Application of Automated Model Generation Techniques to Analog/Mixed-Signal Circuits

Scott Little
School of Computing
University of Utah
Salt Lake City, UT, 84112 USA
Email: little@cs.utah.edu

Alper Sen
Freescale Semiconductor
Austin, TX 78729 USA
Email: alper.sen@freescale.com

Chris Myers
Electrical and
Computer Engineering Department
University of Utah
Salt Lake City, UT, 84112 USA
Email: myers@ece.utah.edu

Abstract—Abstract models of analog/mixed-signal (AMS) circuits can be used for formal verification and system-level simulation. The difficulty of creating these models precludes their widespread use. This paper presents an automated method to generate abstract models appropriate for system-level simulation and formal verification. This method uses simulation traces and thresholds on the design variables to generate a piecewise-linear representation of the system. This piecewise-linear representation can be converted to a Verilog-AMS model or a Labeled Hybrid Petri Net formal model. Results are presented for the model generation, simulation, and verification of a PLL phase detector circuit.

I. INTRODUCTION

System-level design and verification challenges are increasing as analog/mixed-signal (AMS) designs become more common and functionally complex [1]. This increased complexity is leading to an increased number of functionality related bugs in AMS designs. These bugs are due in large part to inadequate modeling and verification methodologies for AMS design. Modeling is a challenge due to the need for models at differing levels of abstraction. Designers need and are comfortable with transistor-level models. However, abstract models are becoming necessary at the system level, but the expertise and time to create these models is often missing. Verification of AMS systems has traditionally been done in an ad hoc manner. More organization and formality is required to properly validate the increasing system complexity. It is clear that automated modeling and verification techniques can have a large impact in improving AMS system validation.

Abstract or reduced order modeling of linear systems is a well understood discipline with a strong foundation [7], [9]. This is not true for nonlinear systems in general. Although there are promising solutions for parts of the space [2], [3], [8]. There have been numerous investigations into the general problem although they have been met with scalability problems [10], [11]. As a result, automated abstract circuit modeling has not become common in AMS workflows.

Significant research effort has been invested in improving the uniformity and formality of design and verification of

digital circuits. However, AMS verification has been slow to benefit from the efforts in the digital domain. Historically, digital verification has been done with switch-level simulation on very large designs. Comparatively, analog designs have been very small and designed using accurate transistor-level simulations at the block level. This history has resulted in two very different verification methodologies. The digital verification methodology is more formal and handled by a group of designers and verification engineers. Analog verification is ad hoc and often handled by a single designer. In the verification of a mixed-signal circuit, these two methodologies converge in a seemingly incompatible mix of methodologies.

We believe that new verification methodologies are needed for mixed-signal designs. Based on this view, in our previous work we developed, LHPN Embedded/Mixed-signal Analyzer (LEMA), a tool to be used in an AMS verification workflow. LEMA is a formal verification tool for the verification of AMS systems. It has two main components, a model generation component and a verification component as shown in Figure 1. The verification component of the system is well described in previous work [4], [12]–[14]. LEMA is primarily intended to aid in AMS verification, but a difficulty with our previous work is creating an LHPN model. An initial discussion of the LHPN model generation component is found in [5]. This paper provides an extended discussion of the LHPN model generation as well as describing Verilog-AMS generation. Verilog-AMS model generation has been added to LEMA to provide designers with an automated method to obtain abstract models of their circuits. These generated models can be used for formal verification or system-level simulation.

II. MOTIVATING EXAMPLE

The switched capacitor integrator circuit shown in Figure 2 is a circuit used as a component in many AMS circuits such as ADCs and DACs. Although only a small piece of these complex circuits, the switched capacitor integrator proves to be a useful example illustrating the type of problems that can be present in AMS circuit designs. Discrete-time integrators typically utilize switched capacitor circuits to accumulate charge. Capacitor mismatch can cause gain errors in integra-
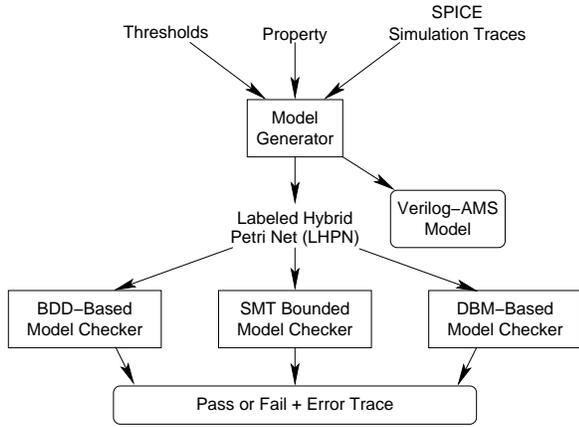
Fig. 1. LEMA workflow diagram.

tors. Also, the CMOS switch elements in switched capacitor circuits inject charge when they transition from closed to open. This charge injection is difficult to control with any precision, and its voltage-dependent nature leads to circuits that have a weak signal-dependent behavior. This can cause integrators to have slightly different gains depending on their current state and input value. Circuits using integrators run the risk of the integrator saturating near a power supply rail. Therefore, the verification property to check for this circuit is whether the voltage V_{out} can rise above 2000mV or fall below -2000mV. It is essential to ensure that this never happens during operation under any possible permutation of component variations. For simplicity, we assume the major source of uncertainty in this example is that capacitor C_2 can vary dynamically by ± 10 percent from its nominal value. This circuit, therefore, must be verified for all values in this range [6].
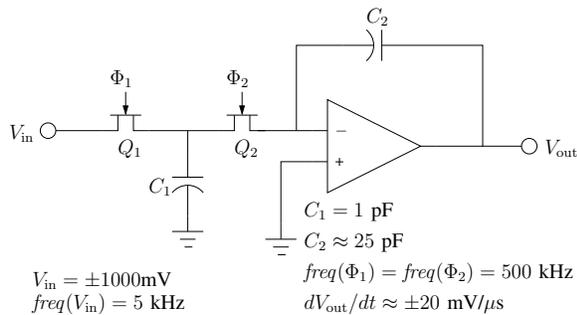

Fig. 2. A schematic of a switched capacitor integrator.

III. MODEL GENERATION

LEMA uses an ad hoc method to create a piecewise linear model for the system. The piecewise approximation for each variable is determined by the thresholds provided for the variable. The thresholds determine the separation points for each piece in the linearization. Based on the simulation traces a rate of change is calculated for each piece and transitions between

the pieces are extracted. Model quality is dependent on the simulation traces, selected design variables, and thresholds.

LEMA's model generation algorithm consists of four major steps as illustrated in Algorithm 1. The inputs to genModel are a set of simulation traces, T; thresholds on the design variables, $thresholds$; a configuration file, c; and a property to verify, $prop$. The configuration file is used to customize model generation parameters. The property is introduced in the graph generation stage and only affects the LHPN output. The model generation is illustrated using simulation data from a $400\mu s$ transient simulation trace of the switched capacitor integrator using a capacitor value of 23 pF for C_2 shown in Table I. The thresholds on V_{in} and V_{out} are 0 V. The property for the design is $V_{out} \geq -2000 \wedge V_{out} < 2000$.

Algorithm 1: genModel(T, $thresholds$, c, $prop$)

1   $G = \emptyset$;
2   **forall** $t \in T$ **do**
3      $binData = \text{binData}(t, thresholds)$;
4      $rates = \text{calculateRates}(t, c.window, binData)$;
5      $dmv = \text{detectDMV}(t, c.\epsilon, c.ratio, c.\tau_{\min})$;
6      $dmv = \text{calculateDMV}(t, c, dmv)$;
7      $G = \text{updateG}(G, binData, rates, dmv, prop)$;

The first step, binData, sorts the data into bins according to thresholds. At $0\mu s$, V_{in} and V_{out} are below the threshold of 0V, so the bin encoding in the fourth column of Table I is 00 for $0\mu s$. The bin vector lists the bin values for V_{in} and V_{out} respectively. As V_{out} rises it exceeds the threshold of 0V resulting in a bin encoding of 01 as shown at the $46.98\mu s$ point of Table I. The bins represent the individual pieces in the piecewise linear model. As a result, increasing the number of thresholds increases the accuracy of the model at a cost of increased complexity.

After bins have been assigned to each data point, the rates are calculated for each bin using calculateRates. A rate is calculated for each eligible data point. Not all points are eligible for rate calculations due to a low pass filtering technique used to smooth edge effects and transitory pulses. The low pass filtering uses a sliding window approach. The size of the window, $c.window$, can be configured in the configuration file. The sliding window approach works by calculating the rate between the current point and a point $c.window$ points further in time. For instance, in Table I, the rate of V_{out} is calculated for 0.5 μs but not for $46.48\mu s$. A rate is not calculated for $46.48\mu s$ because the value of V_{out} $c.window$ points later is in a different bin.

In AMS designs, it is expected that digital signals are present. For efficiency purposes, digital-like signals are treated differently. Instead of allowing them to change with a specific rate, a constant value can be directly assigned to the variable at a specified time or when specific predicates become true. Discrete multi-valued (DMV) variables can be specified in the

Time (μs)	V_{in} (mV)	V_{out} (mV)	Bin	$\Delta V_{\text{in}}/\Delta t$ (mV/μs)	$\Delta V_{\text{out}}/\Delta t$ (mV/μs)	V_{in} time
0.00	-1000	-1000	00	-227.85	21.29	0.0
0.50	-1000	-999	00	0.0	21.74	0.5
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
46.48	-1000	-0.4	00	-	-	46.48
46.98	-1000	10	01	0.0	21.74	46.98
47.48	-1000	21	01	0.0	21.74	47.48
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
100.50	-1000	1174	01	-	-	100.50
100.54	-840	1174	01	-	-	100.54
100.62	-520	1176	01	-	-	100.70
100.78	120	1176	11	275.00	-21.08	0.08
101.00	1000	1174	11	0.0	-21.74	0.30
101.03	1.0	1173	11	0.0	-21.74	0.33
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
154.98	1000	0.3	11	-	-	54.28
155.48	1000	-11	10	0.0	-21.74	54.78
155.98	1000	-21	10	0.0	-21.74	55.28
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
200.00	1000	-978	10	-	-	99.30
200.04	840	-979	10	-	-	99.34
200.12	520	-980	10	-	-	99.50
200.28	-120	-981	00	-275.00	21.08	0.08
200.50	-1000	-978	00	0.0	21.74	0.30
200.53	-1000	-976	00	0.0	21.74	0.33
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
400.00	1000	-957	10	-	-	99.34

configuration file as well as automatically detected. A DMV variable is detected when it remains constant for a specified amount of time compared to the total time for the simulation. Remaining constant is defined as staying within an ϵ bound for at minimum time, $\tau_{\min}$. The DMV detection algorithm, detectDMV, is described in Algorithm 2. The algorithm tests each variable in the trace separately. The analysis begins with the first point and checks to see if the second point is equivalent within the specified ϵ. If equivalence is found the next point is tested. This occurs until a point is found, that is not equivalent. When this occurs, the time elapsed between the initial point and the current position is tested. If this time, $(\tau_{\text{pos}} - \tau_{\text{point}})$, is greater than $\tau_{\min}$, the value is added to the running total of constant time, t_{const}. The algorithm then continues from the next point. When all points have been analyzed, the ratio of constant time, τ_{const}, to total time for the trace, τ_{var}, is calculated. If this ratio exceeds the specified ratio, the variable is marked DMV.

The results of detectDMV are dependent on the type of variable. Input variables, specified in the configuration file, require extra calculation. The algorithm assumes that input DMV variables are periodic, so a minimum and maximum delay is calculated for each constant value. Constant values are calculated for all DMV variables. For instance, V_{in} is an input variable that is found to be a DMV variable. As can be deduced from column two of Table I, V_{in} is found to have two

---

Algorithm 2: detectDMV($t, \epsilon, ratio, \tau_{\min}$)

1 **forall** $var \in t$ **do**
2     $point, pos, \tau_{\text{const}} = 0$;
3   **while** $point < \text{len}(t[var])$ **do**
4         $point = pos$;
5   **while** $\text{equiv}(\epsilon, point, pos + 1)$ **do**
6             $pos + +$;
7   **if** $(\tau_{\text{pos}} - \tau_{\text{point}}) \geq \tau_{\min}$ **then**
8             $\tau_{\text{const}} + = \tau_{\text{pos}} - \tau_{\text{point}}$;
9   **if** $(\tau_{\text{const}}/\tau_{\text{var}}) \geq ratio$ **then**
10         $dmv = dmv \cup var$;
11 **return** dmv;

---

constant values, $-1V$ and $1V$. Each value is found to have a delay between 99 and 101 μs as demonstrated in the final column of Table I.

After the needed information has been calculated for a trace, updateGraph updates or creates the graph, G, with the new information. Nodes in the graph are created for each bin discovered as well as for each constant value of each DMV variable. The nodes are annotated with the appropriate rate, constant value, and delay information. Edges in the graph are created when a transition is observed between bins or constant values. The final graph is used to generate the Verilog-AMS and LHPN models.

A. Verilog-AMS model generation

Throughout the model generation process, ranges of values are extracted. Verilog-AMS does not support ranges of values. As a result, ranges are averaged for the Verilog-AMS model. A top level **inout** variable is created for each variable in the graph. Vin_io and Vout_io are the top level variables in the Verilog-AMS code for the switched capacitor integrator shown in Figure 3. This is the only information provided by the Verilog-AMS model for input variables. It is expected that the Verilog-AMS model will be driven by another circuit or test bench environment and this entity will provide the input. **real** variables of the form <varname>_var are created to hold the current value of each non-input variable, Vout_var for the switched capacitor integrator example. Variables with a rate (non-DMV variables) are also provided a **real** variable to store the current rate, for example Vout_rate.

The initial conditions for each variable and rate are set in the **initial_step** statement. Each edge containing a rate or constant value assignment is translated into a **cross** statement. The parameters for the cross statement are are extracted from the nodes between the edge. The source and sink nodes are compared. It is assumed that there is only one bin difference between the nodes. The variable for the changing bin is set as the compare variable. The threshold representing the division between the changing bins is the numerical value. The direction of the cross statement is calculated based on the signal's direction. If the signal is increasing, a 1 is used, and if the signal is decreasing, a -1 is used. In Figure 3, the first **cross** statement is created for an edge where V_{in} changes from $1V$ to $-1V$ by crossing the $0V$ threshold. As a result, Vin_io is the compare variable; 0.0 is the numerical value; and -1 is the direction. The sink place sets the rate of V_{out} to 0.020, so this assignment is made to Vout_rate in the execution block of the **cross** statement. If there are multiple differences, an **if** statement will be used in place of the **cross** statement. The **if** statement is considered a less optimal solution as it has a weaker interaction with the simulator and does not provide a method to specify the direction of signal change. A global timer is added to update all rates at an appropriate interval which can be specified in the configuration file. For the switched capacitor integrator, an interval of $1\mu s$ is used to update the value of Vout_var based on Vout_rate. Finally, a **transition** statement is added for each non-input variable to quickly transition the value of the internal variable to the external interface.

B. LHPN model generation

The LHPN model generation is more direct than the Verilog-AMS because an LHPN is a directed graph that supports ranges on variables. LHPNs are not described in depth here, but a full description is found in [14]. In the translation, each node of the graph is translated to a place, p, in the LHPN. The fourth column of Table I shows four different bins discovered in the simulation trace. These four bins are represented by p_3 - p_6 of Figure 4a which models V_{out}. Figure 4b is the

```
module swCap(Vin_io,Vout_io);
  inout Vin_io, Vout_io;
  electrical Vin_io, Vout_io;
  real Vout_var, Vout_rate;
  analog begin
    @(initial_step) begin
      Vout_var = -1.00;
      Vout_rate = 0.020;
    end
    @(cross(V(Vin_io)-0.0,-1)) begin
      Vout_rate = 0.020;
    end
    @(cross(V(Vin_io)-0.0,1)) begin
      Vout_rate = -0.020;
    end
    @(timer(0.0,1e-06)) begin
      Vout_var = Vout_var + Vout_rate;
    end
    V(Vout_io) <+
      transition(Vout_var,1p,1p,1p);
  end
endmodule
```

Fig. 3. Verilog-AMS code for the switched capacitor integrator.

LHPN modeling V_{in}. This LHPN has two places, one for each constant value of V_{in}. Each edge in the graph is translated to a transition, t, in the LHPN. The annotations on each graph node are moved to the incoming transition. There are three types of annotations in Figure 4: predicates, assignments, and delay bounds. Predicates are specified with curly braces, {}; assignments are specified within angle brackets, $\langle\rangle$; and delay bounds are specified with square brackets, []. The input variables are modeled in the LHPN as a model for the environment is required to check properties of the system. The property is also integrated into the LHPN model. It is negated and added as an enabling condition to a single place and transition LHPN as shown in Figure 4c. When the transition fires indicating the property has been violated a special Boolean, *fail*, is set to true indicating an error to the analysis engine.
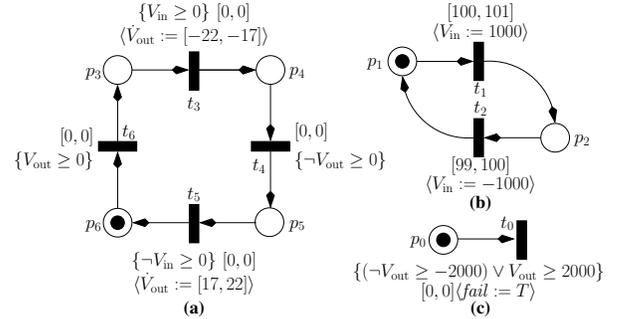

Fig. 4. An LHPN model for the switched capacitor integrator.

IV. PLL PHASE DETECTOR EXAMPLE

Phase locked loops (PLLs) are notoriously difficult circuits to design and validate. There are a small number of major components to a PLL which traditionally include a phase detector, low pass filter, VCO, and a frequency divider. We analyze a phase detector as shown in the schematic diagram of Figure 5. The phase detector is used to measure the phase difference between two input signals and provide this information to the VCO. The VCO uses this information to adjust the phase of the internal PLL clock in order to align the phase of the two input signals. The inputs to the phase detector are *clk* and *gclk*. The $\overline{up}$ and $\overline{down}$ signals are asserted to provide instruction on how to adjust the VCO frequency.
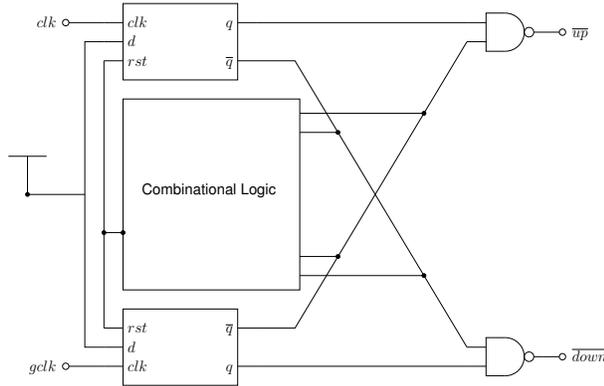

Fig. 5.   A schematic diagram for a PLL phase detector.

The phase detector is simulated using a piecewise linear simulation input 1 μs long that represents reasonable clock skew for one input and a periodic clock signal for the other input. Simulations are also performed using two periodic signals of fixed, but different periods. These simulations are used to build the LHPN and Verilog-AMS models for the phase detector.

The Verilog-AMS and LHPN model are generated in approximately 20 seconds for the phase detector example using two 1 μs transient simulations of the piecewise linear input and a periodic clock input. Eight variables are required for model generation to accurately capture the state of the phase detector. Four of the signals are the inputs and outputs while the remaining four signals are chosen from within the internal signals of the latches. It is logical that signals from the state holding latches are required to delineate the states of the phase detector. As all eight variables are digital signals, each variable is assigned a single threshold equal to $\frac{V_{dd}}{2}$.

Comparison between simulation times for the transistor-level design and the Verilog-AMS model are performed using the same simulation inputs and simulator. Table II presents the results of these simulations. The first four simulations use one piecewise linear input and one periodic input. The final table entry is a result for two periodic inputs. A comparison of the waveforms produced by the two simulations is shown in Figure 6. There is a difference between the two waveforms,

but the abstracted model is accurate enough to be used in a system-level simulation.

TABLE II
A COMPARISON OF SIMULATION TIMES BETWEEN THE
TRANSISTOR-LEVEL MODEL AND THE VERILOG-AMS MODEL.

Sim Length	Verilog-AMS (s)	Transistor-level (s)	Speed-up
0.5 μs	0.54	18.28	33.8
0.5 μs	0.54	17.92	33.2
1 μs	0.81	36.67	45.3
1 μs	0.81	40.46	49.9
2 μs	0.38	9.47	24.9

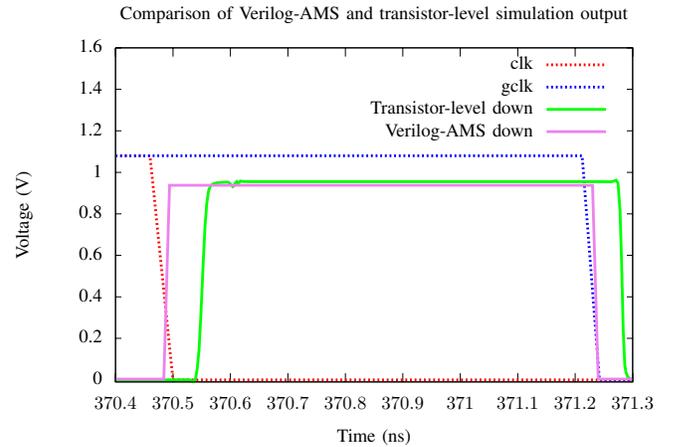Fig. 6.   A comparison of a transistor-level and Verilog-AMS PLL phase detector models.

The LHPN model for the PLL phase detector is composed of 69 places and 87 transitions. The property $\neg(\overline{down} \wedge \overline{up})$ verifies in 0.3 seconds. This property is a sanity check on the outputs of the phase detector ensuring that $\overline{down}$ and $\overline{up}$ are not asserted at the same time. A more complex property for the PLL phase detector is shown as pseudocode in Figure 7. This pseudocode is a behavioral description of the correct input/output operation of the phase detector. This property can not be specified directly in LEMA's property language. To verify the property it is necessary to convert the property to an LHPN with the appropriate *fail* transitions. The resulting property LHPN is composed of 20 transitions and 14 places. This property verifies in 2.12 seconds.

```
if gclk 1 → 0 then
  if up = 1 then up = 0 within 5 ns
  elsif down = 0 then down = 1 within 5 ns
if clk 1 → 0 then
  if down = 1 then down = 0 within 5 ns
  elsif up = 0 then up = 1 within 5 ns
```

Fig. 7.   A property to verify for the PLL phase detector.

V. Conclusions

Algorithms have been developed to generate Verilog-AMS and LHPN models from a set of simulation traces. These algorithms create a piecewise linear model for the circuit. These methods are evaluated using a PLL phase detector circuit. The methods are successful in the case, but during the course of applying the methodology to other circuits several lessons are learned. Modeling quality is dependent on a number of factors which are difficult to understand and control for many circuits. To create a quality model the number and type of simulations must adequately characterize all behaviors of the circuit. It may be difficult to determine and create the needed simulations. The proper variables to distinguish individual states must be present in the simulation traces. Even with a digital circuit like the PLL phase detector it is initially non-obvious which signals are required to delineate the state of the system. This problem is even more complex when dealing with a circuit containing more analog components. The modeling method does not handle start-up conditions well as it is not often possible to find a state variable that distinguishes the start-up of the circuit from steady state operation. The results of the phase detector do show the promise of abstract modeling to decrease system-level simulation. Further exploration in automated abstract model generation should be undertaken.

References

[1] H. Chang and K. Kundert. Verification of complex analog and RF IC designs. In *Proceedings of the IEEE*, volume 95, pages 622–639, Mar. 2007.

[2] X. Lai and J. Roychowdhury. Tp-ppv: Piecewise nonlinear, time-shifted oscillator macromodel extraction for fast, accurate pll simulation. In S. Hassoun, editor, *Proc. International Conference on Computer Aided Design (ICCAD)*, pages 269–274. ACM Press, 2006.

[3] P. Li and L. T. Pileggi. Norm: compact model order reduction of weakly nonlinear systems. In *Proc. Design Automation Conference (DAC)*, pages 472–477, 2003.

[4] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda. Verification of analog/mixed-signal circuits using labeled hybrid petri nets. In *Proc. International Conference on Computer Aided Design (ICCAD)*, pages 275–282. IEEE Computer Society Press, 2006.

[5] S. Little, D. Walter, and C. Myers. Analog/mixed-signal circuit verification using models generated from simulation traces. In *Automated Technology for Verification and Analysis (ATVA)*, Lecture Notes in Computer Science. Springer, 2007.

[6] C. J. Myers, R. R. Harrison, D. Walter, N. Seegmiller, and S. Little. The case for analog circuit verification. *Electronic Notes Theoretical Computer Science*, 153(3):53–63, 2006.

[7] A. Odabasioglu, M. Celik, and L. T. Pileggi. Prima: passive reduced-order interconnect macromodeling algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):645–654, 1998.

[8] J. R. Phillips. Projection-based approaches for model reduction of weakly nonlinear, time-varying systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(2):171–187, 2003.

[9] L. T. Pillage and R. A. Rohrer. Asymptotic waveform evaluation for timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(4):352–366, 1990.

[10] M. Rewieński and J. White. Model order reduction for nonlinear dynamical systems based on trajectory piecewise-linear approximations. *Linear Algebra and its Applications*, 415(2–3):426–454, jun 2006.

[11] S. K. Tiwary and R. A. Rutenbar. Faster, parametric trajectory-based macromodels via localized linear reductions. In S. Hassoun, editor, *Proc. International Conference on Computer Aided Design (ICCAD)*, pages 876–883. ACM Press, 2006.

[12] D. Walter, S. Little, and C. Myers. Bounded model checking of analog and mixed-signal circuits using an smt solver. In *Automated Technology for Verification and Analysis (ATVA)*, Lecture Notes in Computer Science. Springer, 2007.

[13] D. Walter, S. Little, N. Seegmiller, C. J. Myers, and T. Yoneda. Symbolic model checking of analog/mixed-signal circuits. In *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 316–323, 2007.

[14] D. C. Walter. *Verification of analog and mixed-signal circuits using symbolic methods*. PhD thesis, University of Utah, May 2007.