N. Tega, H. Miki, Z. Ren, C. P. D'Emic, Y. Zhu, D. J. Frank, J. Cai, M. A. Guillorn, D. G. Park, W. Haensch, and K. Torii, "Reduction of random telegraph noise in high-K metal-gate-stacks for 22 nm generation FETs," in *Proc. IEEE IEDM*, Dec. 2009, pp. 771–774.

[8] M. J. Kirton and M. J. Uren, "Noise in solid-state microstructures: A new perspective on individual defects, interface states and low-frequency 1/f noise," *Advances Phys.*, vol. 38, no. 4, pp. 367–468, 1989.

[9] A. van der Ziel, *Noise in Solid State Devices and Circuits*. New York: Wiley Interscience, 1976.

[10] S. Lee, H. J. Cho, Y. Son, D. S. Lee, and H. Shin, "Characterization of oxide traps leading to RTN in high-K and metal gate MOSFETS," in *Proc. IEEE IEDM*, Dec. 2009, pp. 763–766.

[11] G. I. Wirth, J. Koh, R. da Silva, R. Thewes, and R. Brederlow, "Modeling of statistical low frequency noise of deep submicron MOSFETS," *IEEE Trans. Electron Devices*, vol. 52, no. 7, pp. 1576–1588, Jul. 2005.

[12] T. Nagumo, K. Takeuchi, S. Yokogawa, K. Imai, and Y. Hayashi, "New analysis methods for comprehensive understanding of random telegraph noise," in *Proc. IEEE IEDM*, Dec. 2009, pp. 759–762.

[13] N. Tega, H. Miki, F. Pagette, D. J. Frank, A. Ray, M. J. Rooks, W. Haensch, and K. Torii, "Increasing threshold voltage variation due to random telegraph noise in FETs as gate lengths scale to 20 nm," in *Proc. Symp. VLSI Technol.*, 2009, pp. 50–51.

[14] A. S. Roy and C. C. Enz, "Analytical modeling of large-signal cyclo-stationary low-frequency noise with arbitrary periodic input," *IEEE Trans. Electron Devices*, vol. 54, no. 9, pp. 2537–2545, Sep. 2007.

[15] Y. Ye, C. C. Wang, and Y. Cao, "Simulation of random telegraph noise with 2-stage equivalent circuit," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 2010, pp. 709–713.

[16] K. V. Aadithya, A. Demir, S. Venugopalan, and J. Roychowdhury, "SAMURAI: An accurate method for modelling and simulating non-stationary random telegraph noise in SRAMs," in *Proc. Design Autom. Test Conf. Eur.*, Mar. 2011, pp. 1113–1118.

[17] K. V. Aadithya, A. Demir, S. Venugopalan, and J. Roychowdhury, "MUSTARD: A coupled, stochastic/deterministic, discrete/continuous technique for predicting the impact of random telegraph noise on SRAMs and DRAMs," in *Proc. Design Autom. Conf.*, Jun. 2011, pp. 292–297.

[18] A. van der Ziel, "Unified presentation of 1/f noise in electron devices: Fundamental 1/f noise sources," *Proc. IEEE*, vol. 76, no. 3, pp. 233–258, Mar. 1988.
[19] K. K. Hung, P. K. Ko, C. Hu, and Y. C. Cheng, "Random telegraph noise of deep-submicrometer MOSFETs," *IEEE Electron Device Lett.*, vol. 11, no. 2, pp. 90–92, Feb. 1990.
[20] K. K. Hung, P. K. Ko, C. Hu, and Y. C. Cheng, "A physics-based MOSFET noise model for circuit simulators," *IEEE Trans. Electron Devices*, vol. 37, no. 5, pp. 1323–1333, May 1990.
[21] J. Roychowdhury, "Numerical simulation and modeling of electronic and biochemical systems," *Found. Trends Electron. Design Autom.*, vol. 3, nos. 2–3, pp. 97–303, 2009.
[22] P. Heidelberger and D. M. Nicol, "Conservative parallel simulation of continuous time Markov chains using uniformization," *IEEE Trans. Parallel Distributed Syst.*, vol. 4, no. 8, pp. 906–921, Aug. 1993.
[23] A. P. A. van Moorsel and K. Wolter, "Numerical solution of nonhomogeneous Markov processes through uniformization," in *Proc. 12th Eur. Multiconference Simulation*, 1998, pp. 710–717.
[24] J. G. Shanthikumar, "Uniformization and hybrid simulation/analytic models of renewal processes," *Oper. Res.*, vol. 34, no. 4, pp. 573–580, 1986.
[25] [Online]. Available: http://www.spiceopus.si/.
[26] A. Papoulis, S. U. Pillai, and S. Unnikrishna, *Probability, Random Variables, and Stochastic Processes*. New York: McGraw-Hill, 2002.
[27] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *J. Comput. Phys.*, vol. 22, no. 4, pp. 403–434, 1976.
[28] [Online]. Available: http://ptm.asu.edu/modelcard/HP/22nm_HP.pm
[29] H. Miki, M. Yamaoka, N. Tega, Z. Ren, M. Kobayashi, C. P. D'Emic, Y. Zhu, D. J. Frank, M. A. Guillorn, D. Park, W. Haensch, and K. Torii, "Understanding short-term BTI behavior through comprehensive observation of gate-voltage dependence of RTN in highly scaled high-K/metal-gate pFETs," in *Proc. Symp. VLSI Technol.*, 2011, pp. 148–149.

Karthik V. Aadithya received the Bachelors degree in electrical engineering from the Indian Institute of Technology Madras, Chennai, India, in 2009.

He is currently a Graduate Student with the Donald O. Pederson Center for Electronic Systems Design, University of California, Berkeley. He works with Prof. Roychowdhury on developing computer-aided design tools to analyze the circuit-level impact of device-level variability or noise. His current research interests include modeling, analysis, and simulation of analog effects in current and future generation analog, digital, and mixed-signal circuits.

Alper Demir (F'12) received the B.S. degree in electrical engineering from Bilkent University, Ankara, Turkey, in 1991, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1994 and 1997, respectively.

He was with Motorola, Austin, TX, in 1995, with Cadence Design Systems, San Jose, CA, in 1996, with Bell Laboratories Research, Murray Hill, NJ, from 1997 to 2000, with CeLight, Iselin, NJ (a start-up company in optical communications), from 2000 to 2002, with the Research Laboratory for Electronics, Massachusetts Institute of Technology, Cambridge, in 2002 and August 2005, and with the University of California, Berkeley, from September 2009 to September 2010, as a Visiting Professor. He was with the Department of Electrical and Electronics Engineering, Koç University, Istanbul, Turkey, as an Assistant Professor from February 2002 to December 2007, and has been an Associate Professor since January 2008. His work at Bell Laboratories and CeLight is the subject of six patents. He has co-authored two books in the areas of nonlinear-noise analysis and analog-design methodologies, and has published about 50 articles in journals and conferences. His current research interests include computational prototyping of electronic and optoelectronic systems, numerical modeling and analysis, stochastic dynamical systems, and noise in nonlinear electronic, optical, communication and biological systems.

Dr. Demir is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF CIRCUITS AND SYSTEMS. He was the recipient of several Best Paper Awards, including the 2002 Best of International Conference on Computer-Aided Design (ICCAD) Award: 20 Years of Excellence in CAD, the 2003 IEEE/ACM William J. McCalla ICCAD Best Paper Award, and the 2004 IEEE Circuits and Systems Society Guillemin-Cauer Best Paper Award. In 1991, he was the recipient of the Regents Fellowship and the Eugene-Mona Fay Gee Scholarship from the University of California, Berkeley, and was selected to be an Honorary Fellow of the Scientific and Technological Research Council of Turkey (TUBITAK). In 2003, he was selected by the Turkish Academy of Sciences to receive the Distinguished Young Scientist Award, in 2005, he won a TUBITAK Career Award, and in 2007, he received the TUBITAK Young Scientist Award. In 2009, he was awarded the 2219 Research Fellowship by TUBITAK for his sabbatical year at Berkeley.

Sriramkumar Venugopalan received the B.Tech. degree in electrical engineering (EE) from the Indian Institute of Technology Kanpur, Kanpur, India, in 2008, and the M.Sc. degree in EE from the University of California, Berkeley (UC Berkeley), in 2009. He is currently pursuing the Ph.D. degree with the BSIM Group, UC Berkeley, working under the guidance of Prof. Hu who is a TSMC Distinguished Professor with the Graduate School.

He was a Research Intern with IMEC, Leuven, Belgium, in 2007, Texas Instruments, Inc., Dallas, in 2011, and GlobalFoundries, Inc., Milpitas, CA, in 2012. He has been involved in the research and development of turnkey SPICE compact models for FinFETs, trigate, gate-all-around transistors (BSIM-CMG), UTB-SOI, UTBB-SOI, and independent multigate transistors (BSIM-IMG), and BSIM6 (a radio frequency design relevant update to BSIM4). His current research interests include semiconductor device physics and technology-design interaction.

Mr. Venugopalan was a recipient of a number of awards, including the TSMC Outstanding Student Research Award in 2011, the Frank and Lucas Margaret Fellowship, UC Berkeley, from 2008 to 2009, and the Caltech Summer Undergraduate Research Fellowship in 2006.

Jaijeet Roychowdhury (F'09) received the Bachelors degree in electrical engineering from the Indian Institute of Technology Kanpur, Kanpur, India, in 1987, and the Ph.D. degree in electrical engineering and computer science (EECS) from the University of California, Berkeley (UC Berkeley), in 1993.

He is currently a Professor of EECS with UC Berkeley. From 1993 to 1995, he was with the Computer-Aided Design (CAD) Laboratory, AT&T Bell Laboratories, Allentown, PA. From 1995 to 2000, he was with the Communication Sciences Research Division, Bell Laboratories, Murray Hill, NJ. From 2000 to 2001, he was with CeLight, Inc. (an optical networking startup), Silver Spring, MD. From 2001 to 2008, he was with the Department of Electrical and Computer Engineering and the Digital Technology Center, University of Minnesota, Minneapolis. He was cited for extraordinary achievement by Bell Laboratories in 1996. Over the years, he has authored or co-authored seven best or distinguished papers at ASP-DAC, DAC, and ICCAD. His current research interests include the analysis, simulation, and design of electronic, biological, and mixed-domain systems.

Dr. Roychowdhury was an IEEE Circuits and Systems Society Distinguished Lecturer from 2003 to 2005 and served as the Program Chair of IEEE's CANDE and BMAS Workshops in 2005. He has served on the Technical Program Committees of ICCAD, DAC, DATE, ASP-DAC, and other EDA conferences, on the Executive Committee of ICCAD, and on the Nominations and Appointments Committee of CEDA. He has been an Officer of CANDE.