Recursive Statistical Blockade: An Enhanced Technique for Rare Event Simulation with Application to SRAM Circuit Design

Amith Singhee[1], Jiajing Wang[2], Benton H. Calhoun[2], Rob A. Rutenbar[1]

[1]Dept. of ECE, Carnegie Mellon University,
Pittsburgh, PA 15213, USA
{asinghee,rutenbar}@ece.cmu.edu

[2]ECE Dept. University of Virginia,
Charlottesville, VA 22903, USA
{jjwang,bcalhoun}@virginia.edu

Abstract

Circuit reliability under statistical process variation is an area of growing concern. For highly replicated circuits such as SRAMs and flip flops, a rare statistical event for one circuit may induce a not-so-rare system failure. The authors of [1] proposed Statistical Blockade as a Monte Carlo technique that allows us to efficiently filter—to *block*—unwanted samples insufficiently rare in the tail distributions we seek. However, there are significant practical problems with the technique. In this work, we show common scenarios in SRAM design where these problems render Statistical Blockade ineffective. We then propose significant extensions to make Statistical Blockade practically usable in these common scenarios. We show speedups of 10^2+ over standard Statistical Blockade and 10^4+ over standard Monte Carlo, for an SRAM cell in an industrial 90nm technology.

1. Introduction

Circuit reliability under statistical process variation is an area of growing concern. Designs that add excess safety margin, or rely on simplistic assumptions about "worst case" corners no longer suffice. Worse, for critical circuits such as SRAMs and flip flops, replicated across 10K - 10M instances on a large design, we have the new problem that statistically rare events are magnified by the sheer number of these elements. In such scenarios, an exceedingly rare event for one circuit may induce a not-so-rare failure for the entire system.

Monte Carlo analysis (MC) [2] remains the gold standard for the required statistical modeling. Standard Monte Carlo techniques are, by construction, most efficient at sampling the statistically likely cases. When used for simulating statistically unlikely or rare events, these techniques are extremely slow. For example, to simulate a 5σ event, 100 million circuit simulations would be required, on average.

One avenue of attack is to abandon Monte Carlo. Several analytical and semi-analytical approaches have been suggested to model the behavior of SRAM cells [3][4][5] and digital circuits [6] in the presence of process variations. All suffer from approximations necessary to make the problem tractable. [4] and [6] assume a linear relationship between the statistical variables and the performance metrics (*e.g.* static noise margin), and assume that the statistical process parameters and resulting performance metrics are normally distributed. This can result in gross errors, especially while modeling rare events, as we shall show later. When the distribution varies significantly from Gaussian, [4] chooses an F-distribution in an *ad hoc* manner. [3] presents a complex analytical model limited to a specific transistor model (the transregional model) and further limited to only static noise margin analysis for the 6T SRAM cell. [5] again models only the static noise margin (SNM) for SRAM cells under assumptions of independence and identical distribution of the upper and lower SNM, which may not always be valid.

A different avenue of attack is to modify the Monte Carlo strategy. [7] shows how Importance Sampling can be used to predict failure probabilities. Recently, [8] applied an efficient formulation of these ideas for modeling rare failure events for single 6T SRAM cells, based on the concept of *Mixture Importance Sampling* from [9]. The approach uses real SPICE simulations with no approximating equations. However, the method only estimates the exceedence probability of a *single* value of the performance metric. A re-run is needed to obtain probability estimates for another value. No complete model of the tail of the distribution is computed. The method also combines all performance metrics to compute a failure probability, given *fixed* thresholds. Hence, there is no way to obtain separate probability estimates for each metric, other than a separate run per metric. Furthermore, given that [7] advises against importance sampling in high dimensions, it is unclear if this approach will scale efficiently to large circuits with many statistical parameters.

The authors of [1] presented Statistical Blockade (SB), a general and efficient MC method that addresses both problems previously described: very fast generation of samples—rare events—with sound models of the tail statistics for any performance metric. The method imposes almost no *a priori* limitations on the form of the statistics for the process parameters, device models, or performance metrics. The key observation behind Statistical Blockade is that *generating* each sample is not expensive: we are merely creating the parameters for a circuit.

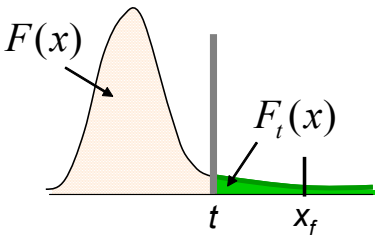

FIGURE 1. Example distribution of a circuit performance metric (e.g., SRAM write time). The solid region is the *tail* region. SB focuses the sampling to this region for fast sampling of rare events.

Evaluating the sample is expensive, because we simulate it. The paper developed a method to quickly *filter* these samples, and *block* those that are unlikely to fall in the low-probability tails we seek. It used techniques from *data mining* [10] to build *classifier* structures, from a small set of Monte Carlo training samples, to create the necessary blocking filter. Given these samples, it showed how to use the rigorous mathematics of *Extreme Value Theory* (EVT [11]) to build sound models of these tail distributions. The paper successfully applied SB to a variety of circuits with dimensionality ranging up to 403, with speedups of up to 2 orders of magnitude over Standard Monte Carlo.

SB can, however, completely fail for certain commonly seen SRAM metrics (e.g., data retention voltage) because of the presence of conditionals in the formulation of the metric. Also, if rare samples with extremely low probability (e.g. 5σ and beyond) are required, SB can still become prohibitively expensive. In this work, we extend the SB technique in two significant ways: 1) we propose a solution to solve the problem of SB failing for certain common SRAM metrics, and 2) we develop a recursive strategy to achieve further speedups of orders of magnitude, while simulating extremely rare events (5σ and beyond).

This paper is organized as follows. Section 2 reviews the Statistical Blockade filtering technique from [1]. Section 3 presents the first problem with the formulation, circuit metrics with conditionals, and proposes a solution. Section 4 presents the second problem with SB, sampling extremely rare events, and proposes a solution. Section 5 presents experimental results and Section 6 offers concluding remarks.

2. Background: Statistical Blockade Filtering

Fig. 1 shows an example distribution $F(x)$ of a circuit metric; e.g., SRAM write time. As an example, consider a 1Mb cache, where the SRAM cell has a failure probability of 1ppm, given a failure threshold, x_f. In such a case we would need to simulate 1 million MC samples to generate one such failure event and made any prediction about the failure probability. In fact, we would need many more to generate sufficient failure events to ensure statistical confidence of the prediction. This approach would become much worse for lower failure

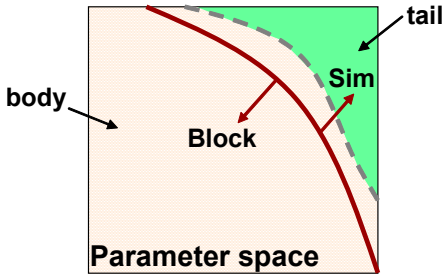

FIGURE 2. The classifier in statistical parameter space is shown as a solid boundary. The dashed line is the exact tail region boundary. The relaxed classification boundary allows us to *block* most non-tail points.

probabilities. This scenario is common in today's SRAM designs.

SB was proposed in [1] to significantly speed up the simulation of rare events and prediction of low failure probabilities. SB defines a *tail threshold* (for example, the 99% point) t, as shown in Fig. 1. Without loss of generality, the part of the distribution greater than t is called the tail. The key idea is to identify that region in the parameter (process variable) space that yields circuit performance values (e.g., SRAM write time) greater than t. Once this is known, those MC samples that do not lie in this tail region are not simulated, or *blocked*. Only those MC samples that lie in the tail region are simulated. Hence, the number of simulations can be significantly reduced. For example, if t is the 99-th percentile, only 1% of the MC samples will be simulated, resulting in an immediate speedup of 100x over standard MC.

To build this model of the boundary of the tail region a small MC sample set (1,000 points) is used to train a classifier. A classifier is an indicator function that allows us to determine set membership for complex, high-dimensional, nonlinear data. Given a data point, the classifier reports true or false on the membership of this point in some arbitrary set. For Statistical Blockade, this is the set of parameter values *not* in the tail region we seek. However, it is difficult, if not impossible, to build an exact model of the tail region boundary. Hence, we relax the requirement to allow for classification error. This is

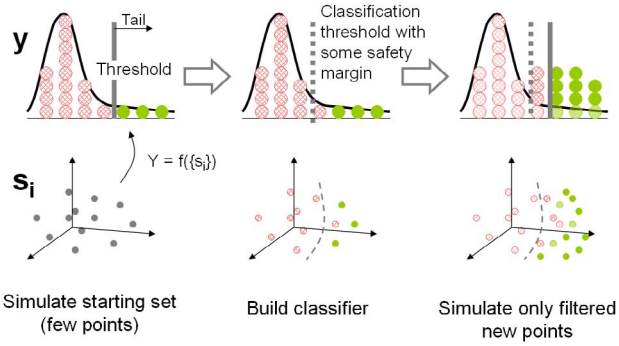

FIGURE 3. Classification based sampling

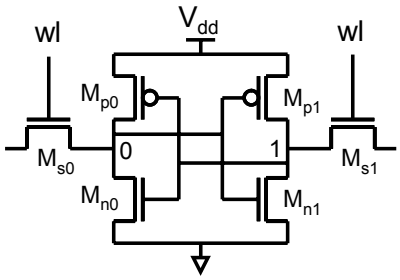

FIGURE 4. A standard 6-T SRAM cell.

done by building the classification boundary at a *classification threshold* t_c that is less than the tail threshold t. Fig. 2 shows this relaxed classification boundary in the parameter space. The dashed line is the exact boundary of the tail region for the tail threshold t, and the solid line is the relaxed classification boundary for the classification threshold t_c.

SB filtering is then accomplished in three steps (Fig. 3):

1) *Perform initial sampling* to generate data to build a classifier. This initial sampling can be standard Monte Carlo or importance sampling.
2) *Build a classifier* C using a classification threshold t_c. To minimize false negatives (tail points classified as non-tail points), choose $t_c < t$.
3) *Generate more samples* using MC, following the CDF F, but simulate only those that are classified as tail points.

From the simulated samples, some will be in the tail region and some will be in the non-tail region. [1] shows how to use Extreme Value Theory to fit a parametric distribution (the Generalized Pareto Distribution) to these tail points to generate an analytical model for the failure probability, given any failure threshold $x_f > t$.

In the rest of this paper, we will focus on the classifier building and the sample filtering parts of this framework. We will show how they can fail for certain common scenarios and present effective solutions.

3. Classifier failure: Conditionals

3.1 The problem

Consider the 6-T SRAM cell shown in Fig. 4. With scaling reaching nanometer feature sizes, subthreshold and gate leakage become very significant. Particularly for the large memory blocks seen today, the standby power consumption due to leakage can be intolerably high. Supply voltage (V_{dd}) scaling [12] is a powerful technique to reduce this leakage, whereby the supply voltage is reduced when the memory bank is not being accessed. However, lowering V_{dd} also makes the cell unstable, ultimately resulting in data loss at some threshold value of V_{dd}, known as *Data Retention Voltage* or DRV. Hence, DRV is the lowest supply voltage that still preserves

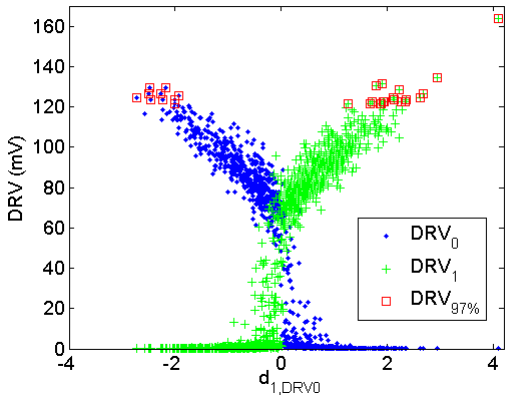

FIGURE 5. Behavior of DRV_0 and DRV_1 along the direction of maximum variation in DRV_0. The worst 3% DRV values are shown as squares, clearly showing the disjoint tail regions (along this direction in the parameter space).

the data stored in the cell. DRV is computed as follows.

$$DRV = max(DRV_0, DRV_1) \quad (1)$$

where DRV_0 is the DRV when the cell is storing a 0, and DRV_1 is the DRV when it is storing a 1. If the cell is balanced (symmetric), then $DRV_0 = DRV_1$. However, if there is any mismatch due to process variations, they become unequal. This creates a situation where the standard SB classification technique would fail. We will explain this in more detail now.

Suppose we run a 1,000 sample MC, varying all the mismatch parameters in the SRAM cell according to their statistical distributions. This would give us distributions of values for DRV_0, DRV_1 and DRV. In certain parts of the mismatch parameter space $DRV_0 > DRV_1$, and in other parts $DRV_0 < DRV_1$. This is clearly illustrated in Fig. 5. Using SiLVR, from [13], we extracted the direction in the parameter space that has maximum impact on DRV_0 (maximum variation), called *latent variable* in the paper. The figure plots the simulated DRV_0 and DRV_1 values along this direction ($d_{1, DRV0}$). We can clearly see that they are inversely related: one decreases as the other increases. Now let us take the *max* as in (1), and choose the classification threshold t_c for DRV as the 97-th percentile. Then we pick out the worst 3% points from the classifier training data and plot them against the same

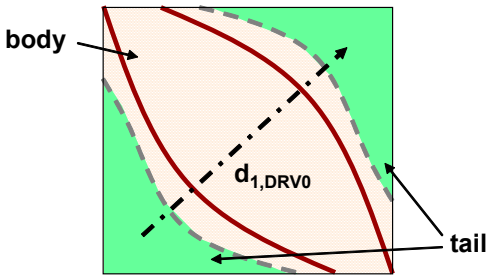

FIGURE 6. Parameter space with two disjoint tail regions for the same circuit metric (e.g. DRV).

direction $d_{1,DRV0}$, in Fig. 5. We can clearly see that these points (squares) lie in two disjoint parts of the parameter space. Since the tail region defined by a tail threshold $t > t_c$ would be a subset of the classifier tail region (defined by t_c), it is obvious that the tail region consists of two disjoint regions of the parameter space. This is illustrated with a 2-D example in Fig. 6. The figure also shows the direction vector for $d_{1,DRV0}$. The solid tail regions on the top-right and bottom-left corners of the parameter space correspond to the large DRV values shown as squares in Fig. 5.

In such a situation the SB classifier is unable to create a single boundary to separate the tail and non-tail regions. The problem stems from the *max* operation in (1), since it combines subsets of the tail regions of DRV_0 and DRV_1 to generate the tail region of DRV. The same problem occurs for any other such metric (e.g., Static Noise Margin) with a conditional operation. We now propose a solution to this problem.

3.2 Solution

Instead of building a single classifier for the tail of DRV in (1), we will build two separate classifiers, one for the the 97-th percentile ($t_c(DRV_0)$) of $DRV0$, and another for the 97-th percentile ($t_c(DRV_1)$) of DRV_1. The generated MC samples will then be filtered through both these classifiers: points classified as non-tail by *both* the classifiers will be blocked, and the rest will be simulated. In the general case, if the circuit metric y is given as

$$y = max(y_0, y_1, \ldots) \quad (2)$$

the resulting algorithm is as follows.

1) Perform initial sampling to generate data to build a classifier and estimate tail and classification thresholds.
2) For each argument y_i of the conditional (2), build a classifier C_i at a classification threshold $t_c(y_i)$ that is less than the tail threshold $t(y_i)$.
3) Generate more samples using MC, but block the samples classified as *non-tail* by *all* the classifiers. Simulate the rest and compute y for the simulated points.

Hence, in the case of Fig. 6, we build a separate classifier for each of the two boundaries. From the simulated points, those with $y > t$ are chosen as tail points for further analysis [1]. Also note that this same algorithm can be used for the case of multiple metrics. Each metric would have its own thresholds and its own classifier, just like each argument in (2).

4. Simulating Extremely Rare Events

4.1 The problem

Consider a 10 Mb memory, with no redundancy or error correction. Even if the failure probability of each cell is as low as 0.1 ppm, every such chip will still fail on average. Hence, the worst case (largest) DRV from a 10 million MC should, on average, be below the standby voltage. To estimate this at least 10 million MC samples have to be run. If we want to reduce the *chip* failure probability to less than 1%, we need to look at the worst case DRV from a 1 billion MC run. This is equivalent, approximately, to the 6σ value of DRV -- the 6σ point from a standard normal distribution has the same failure probability. Using Statistical Blockade, we can reduce the number of samples, using a classification threshold t_c = 97-th percentile. This would reduce the number of simulations from 1 billion to 30 million, which is still very large. Even with a perfect classifier, where we can choose $t_c = t$ = 99-th percentile, the number of simulations would still be 10 million. Moving t_c to higher percentiles will help reduce this further, but many more initial samples will be needed for a believable estimate of t_c and for training the classifier. Now we describe a recursive formulation that reduces the simulation count drastically.

```
1.for each argument y_i in y = max(y_i)
2.   n = n_0 = 1000
3.   y_{i,tail} = Simulate(MCarlo(n))
4.   while (n < N)
5.      y_{i,tail} = GetWorst(n_0, y_{i,tail})
6.      t = Percentile(y_{i,tail}, 99)
7.      t_c = Percentile(y_{i,tail}, 97)
8.      C = BuildClassifier(y_{i,tail}, t_c)
9.      n = n * 100
10.     y_{i,tail} = Simulate(Filter(C, MCarlo(n)))
11. end
12. y_tail = max(y_{0,tail}, y_{1,tail}, ...)
```

FIGURE 7. A recursive formulation for Statistical Blockade for simulating extremely rare events, that can also handle conditionals.

4.2 Solution

Let us first assume that there are no conditionals. For a tail threshold equal to the α-th percentile, let us represent it as t^α, and the corresponding classification threshold as t_c^α. Using the algorithm from Section 3.2, build a classifier C^α and generate sufficient points with $y > t^\alpha$, so that a higher percentile (t^β, t_c^β, $\beta > \alpha$) can be estimated. For this new, higher threshold a new classifier C^β is trained and a new set of tail points ($y > t^\beta$) are generated. This new classifier will block many more points than C^α, significantly reducing the number of simulations. This procedure is repeated to push the threshold out more till the tail region of interest is reached. The complete algorithm is shown in Fig. 7.

Now, we will describe the functions used in the algorithm. In line 1, we repeat lines 2-10 for each argument of the conditional. If there is no conditional, lines 2-10 are not repeated. The conditional *max* is used without loss of generality. N is the total number of MC samples that would be needed to reach the tail regions required; e.g., N = 1 billion for reaching 6σ. The function MCarlo(n) generates n samples, and the function Simulate() actually simulates the samples passed to it.

The returned vector consists of both the input parameter sets for simulation and the corresponding circuit metrics computed for each sample. The function GetWorst(n_0, $\boldsymbol{x}$) returns the n_0 worst samples from the set $\boldsymbol{x}$. BuildClassifier($\boldsymbol{x}$, t_c) builds a classifier using training points $\boldsymbol{x}$. Hence in line 8, C is a classifier. The function Filter(C, $\boldsymbol{x}$) blocks the samples in $\boldsymbol{x}$ classified as non-tail by C and returns the samples classified as tail points. The function Percentile($\boldsymbol{x}$, p) computes the p-th percentile of the output values in the set $\boldsymbol{x}$.

The basic idea is to use a tail threshold (and its corresponding classification threshold) that is very far out in the tail, so that the simulations are restricted to the very rare events we are interested in. This is being done in a recursive manner by estimating lower thresholds first and using them to estimate the higher threshold without having to simulate a large number of points. For example, if we want to use the 99.9999 percentile as the tail threshold $t^{99.9999}$, we first estimate the 99.99 percentile threshold $t^{99.99}$. To estimate this in turn, we first estimate the 99 percentile threshold t^{99}. At each stage we use a classifier corresponding to that threshold to reduce the number of simulations for estimating the next higher threshold. The next section will present experimental results.

5. Experimental Results

The techniques described in this paper were applied to a standard 6T SRAM cell, for the case of DRV. The cell was implemented in an industrial 90nm process and all the mismatch statistical parameters were varied as per the industrial process design kit (PDK). We used a Support Vector Machine classifier [14], similar to [1].

5.1 An analytical model for DRV

The authors in [15] develop an analytical model for predicting the Cumulative Density Function (CDF) of the DRV, that uses not more than 5,000 MC simulations. The CDF is given as

$$F_{DRV}(x) = 1 - erfc\left(\frac{\mu_0 + k(x - V_0)}{\sqrt{2}\sigma_0}\right) + \frac{1}{4}\left(erfc\left(\frac{\mu_0 + k(x - V_0)}{\sqrt{2}\sigma_0}\right)\right)^2 \quad (3)$$

where x is the DRV value. k is the sensitivity of DRV to the supply voltage, computed using a DC sweep. μ_0 and σ_0 are the mean and standard deviation of the Static Noise Margin distribution for the circuit, for a user-defined supply voltage V_0. These are computed using a short Monte Carlo run. Complete details regarding this analytical model are provided in [15]. The q-th quantile (e.g., the 6σ point) can be estimated as

$$DRV(q) = \frac{1}{k}(\sqrt{2}\sigma_0 erfc^{-1}(2 - 2\sqrt{q}) - \mu_0) + V_0 \quad (4)$$

Hence, $DRV(q)$ is the supply voltage V_{dd} such that

$$P(DRV \le V_{dd}) = q \quad (5)$$

We compare the worst case DRV values from our technique, for a given number of MC samples, with the value predicted by (5) for the corresponding quantile. For example, we can compute the 4.5σ DRV value from (5) and compare it with the worst case DRV from a 1 million sample MC run: 1 ppm is the failure probability of the 4.5σ point.

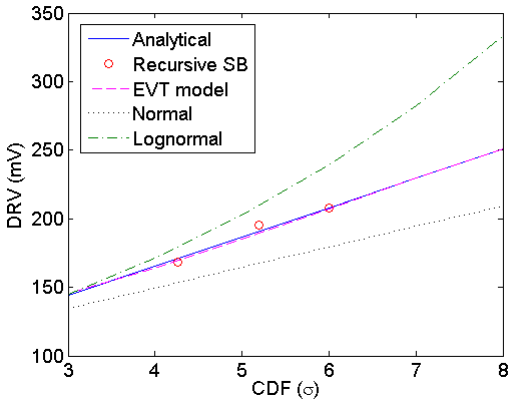

FIGURE 7. The worst case DRV values from RSB closely match the model in (5). Fitting an EVT model as per [1] to the data from RSB also shows close match with (5). Normal and log-normal fits are inaccurate.

5.2 Results

Fig. 7 shows a graphical comparison of five different methods:

1) **Analytical**: The 3σ to 8σ DRV values (quantiles) predicted by equation (5).
2) **Recursive SB**: The algorithm in Fig. 7 was run for N = 1 billion: lines 5-10 were run three times, corresponding to 100,000, 10 million and 1 billion MC samples, respectively. The worst case DRV from these 3 recursion stages are estimates of the 4.26σ, 5.2σ and 6σ points, respectively.
3) **EVT model**: The tail points from the last recursion stage (1 billion MC) are used to fit a Generalized Pareto Distribution (GPD), as per [1]. This GPD is then used to predict the 3σ to 8σ DRV values.
4) **Normal**: A normal distribution is fit to data from a 1,000 sample MC run, and used to predict the same DRV values.
5) **Lognormal**: A lognormal distribution is fit to the same 1,000 MC samples, and used for prediction.

From the plots, we can immediately see that Recursive SB estimates are very close to the estimates from the analytical model. Table 1 shows the number of circuit simulations performed at each of the three recursion stages, along with the initial 1,000 sample MC run. The total number of simulations used is a very comfortable 41,721, resulting in a speedup of 4 orders of magnitude over standard MC and 700 times over SB.

Also, we can extend the prediction power to 8σ without any additional simulations, by using the GPD model. Standard MC would need over 1.5 quadrillion points to generate an 8σ

point. For this case the speedup over standard MC is extremely large. The normal and lognormal fits show significant error compared to the analytical model. The normal fit is unable to capture the skewness of the actual DRV distribution, while the lognormal distribution has a heavier tail than the true DRV distribution and, hence, over-estimates the skewness.

Stage	Num. simulations
Init	1,000
1	11,032
2	14,184
3	15,505
Total	**41,721**
Speedup over MC	**23,969x**
Speedup over SB	**719x**

TABLE 1. Number of circuit simulations run per recursion stage to generate a 6σ DRV sample

A final point to highlight is that recursive SB is a completely general technique to estimate rare events and their tail distributions. In the case of the SRAM cell DRV experiment, we were lucky enough to have an extremely recent analytical result against which to compare performance. Obviously, if one has such analytical models available, one should use them. Unfortunately, in most cases, one does not, and one must fall back on some sort of Monte Carlo analysis. In such scenarios, recursive Statistical Blockade has three attractive advantages:

1) it is circuit-neutral, by which we mean that any circuit that can be simulated can be attacked with the technique;
2) it is metric-neutral, by which we mean that any circuit performance metric that can be simulated can be analyzed with the technique;
3) as seen in our SRAM DRV experiments, it is extremely efficient, faster usually by several orders of magnitude than simple-minded brute-force Monte Carlo algorithms.

6. Conclusions

Statistical Blockade was proposed in [1] for 1) efficiently generating samples in the tails of distributions of circuit performance metrics, and 2) deriving sound statistical models of these tails. However, the method has some practical shortcomings: it fails for the case of circuit metrics with conditionals, and it requires prohibitively large number of simulations while sampling extremely rare events. This paper presents a recursive formulation of SB that overcomes both these issues efficiently. This new technique was applied to an SRAM cell in an industrial 90nm technology to obtain speedups of up to 4 orders of magnitude over standard Monte Carlo and 2 orders of magnitude over standard SB.

Acknowledgements: The authors acknowledge the support of the Focus Center for Circuit & System Solutions (C2S2, http://www.c2s2.org), one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

References

[1] A. Singhee, R.A. Rutenbar, "Statistical Blockade: A Novel Method for Very Fast Monte Carlo Simulation of Rare Circuit Events, and its Application", *Proc. DATE*, 2007.
[2] G.S. Fishman, "A First Course in Monte Carlo", Duxbury Press, Oct. 2005.
[3] A.J. Bhavnagarwala, X. Tang, J.D. Meindl, "The Impact of Intrinsic Device Fluctuations on CMOS SRAM Cell Stability", *J.Solid State Circuits*, 26(4), pp 658-665, Apr. 2001.
[4] S. Mukhopadhyay, H. Mahmoodi, K. Roy, "Statistical Design and Optimization of SRAM Cell for Yield Enhancement", *Proc. ICCAD*, 2004.
[5] B.H. Calhoun, A. Chandrakasan, "Analyzing Static Noise Margin for Sub-threshold SRAM in 65nm CMOS", *Proc. ESSCIRC*, 2005.
[6] H. Mahmoodi, S. Mukhopadhyay, K. Roy, "Estimation of Delay Variations due to Random-Dopant Fluctuations in Nanoscale CMOS Circuits", *J. Solid State Circuits*, 40(3), pp 1787-1796, Sep. 2005.
[7] D.E. Hocevar, M.R. Lightner, T.N. Trick, "A Study of Variance Reduction Techniques for Estimating Circuit Yields', *IEEE Trans. CAD*, 2(3), July, 1983.
[8] R. Kanj, R. Joshi, S. Nassif, "Mixture Importance Sampling and its Application to the Analysis of SRAM Designs in the Presence of Rare Failure Events", *Proc. DAC*, 2006.
[9] T.C. Hesterberg, "Advances in Importance Sampling", PhD Dissertation, Dept. of Statistics, Stanford University, 1988, 2003.
[10] T. Hastie, R. Tibshirani, J. Friedman, "The Elements of Statistical Learning", Springer Verlag, 2003.
[11] A.J. McNeil, "Estimating the Tails of Loss Severity Distributions using Extreme Value Theory", *ASTIN Bulletin*, 27(1), pp 117-137, 1997.
[12] R. K. Krishnamurthy et al., "High-performance and low-power challenges for sub-70 nm microprocessor circuits," *Proc. CICC*, 2002.
[13] A. Singhee, R. A. Rutenbar, "Beyond Low-Order Statistical Response Surfaces: Latent Variable Regression for Efficient, Highly Nonlinear Fitting", *Proc. DAC*, 2007.
[14] T. Joachims, Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.
[15] J. Wang, A. Singhee, R.A. Rutenbar, B.H. Calhoun, "Statistical Modeling for the Minimum Standby Supply Voltage of a Full SRAM Array", *Proc. ESSCIRC*, 2007.