International Journal of Foundations of Computer Science
Vol. 21, No. 2 (2010) 191–210
© World Scientific Publishing Company
DOI: 10.1142/S0129054110007209

ANALOG/MIXED-SIGNAL CIRCUIT VERIFICATION USING MODELS GENERATED FROM SIMULATION TRACES*

SCOTT LITTLE
Freescale Semiconductor
Austin, TX 78729, USA
little@scomotle.org

DAVID WALTER
Virginia State University
Petersburg, VA 23806, USA

KEVIN JONES and CHRIS MYERS
University of Utah
Salt Lake City, UT 84112, USA

ALPER SEN
Bogazici University
Istanbul, Turkey

Received 28 July 2008
Accepted 8 June 2009
Communicated by Kedar Namjoshi and Tomohiro Yoneda

Verification of analog/mixed-signal (AMS) circuits is complicated by the difficulty of obtaining circuit models at suitable levels of abstraction. We propose a method to automatically generate abstract models suitable for formal verification and system-level simulation from transistor-level simulation traces. This paper discusses the application of the proposed methodology to a switched capacitor integrator and PLL phase detector.

Keywords: Formal methods; hybrid Petri nets; analog/mixed-signal circuits; abstract model generation.

1. Introduction

Increased interest in system on a chip design has resulted in a need to improve verification methods for *analog/mixed-signal* (AMS) circuits. Digital circuit verification methodology has changed dramatically in the past ten years while AMS circuit validation methodology remains largely the same. AMS circuit verification is still largely driven by designers using many simulation traces to validate specific properties of a circuit. While this methodology has been used with success for many years, recent trends are stretching it beyond its capacity. Increase in process

*This research is supported by SRC contracts 2005-TJ-1357 and 2008-TJ-1851 and an SRC Graduate Fellowship.

variations and use of mixed-signal circuits present challenges that this simulation only methodology is not well prepared to address.

Currently, most AMS designers use an informal approach to circuit verification. With the aid of a simulator, the designer creates a circuit that under ideal conditions meets a set of specifications. A major concern for circuit designers using today's process technologies is the circuit's resilience to process variation. To help understand how the circuit operates under global variation, corner simulations are run. These simulations evaluate circuit performance for various values of global parameters such as process, voltage, and temperature. There may also be local transistor to transistor process variation. To understand how this variation affects the circuit, Monte Carlo simulation is employed. These methods for exploring global and local variation are very expensive. This expense increases dramatically as more sources of variation are explored. As a result, only the most common sources of variation of the most critical specifications of the most critical circuits are thoroughly validated. The design team also has no real measure of the quality of the verification performed on the design. The correctness of the design is almost solely the responsibility of each designer. Lack of feedback to the designer and large cost to verify the circuit under variation are major concerns when using this simulation only methodology.

Based on the success of formal methods for digital circuits, there has been an increasing body of work in formal methods for AMS circuits. Several tools and methods have been developed to explore the continuous state space of these systems [10, 2, 8, 13, 19, 18]. These methods work well on small examples and have shown some promise to work on larger circuits. One challenge for these methods is the significant effort required to create an appropriate abstract formal model for each circuit of interest. These methods also suffer from high computation costs for the analysis of the model. The more accurately the method explores the state space of the system, the more computationally intensive it is.

In response to these challenges, there has been recent work in verifying formal properties within the framework of simulation. One type of approach attempts to find a finite number of simulation traces that are sufficient to represent all trajectories of the system and therefore prove correctness of the circuit [6, 3, 9, 7]. A second approach verifies formal properties on the simulation traces [17, 11]. A third approach uses simulation traces to generate a formal model which is then analyzed using a state space exploration engine [4]. The method used in `LEMA` [15, 14] is similar to the method by Dastidar et al.[4] that generates a finite state machine (FSM) from a systematic set of simulation traces. Their FSM includes currents, voltages, and time as state variables to generate an acyclic FSM. The state space of the system is divided into symmetric state divisions. After each delta time step, the current state of the simulator is determined and rounded to the center of the appropriate state division. The simulator is then started from this point and run for the next delta time step. This process continues until the global time reaches a user specified maximum. Conversely, our approach uses *Labeled Hybrid Petri Nets* (LHPNs) [19] as the model. The state space is divided as specified by user provided thresholds on signal values. A global timer is not a part of the state space, so the graphs produced may include cycles. Simulation traces are run from start to finish without stopping allowing our model to preserve the original simulation trace.[a]

The novelty of our approach is that the model allows for dynamic variation of parameters. Standard simulation based methods allow for changes in initial conditions and parameters, but these values are then fixed for the duration of the simulation run. Our model explores the system under ranges of initial conditions as well as ranges of dynamically changing parameter values. This additional behavior improves our ability to uncover variation induced errors.

The verification flow supported by our tool, `LEMA`, is shown in Fig. 1. There are two primary input formats accepted by `LEMA`, VHDL-AMS and simulation data. Our previous work [13, 19, 20, 18] describes how a subset of VHDL-AMS can be compiled into a LHPN and formally verified. Each model checker uses a different data structure to represent that state space including: *difference bound matrices* (DBMs) [13, 12], *binary decision diagrams* (BDDs) [19], and *satisfiability modulo theories* (SMT) formulas [18]. The other option is to use the simulation traces already produced by the designer as well as safety properties and thresholds on the signal levels of the design variables to automatically generate AMS HDL and LHPN models of the system [20, 15, 14]. The AMS HDL models are intended for use in system-level simulations. These models are not as accurate as the transistor-level models, but they simulate much faster and support standard simulation environments and workflows. The LHPN model generated by the model generator can be used to formally verify safety properties of the system using one of `LEMA`'s model checkers. Our model generator is the subject of this paper.

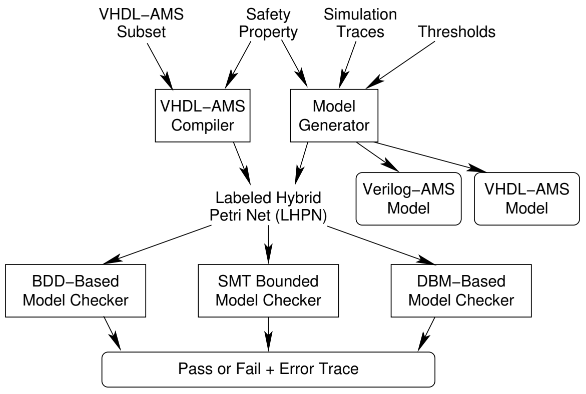

Fig. 1. `LEMA` workflow diagram.

2. Motivating Example

The switched capacitor integrator circuit shown in Fig. 2 is a circuit used as a component in many AMS circuits such as ADCs and DACs. Although only a small

[a]Due to the low pass filtering of the windowing technique, the model does not necessarily preserve the exact input traces. However, the algorithms do ensure that the low pass filtered versions of the input traces are valid possible output traces of the resulting LHPN model.

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. The switched capacitor portion of this circuit includes transistors Q_1 and Q_2 along with capacitor C_1, and it behaves like a resistor with resistance of $1/(C \cdot \mathit{freq})$ (i.e. a 2 MΩ resistor). Capacitor mismatch can cause gain errors in integrators. 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 one of the power supply rails. Therefore, the verification property to check for this circuit is whether or not the voltage V_{out} can rise above 2000 mV or fall below -2000 mV where the values of ± 2000 mV represents the saturation points near the power rails. It is essential to ensure that this never happens during operation under any possible permutation of component variations. For simplicity, this paper assumes for this example that the major source of uncertainty is that the 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 [16].

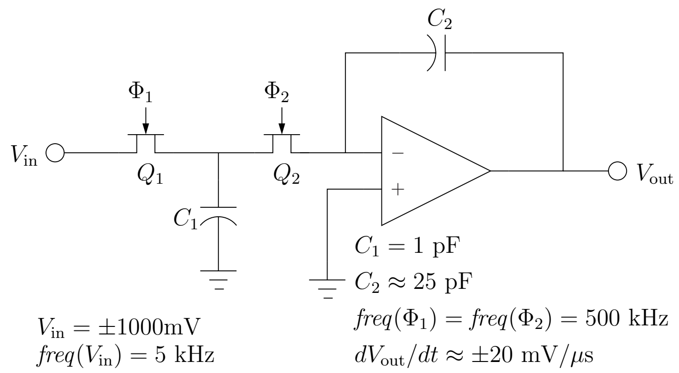
C_2
Φ_1
Φ_2
V_{in}
Q_1
Q_2
C_1
$-$
$+$
V_{out}
$C_1 = 1$ pF
$C_2 \approx 25$ pF
$V_{\text{in}} = \pm 1000$mV
$\mathit{freq}(V_{\text{in}}) = 5$ kHz
$\mathit{freq}(\Phi_1) = \mathit{freq}(\Phi_2) = 500$ kHz
$dV_{\text{out}}/dt \approx \pm 20$ mV/μs

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

3. Labeled Hybrid Petri Nets

An LHPN is a Petri net model developed to represent AMS circuits. The model is inspired by features in both hybrid Petri nets [5] and hybrid automata [1]. An LHPN is a tuple $N = \langle P, T, B, V, F, L, M_0, S_0, Q_0, R_0 \rangle$ where:

- P is a finite set of places;
- T is a finite set of transitions;
- B is a finite set of Boolean signals;
- V is a finite set of continuous variables;

- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation;
- L is a tuple of labels defined below;
- $M_0 \subseteq P$ is the set of initially marked places;
- S_0 is the set of initial Boolean signal values;
- Q_0 is the set of initial ranges of values for each continuous variable; and
- R_0 is the set of initial ranges of rates for each continuous variable.

A key component of LHPNs are the labels. The labels permitted in LHPNs are represented using a tuple $L = \langle En, D, BA, VA, RA \rangle$:

- $En : T \rightarrow \mathcal{P}$ labels each transition $t \in T$ with an enabling condition;
- $D : T \rightarrow |\mathbb{Q}| \times (|\mathbb{Q}| \cup \{\infty\})$ labels each transition $t \in T$ with a lower and upper bound $[d_\mathrm{l}, d_\mathrm{u}]$ on the delay for t to fire;
- $BA : T \times B \rightarrow \{0, 1, \mathbf{unc}\}$ la bels each transition $t \in T$ and Boolean variable $b \in B$ with the Boolean assignment made to b when t fires.
- $VA : T \times V \rightarrow (\mathbb{Q} \times \mathbb{Q}) \cup \{\mathbf{unc}\}$ labels each transition $t \in T$ and continuous varia ble $v \in V$ with the continuous variable assignment, specified as a range of values $[a_l(t, v), a_u(t, v)]$, that is made to v when t fires.
- $RA : T \times V \rightarrow (\mathbb{Q} \times \mathbb{Q}) \cup \{\mathbf{unc}\}$ labels each transition $t \in T$ and continuous vari able $v \in V$ with the continuous rate assignment, specified as a range of values $[r_l(t, v), r_u(t, v)]$, that is made to v when t fires.

Assignments to **unc** leave the value unchanged. These assignments are not represented in the graphical representation.

The enabling condition is defined using a restricted set of *hybrid separation logic* (HSL) formulas from the set $\mathcal{P}$ which are a Boolean combination of Boolean variables and separation predicates (inequalities relating continuous variables to constants). These formulas satisfy the following grammar:

$$\phi \quad ::= \quad \mathbf{true} \mid \mathbf{false} \mid b_i \mid \neg\phi \mid \phi \wedge \phi \mid v_i \geq k_i$$

where b_i is a Boolean signal, v_i is a continuous variable, and k_i is a rational constant in $\mathbb{Q}$. Since non-strict inequalities are not supported by the DBM-based state space exploration analyzer, the negation of $\geq$ inequalities represent $\leq$ inequalities.

The semantics of the LHPN model are briefly illustrated using an LHPN model of the switched capacitor integrator shown in Figure 3. A formal description of the semantics for LHPNs can be found in [20]. The output voltage, V_out, is modeled by the LHPN shown in Figure 3a. The rate of the output voltage changes based on the value of V_out and the input voltage. The square wave input voltage, V_in, is modeled using the LHPN shown in Figure 3b. V_in is modeled as a stable, multi-valued continuous quantity. Stable, multi-valued continuous quantities are modeled using continuous variables with a rate of zero and are updated using a variable assignment after a time delay. The LHPN shown in Figure 3c is used to detect a failure. The enabling condition on the transition is the negation of an HSL formula for the safety property being verified. When this transition is enabled and fires, a failure is detected. In the initial state, p_0, p_1, and p_6 are marked; *fail* is **false**; V_{out} is -1000 mV; V_{in} is -1000 mV; the rate of V_{in} is 0; and the rate of V_{out} is 17 to 24 mV/μs. Initially, t_1 is the only enabled transition. However, as time passes, V_{out} crosses 0 V enabling t_6 which fires immediately moving the token from p_6 to p_3. After 100 to 101 μs from the initial state, t_1 fires and sets V_{in} to 1000 mV. This change on V_{in} enables transition t_3 which fires immediately and sets the rate of V_{out} to be between -24 and -17 mV/μs. Transition t_4 fires next in zero time when $V_{\text{out}} < 0\ V$. After this firing, transition t_2 fires after being enabled 99 to 100 μs. This firing sets V_{in} to -1000 mV and enables transition t_5 which fires immediately and sets the rate of V_{out} to be between 17 and 24 mV/μs. This behavior continues until the range of V_{out} enables transition t_0 which fires and sets *fail* to **true**.

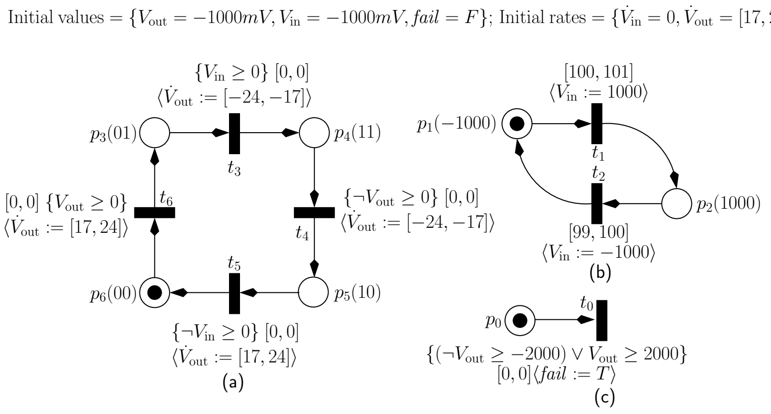
Initial values = $\{V_{\text{out}} = -1000mV, V_{\text{in}} = -1000mV, fail = F\}$; Initial rates = $\{\dot{V}_{\text{in}} = 0, \dot{V}_{\text{out}} = [17, 24]\}$
$\{V_{\text{in}} \geq 0\}$ [0, 0]
$\langle \dot{V}_{\text{out}} := [-24, -17] \rangle$
$p_3(01)$
t_3
$p_4(11)$
[0, 0] $\{V_{\text{out}} \geq 0\}$
$\langle \dot{V}_{\text{out}} := [17, 24] \rangle$
t_6
t_4
$\{\neg V_{\text{out}} \geq 0\}$ [0, 0]
$\langle \dot{V}_{\text{out}} := [-24, -17] \rangle$
t_5
$p_6(00)$
$p_5(10)$
$\{\neg V_{\text{in}} \geq 0\}$ [0, 0]
$\langle \dot{V}_{\text{out}} := [17, 24] \rangle$
(a)
[100, 101]
$\langle V_{\text{in}} := 1000 \rangle$
$p_1(-1000)$
t_1
t_2
$p_2(1000)$
[99, 100]
$\langle V_{\text{in}} := -1000 \rangle$
(b)
p_0
t_0
$\{(\neg V_{\text{out}} \geq -2000) \vee V_{\text{out}} \geq 2000\}$
$[0, 0]\langle fail := T \rangle$
(c)

Fig. 3. A simple LHPN example for the switched capacitor integrator. (a) V_{out}, (b) V_{in}, and (c) property portions of the LHPN.

4. Abstract Model Generation

Our modeling work differs from the previous work in several ways. The most pronounced differences are the accuracy of the abstract model and the use of non-determinism. Previous methods attempt to abstract the model while maintaining transistor-level accuracy. The abstract models produced by `LEMA`'s model generator do not attempt to maintain this level of accuracy but do model ranges of parameters and conditions using non-determinism. They are intended to be used in system-level simulations to verify properties such as connectivity between the digital and analog circuits or for use in formal verification. As a result, these models are less general, but the model generation and simulations using these models are more efficient.

During the course of traditional analog circuit verification, designers run many different simulations to verify that the circuit meets its specification. The model generator's goal is to automatically generate abstract models from this simulation data. The generated circuit models are conservative and model all the provided simulation traces plus additional behavior. By using simulations already produced by the designer, no additional simulation time is required. However, the quality of the model is directly related to the simulations used to create it. If the designer has inadequately simulated the design, the model may not exhibit the full behavior of the system. In this case, there is a potential that the actual circuit may have a failing behavior that is not included in the generated model. To help address this issue, Section 5 discusses the use of coverage metrics.

This section first describes the algorithms to construct an LHPN model. Next, it describes methods to produce a normalized LHPN model for formal verification as well as VHDL-AMS and Verilog-AMS models for system-level simulation.

4.1. *Model generation algorithms*

Two simulations of the switched capacitor integrator are used to construct the model for this example. In particular, the switched capacitor integrator is simulated with capacitance values of 23 pF and 27 pF for capacitor C_2. The simulation data is recorded for the nodes representing the input voltage, V_{in}, and output voltage, V_{out}, during a 400 μs transient simulation for each capacitance value. Part of the data from the 23 pF simulation is shown in the first three columns of Table 1.

Table 1. Simulation data with $C_2 = 23$ pF for the switched capacitor integrator.

Time (μs)	V_{in} (mV)	V_{out} (mV)	Region	$\Delta V_{\text{in}}/\Delta t$ (mV/μs)	$\Delta V_{\text{out}}/\Delta t$ (mV/μs)
0.00	-1000	-1000	00	-40.07	21.85
0.51	-1000	-999	00	0.0	21.74
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
28.52	-1000	-391	00	0.0	23.74
32.00	-1000	-304	00	-	-
35.01	-1000	-217	00	-	-
38.51	-1000	-174	00	-	-
41.54	-1000	-87	00	-	-
45.00	-1000	5	01	0.0	21.72
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
100.62	-520	1176	01	-	-
100.78	120	1176	11	275.00	-21.08
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
400.00	1000	-957	10	-	-

Algorithm 1 describes the process of taking simulation data and generating an abstract model. The inputs to the algorithm are *Var*, Σ, θ, $prop_{\text{HSL}}$, *ws*, ϵ, *ratio*, $\tau_{\min}$, *nonC*, *sig*, and *sep*. Each variable $\nu \in Var$ is a design variable in the system being modeled. Σ is the set of time series simulation traces. Each trace $\sigma \in \Sigma$ is an n-tuple $\langle \tau, \nu_0, \ldots, \nu_n \rangle$ where $\tau \in \mathbb{R}$ is the timestamp for the data points $(\nu_0, \ldots, \nu_n) \in \mathbb{R}^{|\mathcal{V}|}$ where $\nu_i \in Var$ and n is $|Var|$. For example, in Table 1 the first column is τ; the second column is ν_0; and the third column is ν_1. To access the timestamp for data point i the notation $\sigma_i(\tau)$ is used. Similarly, to access the data value i for variable ν the notation $\sigma_i(\nu)$ is used. In Table 1, $\sigma_1(\tau)$ is 0.51 μs and $\sigma_1(V_{\text{out}})$ is -999 mV. θ is the set of thresholds on the signal levels of the design variables in *Var*. Increasing the number of thresholds increases both the complexity and accuracy of the model. The thresholds, θ, for each variable ν are $\langle\theta_0(\nu), \ldots, \theta_m(\nu)\rangle$ where $\theta_0(\nu)$ is $-\infty$ and $\theta_m(\nu)$ is ∞. The thresholds are used to group the simulation data into regions ξ, where $\xi_i(\nu) = [\theta_i(\nu), \theta_{i+1}(\nu)]$, and do not need to be equidistant. The lowest region for a variable is $\xi_0(\nu)$ and the highest region is $\xi_{m-1}(\nu)$. $prop_{\text{HSL}}$ is a safety property specified using a restricted HSL formula. The remaining parameters to `genModel`, *ws*, ϵ, *ratio*, $\tau_{\min}$, *nonC*, *sig*, and *sep*, are optional parameters that can be specified to configure the model generation process. *ws* is the window size used in rate calculation and has a default value of 200 lines. ϵ, *ratio*, and $\tau_{\min}$ are used in the detection of digital-like signals and have default values of 0.1, 0.8, and 5e-6, respectively. *nonC* is a function that maps a signal to a set of non-causal signals (i.e., $nonC\colon Var \rightarrow 2^{Var}$). *sig* and *sep* are used in the normalization process and have default values of 2 and 1, respectively.

Algorithm 1: `genModel`($Var, \Sigma, \theta, prop_{\text{HSL}}, ws, \epsilon, ratio, \tau_{\min}, nonC, sig, sep$)

```
1 $N :=$ null;
2 forall $\sigma \in \Sigma$ do
3     $reg :=$ assignRegions($\sigma$, $Var, \theta$);
4     $rate :=$ calculateRates($\sigma$, $Var, ws, reg$);
5     $(dmv, start, end) :=$ detectDMV($\sigma$, $Var, \epsilon, ratio, \tau_{\min}$);
6     $N :=$ updateLHPN($N$, $Var, \sigma, reg, rate, dmv, start, end, nonC, \theta$);
7 writeNormalizedLHPN($N, prop_{\text{HSL}}, sig, sep$);
8 writeVHDLAMS($N, prop_{\text{HSL}}$);
9 writeVerilogAMS($N$);
```

In Algorithm 1, the value null is used to initialize non-set based data types to an initial value (line 1). Each data point for each variable for each simulation trace $\sigma_i(\nu)$ is given a region assignment reg_i based on the thresholds (lines 2-3). The variable's region assignment at data point i is accessed using the notation $reg_i(\nu)$ and is an integer value between 0 and $|\xi(\nu)|$. In Table 1, the fourth column represents the region assignment, so $reg_{|\sigma_i|}$ is 10; $reg_{|\sigma_i|}(V_{\text{in}})$ is 1. Next, ranges of rates are calculated for each continuous variable within each region (line 4). The algorithm assumes nothing about the dependence or independence of the rates. Each rate is calculated individually for each region for each simulation trace. The variable's rate assignment at data point i is accessed using the notation $rate_i(\nu)$. In Table 1, the fifth column represents the rate assignment for V_{in}, so $rate_0(V_{\text{in}})$ is -40.07 mV/μs. It is expected that the rates change during different phases of operation. For this reason, it is important that thresholds are selected to separate the different phases of operation into distinct regions. At this point, continuous variables which are mostly stable but occasionally change are identified as variables that can be approximated by discrete transitions, *discrete multi-valued* (DMV) variables (line 5). The variable *dmv* is the set of variables detected as DMV variables. All of this information is collected for the current simulation trace and combined with the information from other simulation traces in the LHPN representing the system model (line 6). Finally, the abstract models are generated (lines 7-9).

The first step of Algorithm 1 is to assign the data to regions using the user-provided thresholds (line 3). In this example, the thresholds for both V_{in} and V_{out} are 0 V. Algorithm 2 analyzes each variable in the system at each time point in a simulation trace to assign the appropriate region encoding (lines 1-2). The region value is assigned based upon the location of the data point relative to the threshold values for the given variable (lines 3-4). In the data shown in Table 1, each digit in the fourth column represents the region. For instance, at time 100.62 μs in Table 1, the region assigned is $\langle 01 \rangle$ indicating that V_{in} is below 0 V and V_{out} is above 0 V. When V_{in} moves above 0 V at time 100.78 μs, the region is $\langle 11 \rangle$.

Algorithm 2: `assignRegions`(σ, *Var*, θ)

```
1 forall $\nu \in Var$ do
2     forall $i \in [0, |\sigma|]$ do
3         forall $j \in [0, |\xi(\nu)|]$ do
4             if $\sigma_i(\nu) \in \xi_j(\nu)$ then $reg_i(\nu) := j$;
5 return $reg$;
```

After regions have been assigned to each data point, the rates are calculated for each region using Algorithm 3, `calculateRates`. A rate is calculated for each eligible data point in the trace (line 1). Not all points are eligible for rate calculations due to a low pass filtering technique used to smooth edge effects caused by region boundaries and transitory pulses. The low pass filtering uses a sliding window approach that works by calculating the rate of change for a variable between the current point and a point *ws* points further in time if all points between are in the same region (lines 2–3). For example, in Table 1, using a value of four for *ws*, the rate of change for V_{out} at time 28.52 μs in Table 1 is calculated by looking at its value at this time point and the value four points later, 41.54 μs. This value is determined to be 23.74 mV/μs. When the algorithm moves to calculate the rate for the next point, 32.00 μs, it finds that the data point four points later is in a different region and does not calculate a rate for the 32.00 μs point.

Algorithm 3: `calculateRates`(σ, *Var*, *ws*, *reg*)

```
1 forall $i \in [0, |\sigma| - ws]$ do
2     if $\forall j \in [0, ws].reg_i = reg_j$ then
3         forall $\nu \in Var$ do $rate_i(\nu) := (\sigma_{i+ws}(\nu) - \sigma_i(\nu))/(\sigma_{i+ws}(\tau) - \sigma_i(\tau))$;
4 return $rate$;
```

In AMS designs, it is expected that digital signals are present. To take advantage of the digital abstraction and reduce analysis complexity, digital-like signals are detected and modeled discretely. Instead of allowing them to change with a specific rate, a constant value can be directly assigned to the variable at a specified time after entering a region. A DMV variable is detected when it remains constant for a specified ratio of time with respect to the total time for the simulation. Remaining constant is defined as staying within an ϵ bound for a minimum time, $\tau_{\min}$. Algorithm 4 describes the DMV detection algorithm, `detectDMV`. The algorithm tests each variable in the trace separately (line 2). The analysis begins with the first point and checks to see if the second point is equivalent within the specified ϵ bound (lines 3-6). If it is within the ϵ bound, 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 (line 7). If this time, $\sigma_j(\tau) - \sigma_i(\tau)$, is greater than $\tau_{\min}$, the value is added to the running total of constant time, t_{const} (lines 7-8). The start and end points for the constant run of the variable ν are also recorded (lines 9-10). When all points have been analyzed, the ratio of constant time, τ_{const}, to total time for the trace, $\sigma_{|\sigma|}(\tau)$, is calculated. If this ratio exceeds the specified *ratio*, the variable is added to the set of DMV variables (line 11). In the switched capacitor integrator example, the square wave input voltage, V_{in}, is an example of a DMV variable. This can be inferred from Table 1 as V_{in} is largely constant.

Algorithm 4: `detectDMV`(σ, $Var, \epsilon, ratio, \tau_{\min}$)

```
1  $start, end = \emptyset$;
2  forall $\nu \in Var$ do
3      $i, j, \tau_{\text{const}} := 0$;
4      while $i < |\sigma|$ do
5          $i := j$;
6          while $(|\sigma_i(\nu) - \sigma_{j+1}(\nu)| \leq \frac{\epsilon}{2} \wedge j < |\sigma|)$ do  $j := j + 1$;
7          if $(\sigma_j(\tau) - \sigma_i(\tau)) \geq \tau_{\min}$ then
8              $\tau_{\text{const}} := \tau_{\text{const}} + (\sigma_j(\tau) - \sigma_i(\tau))$;
9              $start(\nu) = start(\nu) \cup i$;
10             $end(\nu) = end(\nu) \cup j$;
11     if $(\tau_{\text{const}}/\sigma_{|\sigma|}(\tau)) \geq ratio$ then  $dmv := dmv \cup \nu$;
12 return $(dmv, start, end)$;
```

After the needed information has been calculated for a trace, `updateLHPN` shown in Algorithm 5 updates the LHPN, N, with the new information. Algorithm 6, the `updateRegionGraphs` function, examines each region in the simulation trace adding new regions and updates the rates of existing regions. The `updateDMVGraphs` function shown in Algorithm 7 adds information to the graphs for DMV variables (line 2). The initial value, rate, and region for each variable in each simulation trace are also recorded in the LHPN, N (lines 3-6).

Algorithm 6 (`updateRegionGraphs`) updates the LHPN with region information from each simulation trace. The notation $p(reg_i)$ returns the place representing reg_i. Similarly, the notation $t(reg_i, reg_j)$ returns the transition between the place

Algorithm 5: `updateLHPN`(N, $Var, \sigma, reg, rate, dmv, start, end, nonC, \theta$)

```
1 updateRegionGraphs($N, \sigma, reg, rate, \theta$);
2 updateDMVGraphs($N, \sigma, reg, dmv, start, end, nonC$);
3 forall $\nu \in Var$ do
4     $Q_0(N)(\nu) := Q_0(N)(\nu) \cup \sigma_0(\nu)$;
5     $R_0(N)(\nu) := R_0(N)(\nu) \cup rate_0(\nu)$;
6     $M_0(N)(\nu) := M_0(N)(\nu) \cup p(reg_0(\nu))$;
7 return $N$;
```

representing reg_i and reg_j. The `newT` function updates the flow relation, delay bounds, enabling condition, and rate assignments for the LHPN before returning the newly created transition. The **let** notation is used for convenience to indicate the contents of a tuple (line 1). The algorithm begins by updating the regions for each data point in the given trace (line 3). A node in the graph is added for the new region if the region has not been found previously (line 4). For example, Fig. 3 contains four places $p_3 - p_6$ representing the four different regions discovered, 00, 01, 11, and 10. While in this example a place is created for every possible region assignment, in larger examples, many regions may never be encountered during simulation. Places are not generated for these unreached regions. If not previously seen, transitions between the current region and the previous region are created using `newT` and added to the set of transitions (lines 5-6). It is theoretically possible that this process could result in a fully connected graph, but in practice this is highly unlikely. In Fig. 3, there are four of the possible twelve connections (excluding self-loops). The delay for the transition is set to [0,0] to make it fire immediately as the state of the system moves from one region to the next. The `diffR` function determines the differences between the two regions for use in generating the enabling condition. Each transition is given an enabling condition representing the threshold that is being crossed. In the move from 00 to 01 the enabling condition is $\{V_{\text{in}} \geq 0\}$ on t_3. The rate of change for the continuous variables in each region is recorded if it is outside the current range (lines 7-9). These rates are stored on the transitions between regions as shown in Fig. 3.

Algorithm 6: `updateRegionGraphs`($N, \sigma, reg, rate, \theta$)

```
1 let $N = (P, T, B, V, F, En, D, BA, VA, RA, M_0, S_0, Q_0, R_0)$;
2 $reg_{prev}$ := null;
3 forall $i \in [0, |\sigma|]$ do
4     if $p(reg_i) \notin P$ then $P := P \cup$ newP($reg_i$);
5     if $t(p(reg_{prev}), p(reg_i)) \notin T)$ then
6         $T := T \cup$ newT($N, p(reg_i), p(reg_{prev}), (0,0), \theta,$ diffR($reg_i, reg_{prev}$));
7     forall $t \in T$ do
8         if $(t, p(reg_i)) \in F$ and $rate_i < r_l(t)$ then $r_l(t) := rate_i$;
9         if $(t, p(reg_i)) \in F$ and $rate_i > r_u(t)$ then $r_u(t) := rate_i$;
10    $reg_{prev} := reg_i$;
```

Algorithm 7 (`updateDMVGraphs`) updates the LHPN with DMV variable information from each simulation trace. The algorithm begins by initializing the previous value variable and looping through each DMV variable (lines 2-3). Given a DMV variable ν, the algorithm begins with the first time value in the *start* set (lines 4-5). When a start point is found, the algorithm then searches for the next end point (lines 6-7). Once the start and end points are found, the enabling region and delay range are calculated, and the range of values for the given variable in the given constant run are extracted (lines 8-10). To determine the enabling region, the causal event set must be determined (line 9). The causal event set is determined by finding the region previous to the one where the constant run begins. The set of variables that remain constant between the regions is the causal set. If no variables are constant, then the previous region is examined, etc. The delay range is calculated using the `calcD` function (line 10). The lower bound of the range is calculated from the time the trace enters the previous region until the previous constant run ends. The upper bound of the range is calculated from the previous region until the start of the current constant run. This range allows for the variable to change in the uncertain region between constant runs. V_{in} is a DMV variable in the switched capacitor integrator example and changes to its high value in region 11. The previous region is 01. In this case, V_{in} would be assigned to change to its high value based on the time V_{out} crosses above the threshold of zero. This simplistic causal calculation may result in the incorrect or inconsistent choice for a causal event. For example, although a change on V_{out} may appear to cause V_{in}, this is not the case. This choice would result in a poor model as the rate of change of V_{out} would affect the time when V_{in} changes making V_{in} much less periodic. Therefore, a refinement to this first method is used. The user can specify the fact that there is not a causal relationship between two variables in *nonC*. As a result, these non-causal variables are ignored in the causal region calculation. If V_{out} is specified as non-causal in the previous example, V_{in} is the only remaining variable. If V_{in} is causal only with itself then the delay value is the amount of time V_{in} remains at a given value. The value range is calculated by extracting the minimum and maximum values in the constant run. If a place does not already exist for this value range, then a new one is created (line 12). For the integrator, place p_1 is added for V_{in} equal to $[-1000,-999]$ mV, and p_2 is added to represent that V_{in} is equal to $[999,1000]$ mV. The next step is to create a transition between the current place and the previous place if one does not exist (lines 13-14). Finally, the value and delay assignments are updated (lines 15-16).

4.2. *Creating models*

This section describes methods to create models for formal verification and system-level simulation. Since `LEMA`'s DBM-based analyzer only uses integers, the `writeNormalizedLHPN` function must normalize the values in the LHPN. The normalization process begins by scaling the minimum rate such that its integer value is represented using *sig* values. For instance, if *sig* is two and the minimum rate is 0.02 then all rates would be scaled by a factor of 1000 resulting in 0.02 being

Algorithm 7: `updateDMVGraphs`$(N, \sigma, reg, dmv, start, end, nonC, \theta)$

```
1  let $N = (P, T, B, V, F, En, D, BA, VA, RA, M_0, S_0, Q_0, R_0)$;
2  $val_{prev}$ := null;
3  forall $\nu \in dmv$ do
4      while $start(\nu) \neq \emptyset$ do
5          $i = \texttt{min}(start(\nu))$;
6          $start(\nu) = start(\nu) - i$;
7          $j = \texttt{min}(end(\nu))$;
8          $end(\nu) = end(\nu) - j$;
9          $reg_{En} := \texttt{calcEn}(\sigma, i, nonC(\nu))$;
10         $(\tau_l, \tau_u) := \texttt{calcD}(\sigma, i, j, reg_{En}, end)$;
11         $(val_l, val_u) = val := \texttt{extractVals}(\sigma, \nu, i, j)$;
12         if $p(val) \notin P$ then $P := P \cup \texttt{newP}(val)$;
13         if $t(val_{prev}, val) \notin T$ then
14             $T := T \cup \texttt{newT}(N, p(val_{prev}, val), (\tau_l, \tau_u), \theta, \texttt{diffR}(reg_{En}, reg_i))$;
15         $\texttt{updateAsgVal}(val_l, val_u, t(val_{prev}, val))$;
16         $\texttt{updateDelayVal}(\tau_l, \tau_u, t(val_{prev}, val))$;
17         $val_{prev} := val$;
```

a rate of 20. If this process results in the maximum rate overflowing the integer space, `LEMA` reports an error and terminates. The next step is to adjust the constant values so that at least one integral time unit passes as a variable progresses between the thresholds at the new rates. The scaling of the constants involves scaling both thresholds and constant values for DMV variables such that there are *sep* orders of magnitude between the rates and constants. For instance, if the rate for a continuous variable is 20 and the thresholds are 0 and 1, the variable would pass between the thresholds in less than one time unit which poses a problem for the integer-based analysis. In this case, if the value of *sep* is one, the constants would be scaled to 0 and 100. This would now require five time units for the variable to pass between the thresholds. This function also adds to the LHPN a single initially marked place and a single transition for the safety property. The transition's enabling condition is the complement of the safety property. This transition has a delay of [0,0] and indicates a failure when it fires by setting the Boolean signal *fail* to **true**. Therefore, to verify this safety property, a model checker only needs to determine if there exists any state in which this transition can fire. For the integrator example, the LHPN generated to check for saturation is shown in Fig. 3c.

The user can classify a system variable as one of three types: input, output, or internal. Input variables are not modeled by the system, but the resulting model contains an input port where the system expects to receive external input for this variable (e.g., from a test bench). Output variables are modeled and assigned to an output port. Internal variables are used in the model but no input or output ports are provided. Any unclassified variables are unmodeled and not used in the model generation process. If all variables are marked as internal, the model is self-contained and can be simulated without the aid of an external test bench.

Otherwise, the abstract model is intended to replace the transistor-level model which contains the same inputs and outputs as the original circuit. `LEMA`'s model generator can also produce both deterministic and non-deterministic models.

Figure 4 is the VHDL-AMS for a self-contained non-deterministic model in which `Vin` and `Vout` are marked internal. The creation of a VHDL-AMS description begins by creating a real **quantity** for each internal variable in the system. In this case, real variables are created for `Vin` and `Vout`. Each of these variables are assigned an initial value using a **break** statement which is -1000 mV for both `Vin` and `Vout`. DMV variables, `Vin` in this case, are assigned an initial rate of 0.0 using the `'dot` notation, and it is never changed. The rates for non-DMV variables are set with nested **if-use** statements based upon the threshold values for each region. For instance, the **if** statement models region 00 where both `Vin` and `Vout` are below zero. The rate is set to $[17, 24]$ is this region. The VHDL-AMS description supports ranges of rates using the `span` procedure that accepts two real values and returns a random value within that range. The constant value assignments for DMV variables are specified using **process** statements without sensitivity lists. The **wait** statement waits for the proper Boolean condition and then waits for a range of delays before performing the assignment. In this example, the Boolean condition is implicit, so `Vin` is assigned to 1000 mV in 100 to 101 μs after it goes low and then assigned a value of -1000 mV in 99 to 100 μs after it goes high. Finally, **assert** statements are used to describe basic safety properties about the system using the restricted HSL grammar presented previously. For this example, the **assert** statement is used to check if *Vout* falls below -2000 mV or goes above 2000 mV.

The Verilog-AMS for a deterministic model of the switched capacitor integrator circuit where `Vin` is marked as an input and `Vout` is marked as an output is shown in Figure 5. Model generation begins by creating a top level **inout** variable for each input or output variable in the graph. `Vin_io` and `Vout_io` are the top level variables. The **real** variables are created to hold the current value of each output or internal variable, `Vout_var` in Fig. 5. Variables with a rate 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 extracted from the nodes between the edge. One **cross** statement is created for an edge where V_{in} changes from 1000 mV to -1000 mV by crossing the 0 V 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 `Vout` to `0.020`, so this assignment is made to `Vout_rate` in the execution block of the **cross** statement. A global timer is added to update all rates at an appropriate interval. For the switched capacitor integrator, an interval of 1 μs is used to update the value of `Vout_var` based on `Vout_rate`. Finally, a **transition** statement is added for each output variable to quickly transition the value of the internal variable to the external interface. Note that to make the model deterministic, all ranges are averaged.

```
architecture behavioral of swCap is
  quantity Vin, Vout:real;
begin
  break Vin => -1000.0, Vout => -1000.0;
  Vin'dot == 0.0;
  if not Vin'above(0.0) and not Vout'above(0.0) use
    Vout'dot == span(17.0,24.0);
  elsif not Vin'above(0.0) and Vout'above(0.0) use
    Vout'dot == span(17.0,24.0);
  elsif Vin'above(0.0) and Vout'above(0.0) use
    Vout'dot == span(-24.0,-17.0);
  elsif Vin'above(0.0) and not Vout'above(0.0) use
    Vout'dot == span(-24.0,-17.0); end use;
  process begin
    wait for delay(100,101);
    break Vin => 1000;
    wait for Vin'above(0.0);
    wait for delay(99,100);
    break Vin => -1000;
    wait for Vin'above(0.0);
  end process;
  assert (Vout'above(-2000.0) and not Vout'above(2000.0))
    report "Error:  Saturation." severity failure;
end behavioral;
```

Fig. 4. Non-deterministic VHDL-AMS model for the switched capacitor integrator circuit generated when both Vin and Vout are marked as internal signals.

```
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. 5. Deterministic Verilog-AMS model of the switched capacitor integrator circuit generated when Vin is marked as an input and Vout is marked as an output.

5. Coverage Metrics

Coverage information can be extracted from a set of simulations and is key to LEMA's abstract model generation methodology. Our method includes a basic coverage metric where each simulation trace is given a score and uncrossed thresholds are reported. A higher score is achieved by a simulation trace that exhibits behavior not previously seen. A metric of this type gives a qualitative measure of the utility of an additional simulation trace. This type of metric could be used as an aid to determine the benefit of doing further simulations. Uncrossed thresholds potentially indicate an inadequate simulation set as the thresholds should characterize the operating regions of the system. The simulation set should provide information for all operating regions of the system. The coverage score for a given simulation is calculated using the following formula with unity weights $cvg := p \cdot w_p + t \cdot w_t + c \cdot w_c + d \cdot w_d$ where p is the number of new places, w_p is the places weight, t is the number of new transitions, w_t is the transitions weight, c is the number of times a range of rate or constant value is updated, w_c is the rates and constant values weight, d is the number of times a delay range is updated, and w_d is the delay weight.

For the integrator example, using just the simulation trace shown in Table 1 with C_2 equal to 23 pF would result in an LHPN with the same structure but different rates from the LHPN shown in Figure 3 and produce a coverage score of 200 using weights of one. Adding the simulation trace with C_2 equal to 27 pF results in the exact same LHPN structure, but the ranges of rate for V_{out} would be changed. Therefore, the value of the second trace run is only 94. Finally, if a third trace with C_2 equal to 25 pF is added at this point, the resulting LHPN would not change at all as the rates generated from this trace would be contained in those generated from the first two. Therefore, this trace adds no new knowledge, so the coverage metric would say that it has no value.

6. Case Studies

Using LEMA's model generator, two simulation traces of the switched capacitor integrator result in the LHPN shown in Fig. 3. Although neither of the simulation traces indicate a problem with saturation of the integrator, a state space analysis using the DBM model checker finds a potential circuit failure in less than a second. This failure can occur when the integrator charges the capacitor, C_2, at a rate that is on average faster than the rate of discharge. This situation causes charge to build up on the capacitor and eventually results in V_{out} reaching a voltage above 2000 mV. The reason that this method can find this failure is that the LHPN model represents not only each simulation trace, but also the union of the traces.

Saturation of the integrator can be prevented using the circuit shown in Figure 6. In this circuit, a 4 MΩ resistor in the form of a switched capacitor is inserted in parallel with the feedback capacitor. This causes V_{out} to drift back to 0 V. In other words, if V_{out} is increasing, it increases faster when it is far below 0 V than when it is near or above 0 V. Using the same simulation parameters and thresholds for this circuit, the model generator obtains an LHPN with the same structure as the one

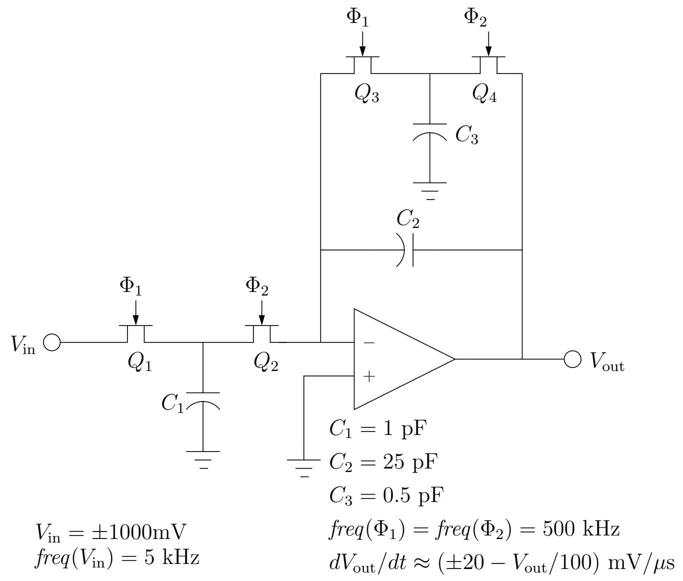
Φ_1
Φ_2
Q_3
Q_4
C_3
C_2
Φ_1
Φ_2
V_{in}
Q_1
Q_2
C_1
$-$
$+$
V_{out}
$C_1 = 1$ pF
$C_2 = 25$ pF
$C_3 = 0.5$ pF
$V_{\text{in}} = \pm 1000\text{mV}$
$freq(V_{\text{in}}) = 5$ kHz
$freq(\Phi_1) = freq(\Phi_2) = 500$ kHz
$dV_{\text{out}}/dt \approx (\pm 20 - V_{\text{out}}/100)$ mV/μs

Fig. 6. Circuit diagram of a corrected switched capacitor integrator.

shown in Fig. 3, but the ranges of rates for each region are [18,32] for $\langle 00 \rangle$, [9,22] for $\langle 01 \rangle$, [-22,-9] for $\langle 11 \rangle$, and [-32,-18] for $\langle 10 \rangle$. This LHPN fails verification as the thresholds are too simple to capture the effect of the additional switched capacitor. Because the rate of change is now dependent on the value of V_{out}, the thresholds on V_{out} are changed to -500 mV, 0 V, and 500 mV. The LHPN for these new thresholds is found to satisfy the property in less than a second.

Phase locked loops (PLLs) are notoriously difficult circuits to design and validate. A PLL is typically composed of a phase detector, a low pass filter, a VCO, and a frequency divider. 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 clock 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. 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 using two 1 μs transient simulations of the piecewise linear input and a periodic clock input. Eight variables are required for model generation. Four of the signals are the inputs and outputs while the remaining four signals internal signals associated with 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 $\frac{V_{\mathrm{dd}}}{2}$. Comparison between simulation times for the transistor-level design and the Verilog-AMS model are performed using the same simulation inputs and simulator. Table 2 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 Fig. 7. There is a difference between the two waveforms, but the abstracted model is accurate enough to be used in a system-level simulation.

Table 2. Simulation times for the transistor-level model and the Verilog-AMS model.

Sim	Verilog-AMS (s)	Transistor (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

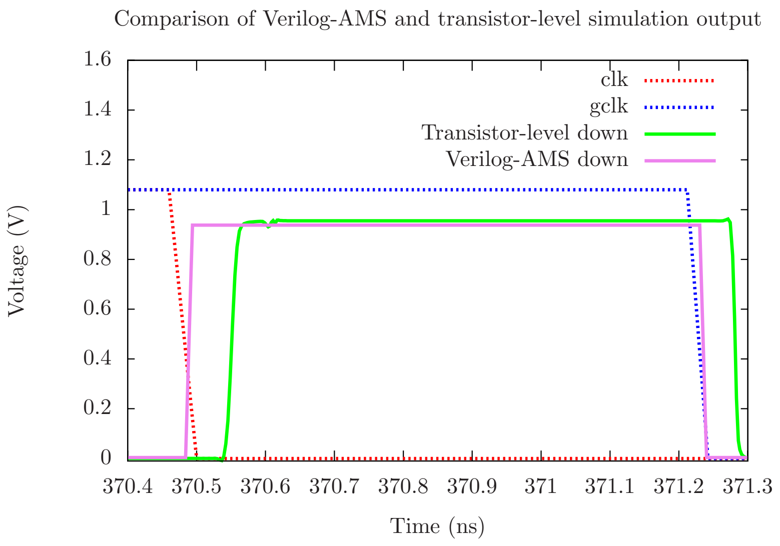

Fig. 7. Simulation traces for the transistor-level and Verilog-AMS models.

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 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 8. This pseudocode is a behavioral description of the correct input/output operation of the phase detector. This property cannot 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. 8. A property to verify for the PLL phase detector.

7. Conclusion

Interest in formal and semi-formal methods for validating AMS circuits is increasing. Many of these methods are seriously handicapped by the difficulty of generating formal models. This paper develops a method to generate a conservative formal LHPN model from a set of simulation traces and thresholds on the state space. This LHPN model can be used by several different model checking engines to prove safety properties about the entire state space of the model. Using two variations of the switched capacitor integrator circuit, this paper shows how an adequate LHPN model can be created using two simulation traces and a basic set of thresholds. The model is analyzed using a DBM based model checker to obtain the expected verification results. Another benefit of the method described in this paper is that an LHPN model can be translated into a VHDL-AMS or Verilog-AMS model. One problem for AMS designers is creation of abstract models of their circuit for use in digital or mixed-mode simulation flows. Models can be created by hand but must be updated to remain consistent as circuits change. Using this method, models can maintain their consistency by running the needed set of simulations after changes and regenerating the HDL model from the simulations.

Future work includes demonstrating an improved model generation algorithm on larger examples that combine both analog and digital components. While method described in this paper requires the user to provide thresholds, initial investigations into the auto-generation of thresholds are underway. This auto-generation of thresholds can potentially use input from failure traces and coverage metrics to improve the model in an abstraction refinement loop. Finally, more expressive property languages are being investigated for AMS circuits.

References

1. R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1992.
2. T. Dang, A. Donzé, and O. Maler. Verification of analog and mixed-signal circuits using hybrid systems techniques. In A. J. Hu and A. K. Martin, editors, *Formal Methods for CAD*, volume 3312 of *LNCS*, pages 21–36. Springer, 2004.
3. T. Dang and T. Nahhal. Randomized simulation of hybrid systems for circuit validation. Technical report, VERIMAG, May 2006.

4. T. R. Dastidar and P. P. Chakrabarti. A verification system for transient response of analog circuits using model checking. In *VLSI Design*, pages 195–200. IEEE Computer Society Press, 2005.
5. R. David and H. Alla. On hybrid petri nets. *Discrete Event Dynamic Systems: Theory and Applications*, 11(1–2):9–40, Jan. 2001.
6. A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. In A. Bemporad, A. Bicchi, and G. Buttazzo, editors, *Hybrid Systems: Computation and Control*, volume 4416 of *LNCS*, pages 174–189. Springer, 2007.
7. G. E. Fainekos, A. Girard, and G. J. Pappas. Temporal logic verification using simulation. In E. Asarin and P. Bouyer, editors, *Formal Modelling and Analysis of Timed Systems*, volume 4202 of *LNCS*, pages 171–186. Springer, 2006.
8. G. Frehse, B. H. Krogh, and R. A. Rutenbar. Verifying analog oscillator circuits using forward/backward refinement. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 257–262. IEEE Computer Society Press, 2006.
9. A. Girard and G. J. Pappas. Verification using simulation. In J. P. Hespanha and A. Tiwari, editors, *Hybrid Systems: Computation and Control*, volume 3927 of *LNCS*, pages 272–286. Springer, 2006.
10. W. Hartong, L. Hedrich, and E. Barke. On discrete modeling and model checking for nonlinear analog systems. In E. Brinksma and K. G. Larsen, editors, *Proc. International Conference on Computer Aided Verification*, volume 2404 of *LNCS*, pages 401–413. Springer, 2002.
11. K. D. Jones, V. Konrad, and D. Nickovic. Analog property checkers: A DDR2 case study. In *Formal Verification of Analog Circuits (FAC)*, 2008.
12. S. Little. *Efficient Modeling and Verification of Analog/Mixed-Signal Circuits Using Labeled Hybrid Petri Nets.* PhD thesis, University of Utah, Dec. 2008.
13. S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda. Verification of analog/mixed-signal circuits using labeled hybrid petri nets. In *Proc. Int. Conf. on CAD*, pages 275–282. IEEE Computer Society Press, 2006.
14. S. Little, A. Sen, and C. Myers. Application of automated model generation techniques to analog/mixed-signal circuits. In *International Workshop on Microprocessor Test and Verification*, Dec. 2007.
15. S. Little, D. Walter, and C. Myers. Analog/mixed-signal circuit verification using models generated from simulation traces. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *Automated Technology for Verification and Analysis*, volume 4762 of *LNCS*, pages 114–128. Springer, 2007.
16. C. J. Myers, R. R. Harrison, D. Walter, N. Seegmiller, and S. Little. The case for analog circuit verification. *Elec. Notes Theor. Comp. Sci.*, 153(3):53–63, 2006.
17. D. Nickovic and O. Maler. AMT: A property-based monitoring tool for analog systems. In *Formal Modelling and Analysis of Timed Systems*, 2007.
18. D. Walter, S. Little, and C. Myers. Bounded model checking of analog and mixed-signal circuits using an SMT solver. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *Automated Technology for Verification and Analysis*, volume 4762 of *LNCS*, pages 66–81. Springer, 2007.
19. 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*, pages 316–323, 2007.
20. D. C. Walter. *Verification of analog and mixed-signal circuits using symbolic methods.* PhD thesis, University of Utah, May 2007.