Analog Circuit Verification by Statistical Model Checking

Ying-Chih Wang*, Anvesh Komuravelli†, Paolo Zuliani† and Edmund M. Clarke†
* Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213
Email: yingchiw@ece.cmu.edu
† Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213
Email:{anvesh, pzuliani, emc}@cs.cmu.edu *

Abstract— We show how statistical Model Checking can be used for verifying properties of analog circuits. As integrated circuit technologies scale down, manufacturing variations in devices make analog designs behave like stochastic systems. The problem of verifying stochastic systems is often difficult because of their large state space. Statistical Model Checking can be an efficient verification technique for stochastic systems. In this paper, we use sequential statistical techniques and model checking to verify properties of analog circuits in both the temporal and the frequency domain. In particular, randomly sampled system traces are sequentially generated by SPICE and passed to a trace checker to determine whether they satisfy a given specification, until the desired statistical strength is achieved.

I. Introduction

As IC technologies move toward higher integration density with substantially small feature sizes, a greater concern of analog designers is circuit resiliency under heavy process variability. Since process variation brings stochastic effects into the model, the problem of verifying analog circuits is turned into the problem of verifying that a stochastic system satisfies a certain property. In the past few years there has been growing interest in the formal verification of stochastic systems by means of model checking techniques, see for example [1, 2, 3, 4, 5, 6] and [7]. The verification problem is to decide whether a stochastic model satisfies a property with a *probability* greater than or equal to a certain threshold.

In this work, we address the verification of both temporal (transient) properties and frequency-domain properties for analog circuits. The properties are expressed by a temporal logic in which the temporal operators are equipped with time *bounds*. For example, the property "the output voltage V_{out} will always stay below 5 volts in the next 80 time units" is written in our logic as $\mathbf{G}^{80}(V_{out} < 5)$. We ask whether a stochastic system $\mathcal{M}$ satisfies that formula with a probability greater than or equal to a fixed threshold (say 0.9999), and we write $\mathcal{M} \models Pr_{\geqslant 0.9999}[\mathbf{G}^{80}(V_{out} < 5)]$. This type of questions can be efficiently answered by *Statistical Model Checking* [3, 2, 5], the technique we use for verifying analog circuit models simulated by SPICE. In particular, randomly sampled system traces are sequentially generated using SPICE and passed to a trace checker (see Section II for a brief description of the checker) to determine whether they satisfy a given specification, until the desired statistical strength is achieved. We demonstrate the technique with a simple analog circuit (Section IV) – an operational amplifier – and we empirically show (Section V) that verification by statistical model checking is a feasible approach.

*This research was sponsored by the GSRC under contract no. 1041377 (Princeton University), National Science Foundation under contracts no. CNS0926181, no. CCF0541245, and no. CNS0931985, Semiconductor Research Corporation under contract no. 2005TJ1366, General Motors under contract no. GMCMUCRLNV301, Air Force (Vanderbilt University) under contract no. 18727S3, and the Office of Naval Research under award no. N000141010188.

II. Bounded Linear Temporal Logic

The Bounded Linear Temporal Logic (BLTL) is a variant of the well-known Linear Temporal Logic [8] in which temporal operators are bounded.

Let SV be a finite set of real-valued variables. An atomic proposition AP is a boolean predicate of the form $e_1 \sim e_2$, where e_1 and e_2 are arithmetic expressions over variables in SV, and the relational operator $\sim$ is either $<, \leq, >, \geq$ or $=$. A BLTL property is built over atomic propositions using boolean connectives and bounded temporal operators. The syntax of the logic is the following:

$$\phi ::= AP \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \phi_1 \mathbf{U}^t \phi_2.$$

The bounded until operator $\phi_1 \mathbf{U}^t \phi_2$ requires that, *within* time t, ϕ_2 will be true and ϕ_1 will hold until then. Bounded versions of the usual $\mathbf{F}$ and $\mathbf{G}$ operators can be easily defined: $\mathbf{F}^t\phi := true\ \mathbf{U}^t\phi$ requires ϕ to hold true within time t; $\mathbf{G}^t\phi := \neg\mathbf{F}^t\neg\phi$ requires ϕ to hold true up to time t.

The semantics of BLTL is defined with respect to *traces* (or executions) of a system. In our case, a trace is a sequence of signal values (*e.g.*, voltages) generated by a SPICE simulation of an analog circuit. Formally, a trace is a sequence of state transitions of the form $\sigma = (s_0, t_0), (s_1, t_1), ...$, which denotes that the system moved to state s_{i+1} after having sojourned for time (duration) t_i in state s_i. If trace σ satisfies BLTL property ϕ we write $\sigma \models \phi$. We denote the trace suffix starting at step k by σ^k. The satisfaction relation of BLTL is defined in terms of durations as follows [2].

978-1-4244-7516-2/11/$26.00 ©2011 IEEE

Definition. The semantics of BLTL on trace σ^k is:

- $\sigma^k \models AP$ iff AP holds true in state s_k;
- $\sigma^k \models \phi_1 \vee \phi_2$ iff $\sigma^k \models \phi_1$ or $\sigma^k \models \phi_2$;
- $\sigma^k \models \phi_1 \wedge \phi_2$ iff $\sigma^k \models \phi_1$ and $\sigma^k \models \phi_2$;
- $\sigma^k \models \neg\phi_1$ iff $\sigma^k \models \phi_1$ does not hold;
- $\sigma^k \models \phi_1 \mathbf{U}^t \phi_2$ iff there exists $i \geq 0$ such that,
  a) $\sum_{l=0}^{i-1} t_{k+l} \leq t$, and
  b) $\sigma^{k+i} \models \phi_2$, and
  c) for each $0 \leq j < i$, $\sigma^{k+j} \models \phi_1$.

It is easy to define an equivalent semantics with respect to time-stamped traces. Also, note that the semantics of BLTL is defined over *infinite* traces, while of course any simulation trace must be finite in length. It can be shown that traces of an appropriate (finite) length are sufficient to decide BLTL properties [2].

A. Monitoring the traces

In this section we describe briefly the algorithm we use to decide "$\sigma \models \phi$", which is also known as the *monitoring* problem. Our monitoring algorithm is designed to work online with the trace generation (after a possible integration with the generator). To be precise, the online nature of the algorithm obviates the necessity to store any portion of the trace for later use and implies the earliest termination of the trace monitoring.

The algorithm first builds a parse tree of ϕ, annotating the nodes of the tree representing the operator $\mathbf{U}^t$ with the time bound t and makes a copy of it for reference (call it the *reference* tree). Then, from the first observation on the trace, it assigns values to all the variables in SV appearing in ϕ, which are also the leaf nodes of the parse tree. These values are then propagated up the tree evaluating each node from the values of all its children. Now, a node may not obtain values from all the children during this propagation or if a node represents $\mathbf{U}^t$ it may need to see *ahead* in the trace, *i.e.,* wait for a future observation, to be evaluated. This is reflected in a *type* associated with each node. Thus, value propagations from children to parents will result in a change of type in the parents (and also, deletion of those children). Once all the possible propagations are done, the algorithm either terminates with a positive or negative answer if the root node is evaluated, and looks for the next observation on the trace otherwise.

A naïve approach to continue monitoring would be to use the *reference* tree to get the corresponding subtree for each of the nodes waiting for a future observation and make the roots of these subtrees children of these (respective) nodes (also, changing the time annotation of the roots to reflect the time duration of the previous observation). This maintains the tree structure and the algorithm looks at the next observation on the trace to evaluate the tree bottom-up as described before. But, the above approach can be very inefficient. To address that, we use an optimization to avoid redundant copies of *similar* nodes, which can be present in the different subtrees mentioned above, resulting in a DAG. The same bottom-up evaluation can be carried out except that a node can now have more than one parent and it has to propagate its value to all of them.

In short, the algorithm can be thought of as a space-efficient unrolling of an automaton of ϕ as the observations of the trace become available. When an accepting or a rejecting state is reached, the algorithm terminates with the answer.

The complexity of the algorithm also depends on the granularity of the system simulation (*i.e.*, the number of simulation steps to cover one real time unit.) If we consider only integral time-stamps (and thereby fix the granularity), the complexity is polynomial in the size of ϕ (which includes both its length and the time constants appearing in it) and linear in the theoretical maximum length of the finite prefix of the trace necessary for monitoring. The complete details of the algorithm and its analysis will be given in a forthcoming paper.

III. Statistical Model Checking

The verification problem for a stochastic system $\mathcal{M}$ can be phrased as follows: given a threshold $\theta \in (0, 1)$ and a BLTL formula ϕ, decide whether $\mathcal{M} \models Pr_{\geqslant\theta}(\phi)$; that is, decide whether $\mathcal{M}$ satisfies ϕ with probability greater than or equal to θ. Note that this problem is well-defined, since it can be shown that the set of traces of $\mathcal{M}$ satisfying ϕ is measurable, thereby defining the probability p that $\mathcal{M}$ satisfies ϕ [3].

Suppose now that the stochastic system $\mathcal{M}$ satisfies the BLTL formula ϕ with some (unknown) probability p. The key idea behind statistical model checking [3] is that the behavior of $\mathcal{M}$ (with respect to property ϕ) can be modeled by a Bernoulli random variable with success parameter p. This random variable can be repeatedly evaluated via system simulation in the following way. Let σ a trace of $\mathcal{M}$, then the Bernoulli random variable X with probability mass function:

$$f(x|p) = p^x(1-p)^{1-x} \quad x \in \{0, 1\} \tag{1}$$

denotes the outcome of $\sigma \models \phi$, (*i.e.*, model checking ϕ on σ). In other words, we have that:

$$X = \begin{cases} 1 & \text{with probability } p \quad (\sigma \models \phi), \\ 0 & \text{with probability } 1-p \quad (\sigma \models \neg\phi). \end{cases} \tag{2}$$

Therefore, by running a system simulation and by checking ϕ on the resulting trace we can obtain a sample from random variable X.

Statistical model checking approaches the verification problem as a statistical inference problem and solves it by randomized sampling of traces (simulations) from the model—by the paragraph above, this is equivalent to sampling from random variable X. The inference problem can then be solved by means of hypothesis testing [3, 9, 5] or estimation [4]. The former amounts to deciding between two hypotheses – $\mathcal{M} \models Pr_{\geqslant\theta}[\phi]$ versus $\mathcal{M} \models Pr_{<\theta}[\phi]$. The latter instead approximates probabilistically (that is, it computes with high probability an *estimate* close to) the true probability p that ϕ holds, and then compares that estimate with θ. In both approaches sampled traces are model checked individually to determine whether property ϕ holds, and the number of satisfying

traces is used by the hypothesis testing (or estimation) procedure to decide between $p \geqslant \theta$ and $p < \theta$. (In the case of estimation, one also has an estimate which is close to p with high probability.) Note that statistical model checking cannot guarantee a correct answer to the verification problem. However, the probability of giving a wrong answer can be made arbitrarily small.

Sequential Bayesian hypothesis testing and estimation have been recently introduced and applied to the verification of stochastic hybrid systems coded as Stateflow/Simulink models [2]. We now briefly describe both techniques.

A. Bayesian Hypothesis Testing

In hypothesis testing we decide between a null hypothesis H_0 and an alternative hypothesis H_1:

$$H_0 : p \geqslant \theta \qquad H_1 : p < \theta \quad . \tag{3}$$

The Bayesian approach assumes that p is given by a random variable, whose distribution is called the *prior distribution*. The prior is usually based on our previous experiences and knowledge about the system. Since p is a probability, we need prior distributions defined over $[0, 1]$. In particular, for mathematical convenience one uses Beta priors, which are defined by the following probability density (for real parameters $\alpha, \beta > 0$):

$$\forall u \in [0,1] \quad g(u, \alpha, \beta) = \frac{1}{B(\alpha, \beta)} u^{\alpha-1}(1-u)^{\beta-1} \tag{4}$$

where the Beta function $B(\alpha, \beta)$ is defined as:

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt \; . \tag{5}$$

For later use in both the hypothesis test and estimation algorithms, the Beta distribution function $F_{(\alpha,\beta)}(u)$ of parameters α, β is defined for all $u \in [0, 1]$:

$$F_{(\alpha,\beta)}(u) = \int_0^u g(t, \alpha, \beta) \; dt \; . \tag{6}$$

Let $d = (x_1, \ldots, x_n)$ denote n samples of the Bernoulli random variable X defined by (2). Let H_0 and H_1 be the hypotheses in (3), and suppose that the *prior probabilities* $P(H_0)$ and $P(H_1)$ are strictly positive and satisfy $P(H_0) + P(H_1) = 1$. By Bayes' theorem, the *posterior probabilities* of H_0 and H_1 with respect to data d are:

$$P(H_i|d) = \frac{P(d|H_i)P(H_i)}{P(d)} \quad (i = 0, 1)$$

for every d with $P(d) > 0$. In our case $P(d)$ is always nonzero - there are no impossible *finite* sequences of data. The hypothesis test method is based on the Bayes Factor, which is the likelihood ratio of the sampled data with respect to the two hypotheses. The Bayes Factor $\mathcal{B}$ of sample d and hypotheses H_0 and H_1 is

$$\mathcal{B} = \frac{P(d|H_0)}{P(d|H_1)} \; .$$

Therefore, $\mathcal{B}$ can be interpreted as a measure of evidence (given by the data d) in favor of H_0. Now, fix a threshold $T > 1$. The algorithm iteratively draws independent and identically distributed (iid) sample traces $\sigma_1, \sigma_2, ...$, and checks whether they satisfy ϕ. After checking each trace, the algorithm computes the Bayes Factor $\mathcal{B}$ to check if it has obtained conclusive evidence. The algorithm accepts H_0 if $\mathcal{B} > T$, and rejects H_0 (accepting H_1) if $\mathcal{B} < \frac{1}{T}$. Otherwise ($\frac{1}{T} \leqslant \mathcal{B} \leqslant T$) it continues drawing iid samples.

The Bayes factor of H_0:$p \geqslant \theta$ vs. H_1:$p < \theta$ with Bernoulli samples $(x_1, \ldots, x_n)$ and Beta prior of parameters α, β can be computed in terms of the Beta distribution function as:

$$\mathcal{B}_n = \frac{P(H_1)}{P(H_0)} \cdot \left(\frac{1}{F_{(x+\alpha, n-x+\beta)}(\theta)} - 1 \right)$$

where $x = \sum_{i=1}^n x_i$ is the number of successes in $(x_1, \ldots, x_n)$ [2]. The Beta distribution function can be efficiently computed by standard software packages, such as the GNU Scientific Library. Therefore, no numerical integration is required for the evaluation of the Bayes Factor.

Finally, it is most important to bound the error probability, *i.e.*, the probability that we reject (accept) the null hypothesis although it is true (false).

Theorem. *[2] The error probability for the Bayesian hypothesis testing algorithm is bounded above by $\frac{1}{T}$, where T is the Bayes Factor threshold given as input.*

We note that the (vastly) dominant complexity factor in statistical model checking is due to system simulation. The complexity of statistical computations *per se* is not an issue.

B. Bayesian Interval Estimation.

In estimation one is interested in computing a value (an estimate) which is, with high probability, close to p, the true probability that the model satisfies a given property. The estimate is usually in the form of a confidence interval - an interval in $[0, 1]$ which contains p with high probability. The estimation method makes use of Bayes' theorem for densities.

Proposition (Bayes Theorem for densities)**.** *Let $x_1, \ldots, x_n$ be a sample drawn from a density $f(\cdot|u)$, where u is given by a random variable U over $(0, 1)$ with density is g. Then, the posterior density of U given the data $x_1, \ldots, x_n$ is:*

$$q(u|x_1, \ldots, x_n) = \frac{f(x_1, \ldots, x_n|u)g(u)}{\int_0^1 f(x_1, \ldots, x_n|v)g(v) \; dv} \tag{7}$$

Since we assume conditionally independent, identically distributed - iid - samples, the density $f(x_1, \ldots, x_n|u)$ factorizes as $\prod_{i=1}^n f(x_i|u)$, where $f(x_i|u)$ is a Bernoulli mass function as in (1).

For a prior distribution over p and sampled data, Bayes' theorem gives the *posterior* distribution of p (*i.e.*, the distribution of p given the data sampled and chosen prior). This means that one can estimate p with the mean of the posterior distribution. Furthermore, by integrating the posterior density over a

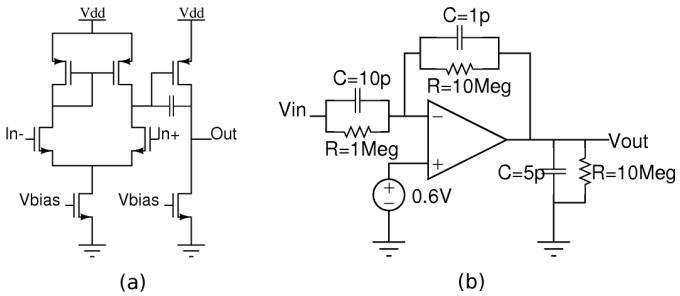

Fig. 1. (a) The two-stage OP amplifier; (b) The inverting configured amplifier

TABLE I
THE SET OF SPECIFICATIONS USED IN THE EXPERIMENT

Spec	Name	Value	Simulation
1	Input Offset Voltage	<1 mV	Transient
2	Output Swing Range	0.2 V to 1.0 V	Transient
3	Slew Rate	>25 V/μSec	Transient
4	Open-Loop Voltage Gain	>8000 V/V	AC
5	Loop-Gain Unit-gain Frequency	>10 MHz	AC
6	Phase Margin	>60°	AC

suitably chosen interval one can compute a Bayes interval estimate with any given confidence probability. Fix a confidence probability $c \in (\frac{1}{2}, 1)$ and a half-interval width $\delta \in (0, \frac{1}{2})$. The algorithm iteratively draws iid traces, checks whether they satisfy ϕ, and builds an interval of total width 2δ, centered on the posterior mean. If the posterior probability over such interval is greater than c the algorithm stops; otherwise, it continues sampling. The algorithm thus returns an interval of length 2δ which contains p with probability at least c. It also returns the posterior mean as an estimate of p.

Again, the posterior probability of an interval is straightforwardly calculated by means of the Beta distribution function. The posterior probability over the interval (t_0, t_1) can be computed as [2]:

$$\int_{t_0}^{t_1} q(u|x_1, \ldots, x_n)\, du = F_{(x+\alpha, n-x+\beta)}(t_1) - F_{(x+\alpha, n-x+\beta)}(t_0)$$

where α and β are the Beta prior parameters, and $x = \sum_{i=1}^{n} x_i$ is the number of successes in the sample $(x_1, \ldots, x_n)$.

IV. SPICE MODEL

We study a gate-level SPICE model describing a two-stage operational amplifier (op amp) as shown in Fig. 1(a). The op amp is compensated in its bandwidth to ensure stability when it is employed in an inverting amplifier system as shown in Fig. 1(b). Finally, the op amp is designed to satisfy the specifications shown in Table I with a CMOS 90nm technology.

In [1], the authors used a MATLAB Simulink model, while in this work we prefer SPICE models, which are more commonly used throughout the analog design and validation process. The SPICE simulator returns both transient and AC responses so that our BLTL trace checker can check the compliance of the simulated traces to both transient and frequency-domain properties. Note that different specifications are measured under different circuit configurations, different input signals and different kinds of simulations.

A. Transient Properties

The compliance of the model circuit to the specifications given in Table I can be measured directly from the simulation traces. Since BLTL formulae specify temporal properties over traces, we can translate the specifications into BLTL formulae. For example, input offset voltage is defined as the value of differential input that makes output equal to the DC biasing voltage, which is 0.6 volts in this example. Therefore, we can specify this property by the following formula: $\mathbf{G}^{[100\mu s]}((V_{out} = 0.6) \rightarrow (|V_{in+} - V_{in-}| < 1mV))$, where 100 μs is the end time of transient simulation. This formula states that "Within 100 μs, it is always the case that when V_{out} equals 0.6 volts, the difference between V_{in+} and V_{in-} is smaller than 1 mV". However, the formula might be evaluated as true while V_{out} never equals 0.6 volts within 100 μs, or there might not be a sample point where V_{out} is *exactly equal* to 0.6 volts although V_{out} *did cross* 0.6 volts. Therefore, another BLTL formula, "Within 100 μs, V_{out} eventually equals 0.6 volts", is added in conjunction to prune out the situation where V_{out} never equals 0.6 volts. Also, we use linear interpolation with an absolute tolerance of 1μV for equality testing to capture crossing events. The resulting BLTL transient specifications are summarized in the upper half of Table II.

B. Frequency-Domain Properties

As opposed to introducing a frequency-domain predicate p_F involving Fourier transformation and then statistically model checking $Pr_{\geq\theta}(p_F)$ as described in [1], we elaborate the notion of frequency-domain predicates for more general analog specifications by providing capability to specify properties on AC small-signal responses. Since AC responses are waveforms sampled at different input frequencies in a strictly increasing manner just like time-stamps in transient responses, we simply substitute frequency-stamps for time-stamps to check frequency domain properties using the same BLTL trace checker. The BLTL specifications for frequency-domain properties are summarized in the lower half of Table II. Note that the AC response is complex-valued. *Vmag* and *Vphase* are used to represent the magnitude and phase of the AC response.

V. RESULTS

A. Experimental Setup

All our experiments have been performed on a Linux virtual machine running on a Windows 7, 2.26GHz i3-350M, 4GB RAM computer. The circuit simulator used was NGSPICE, and the BLTL trace checker was written in C++.

TABLE II
THE SET OF BLTL FORMULAE USED TO SPECIFY PROPERTIES

Spec	BLTL Formula for transient properties
1	$\mathbf{F}^{[100\mu s]}(V_{out} = 0.6)$
	$\wedge\mathbf{G}^{[100\mu s]}((V_{out} = 0.6) \rightarrow (\lvert V_{i+} - V_{i-}\rvert < 0.001))$
2	$\mathbf{F}^{[100\mu s]}(V_{out} < 0.2) \wedge \mathbf{F}^{[100\mu s]}(V_{out} > 1.0)$
3	$\mathbf{G}^{[100\mu s]}(((V_{out} = 1.0 \wedge V_{in} > 0.65) \rightarrow \mathbf{F}^{[0.008\mu s]}(V_{out} < 0.8))$
	$\wedge(V_{out} = 0.2 \wedge V_{in} < 0.55) \rightarrow \mathbf{F}^{[0.008\mu s]}(V_{out} > 0.4))$
Spec	**BLTL Formula for frequency-domain properties**
4	$\mathbf{G}^{[1KHz]}(Vmag_{out} > 8000)$
5	$\mathbf{G}^{[10MHz]}(Vmag_{out} > 1)$
6	$\mathbf{F}^{[10GHz]}(Vmag_{out} = 1)$
	$\wedge\mathbf{G}^{[10GHz]}((Vmag_{out} = 1) \rightarrow (Vphase_{out} > 60°))$

To model the process variation, threshold voltage and oxide thickness, gate width and gate length of each MOSFET are specified by normal random variables, so the circuit might fail to satisfy specifications under certain values of parameters. For example, a differential pair designed symmetrically should have zero input offset voltage. However, process variation can induce mismatches between the differential pair so that the op amp might have a non-zero offset voltage larger than specified.

B. Statistical Model Checking

For each experiment, the null hypothesis (H_0), the property we would like to check, is a formula in Table II with a probability threshold θ. For example, consider Spec 4 and $\theta = 0.95$, we model check the following formula (null hypothesis H_0) :

$$\mathcal{M} \models Pr_{\geqslant 0.95}[\mathbf{G}^{[1KHz]}(Vmag_{out} > 8000)] \; .$$

The probability threshold θ should be set according to the user's needs. To achieve a good yield for the circuit after manufacturing, θ needs to be set close to one, so we choose $\theta = 0.95$ in the first place.

In the commonly used Monte Carlo simulation, the user determines the number of samples, which reflects the testing strength needed. The sample size is normally larger than 1000, and it can grow up to tens of thousand to obtain results with reasonable accuracy. In this experiment, the sample size is chosen to be 1000. In Table III, we summarize the results of the 1000-sample Monte Carlo simulation and Bayesian SMC algorithm with $\theta = 0.95$ and Bayes factor threshold T = 1000, that is, the probability of error is bounded by 0.001. As seen in the table, the measured means and variances of Spec 2, 3, 5 and 6 show that design margins are adequate, so very high yields can be expected. In contrast to Spec 2, 3, 5, 6, the op amp's 3-sigma performance (mean$\pm$3$\times$variance) does not satisfy Spec 1 and 4. This suggests the need for stricter design margins in Spec 1 and 4 for high yield, although Spec 4 satisfies the yield of 0.95. The SMC algorithm terminates in 303s with largest sample size of 239 (less than one-fourth of the Monte Carlo simulation), which is relatively good for such a small error probability. With no surprise, circuit simulation

TABLE III
RESULTS OF MONTE CARLO AND SMC WITH UNIFORM PRIORS, THRESHOLD θ = 0.95, T=1000, ✓: H_0 ACCEPTED, ×: H_0 REJECTED

Monte Carlo (1000 samples) — Measured Value					SMC
	Specification	Mean	Stddev	Yield	Samples/Runtime
1	Offset Voltage (mV)	.436	.597	.826	× 31/39s
2	Swing Range Min (V)	.104	.006	1.00	✓77/98s
	Swing Range Max (V)	1.08	.005	1.00	✓77/98s
3	Negative Slew Rate (V/μSec)	-40.2	1.17	1.00	✓77/98s
	Positive Slew Rate (V/μSec)	56.4	2.54	1.00	✓77/98s
4	Open-Loop Gain (V/V)	8768	448	.975	✓239/303s
5	Loop-Gain UGF (MHz)	19.9	0.30	1.00	✓77/98s
6	Phase Margin (°)	64.1	0.44	1.00	✓77/98s

TABLE IV
NUMBER OF SAMPLES/RUNTIME VS. PROBABILITY THRESHOLD FOR SMC WITH UNIFORM PRIORS AND T=1000, ✓:H_0 ACCEPTED, ×:H_0 REJECTED

	Probability Threshold θ				
Spec	0.7	0.8	0.9	0.99	0.999
1	✓77/105s	✓9933/12161s	× 201/275s	× 10/13s	× 7/9s
2	✓16/18s	✓24/27s	✓44/51s	✓239/280s	✓693/813s
3	✓16/23s	✓24/31s	✓44/57s	✓239/316s	✓693/916s
4	✓23/26s	✓43/49s	✓98/114s	× 1103/1309s	× 50/57s
5	✓16/18s	✓24/28s	✓44/51s	✓239/279s	✓693/807s
6	✓16/20s	✓24/30s	✓44/55s	✓239/303s	✓693/882s

dominates the total runtime (277.9s, 91.7% of the total runtime), while the BLTL trace checker took 24.6s, and the functions for file format conversion and interpolating of crossing events took 0.27s. We note that the time spent on computing the statistical tests was only 0.32s.

Next, in Table IV we report the results of different probability thresholds. In most cases, SMC terminates very quickly even with a probability threshold of 0.999. Higher numbers of samples occur when the actual probability that the formula is true is very close to the threshold. In our op amp example, the probability of satisfying Spec 1 is close to 0.8, and this results in a runtime of 12161s (9933 samples).

Finally, we use the Bayesian estimation algorithm with uniform priors to estimate the unknown probability p that the op amp satisfies each specification. Two experiments were performed with half-interval width $\delta = 0.05$ and $\delta = 0.01$. For each δ, we used two values of the confidence probability. Experimental results are shown in Table V and VI.

TABLE V
POSTERIOR MEAN/SAMPLES/RUNTIME VS. INTERVAL COVERAGE FOR BAYESIAN ESTIMATION METHOD WITH UNIFORM PRIORS AND δ=0.05

	Interval Coverage					
	0.99			0.999		
Spec	Mean	Samples	Runtime	Mean	Samples	Runtime
1	0.8082	410	530s	0.8195	652	858s
4	0.9591	96	122s	0.9685	125	164s
2,3,5,6	0.9777	43	55s	0.9850	65	84s

TABLE VI
POSTERIOR MEAN/SAMPLES/RUNTIME VS. INTERVAL COVERAGE FOR BAYESIAN ESTIMATION METHOD WITH UNIFORM PRIORS AND δ=0.01

	Interval Coverage					
	0.99			0.999		
Spec	Mean	Samples	Runtime	Mean	Samples	Runtime
1	0.8114	10150	13605s	0.8124	16512	21685s
4	0.9660	2204	2860s	0.9685	3860	5098s
2,3,5,6	0.9956	227	296s	0.9970	341	439s

C. Discussion

The experimental results show that the Bayes estimation algorithm is more efficient when p is close to one, but needs more samples when p is closer to 0.5. This is due to the fact that the variance of a Bernoulli random variable is the largest when $p = 0.5$.

The Bayesian SMC algorithm is faster when the threshold probability θ differs significantly from the unknown probability p, and it works better when θ is close to one (or zero). Since circuits are designed in order for manufactured chips to have yields close to one, the threshold probability θ should be set accordingly. Therefore, statistical model checking is likely to work well for the problem of analog circuit verification.

VI. RELATED WORK

Verification of analog circuits is a very active research area in which several different approaches have been proposed. A popular approach is building finite-state abstractions and using reachability analysis techniques, such as [10, 11]. Several works rely on abstracted models of analog circuits. In [12], Little *et al.* used Labeled Hybrid Petri Nets (LHPN) as an abstracted model to verify analog circuits. However, the finite-state abstraction approach does not scale well with system size, and cannot be easily integrated with circuit simulators such as SPICE. Another approach uses a theorem prover to prove design properties by means of inference rules. Recently, Denman *et al.* [13] used the MetiTarski theorem prover and obtained reasonable results by tackling non-linearity with piecewise-linear models. However, the approaches above focus on a given set of parameters with no process variation, which is most troublesome to designers. In [14], Little *et al.* took process variation into account by generating LHPN models from simulation traces. However, this approach is not fully automated because users must manually specify the thresholds which are used to separate different operation phases of the circuit. In [15], stochastic differential equations (SDEs) are used to specify circuits under variation and noise, but SDE-based approaches still have difficulties in dealing with non-linearity.

VII. CONCLUSION

In this paper, we have successfully applied the Bayesian statistical model checking algorithm to analog circuit verification by integrating the SPICE simulator, a BLTL trace checker, Monte Carlo sampling, and statistical testing and estimation. Also, we have demonstrated the feasibility of using BLTL specifications for a simple analog circuit. Since BLTL is a very expressive language which can specify complex interactions between signals, we expect that our approach will be able to verify circuits with more complicated specifications. Although in statistical model checking a correct answer cannot be guaranteed, the error probability can be made arbitrarily small by the user. For more speedup, we are investigating the integration of our algorithm with online BLTL checking, which can terminate simulation as soon as the property can be decided.

REFERENCES

[1] E. M. Clarke, A. Donzé, and A. Legay, "Statistical model checking of mixed-analog circuits with an application to a third order delta-sigma modulator," in *Haifa Verification Conference '08*, ser. LNCS, vol. 5394, 2009, pp. 149–163.

[2] P. Zuliani, A. Platzer, and E. M. Clarke, "Bayesian statistical model checking with application to Stateflow/Simulink verification," in *HSCC*, 2010, pp. 243–252.

[3] H. L. S. Younes and R. G. Simmons, "Statistical probabilistic model checking with a focus on time-bounded properties," *Inf. Comput.*, vol. 204, no. 9, pp. 1368–1409, 2006.

[4] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, "Approximate probabilistic model checking," in *VMCAI*, ser. LNCS, vol. 2937, 2004, pp. 73–84.

[5] K. Sen, M. Viswanathan, and G. Agha, "Statistical model checking of black-box probabilistic systems," in *CAV*, ser. LNCS, vol. 3114, 2004, pp. 202–215.

[6] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan, "Symbolic model checking for probabilistic processes," in *ICALP*, ser. LNCS, vol. 1256, 1997, pp. 430–440.

[7] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time Markov chains," *IEEE Trans. Software Eng.*, vol. 29, no. 6, pp. 524–541, 2003.

[8] A. Pnueli, "The temporal logic of programs," in *FOCS*. IEEE, 1977, pp. 46–57.

[9] R. Grosu and S. Smolka, "Monte Carlo Model Checking," in *TACAS*, ser. LNCS, vol. 3440, 2005, pp. 271–286.

[10] A. Chutinan and B. H. Krogh, "Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations," in *HSCC*, 1999, pp. 76–90.

[11] S. Gupta, B. H. Krogh, and R. A. Rutenbar, "Towards formal verification of analog designs," in *ICCAD*, 2004, pp. 210–217.

[12] S. Little, N. Seegmiller, D. Walter, C. J. Myers, and T. Yoneda, "Verification of analog/mixed-signal circuits using labeled hybrid petri nets," in *ICCAD*, 2006, pp. 275–282.

[13] W. Denman, B. Akbarpour, S. Tahar, M. H. Zaki, and L. C. Paulson, "Formal verification of analog designs using metitarski," in *FMCAD*, 2009, pp. 93–100.

[14] S. Little, D. Walter, K. Jones, and C. J. Myers, "Analog/mixed-signal circuit verification using models generated from simulation traces," in *ATVA*, 2007, pp. 114–128.

[15] R. Narayanan, B. Akbarpour, M. H. Zaki, S. Tahar, and L. C. Paulson, "Formal verification of analog circuits in the presence of noise and process variation," in *DATE*, 2010, pp. 1309–1312.