LEMA: A Tool for the Formal Verification of Digitally-Intensive Analog/Mixed-Signal Circuits

Andrew N. Fisher*, Satish Batchu†, Kevin Jones‡, Dhanashree Kulkarni§,
Scott Little§, David Walter¶, Chris J. Myers*
*University of Utah, Salt Lake City, UT 84112, USA †Qualcomm, Raleigh, NC, USA
‡Lockheed Martin Corporation, Aberdeen, MD, 21005 USA
§Intel Corporation, Hillsboro, OR 97124, USA ¶Virginia State University, Petersburg, VA 23806, USA

***Abstract*—The increasing integration of analog/mixed-signal (AMS) circuits into system designs has further complicated an already difficult verification problem. Recently, formal verification, which has been successful in the purely digital domain, has made some in-roads in the AMS domain. This paper describes one such formal verification tool for AMS circuits, `LEMA`. In particular, `LEMA` is capable of generating a formal model from simulation traces that, when coupled with a formal property provided in our new property language, can be model checked with one of three model checkers within `LEMA`. This paper briefly describes the capabilities of the `LEMA` AMS verification tool flow.**

I. Introduction

The increasing demand for smaller, more efficient circuits has created a need for both digital and analog designs to scale down. Digital technologies have been successful in meeting this challenge, but analog circuits have lagged behind due to limited automation support. Analog design must rely on specialists leading to the 20 percent of a circuit that is analog requiring 40 percent of the design effort. To address this problem, portions of traditionally analog designs are now constructed using digital components. One example is the *phase lock loop* (PLL) which was once completely analog is now mostly digital as shown in Fig. 1.

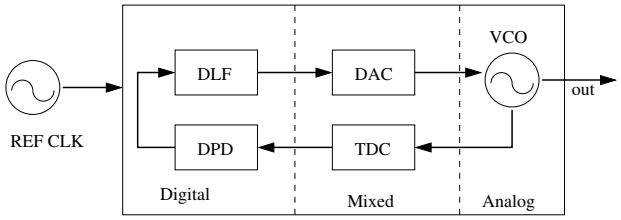

Fig. 1. Digitally-intensive AMS design of a PLL.

While the addition of digital circuitry has reduced some of the design burden, it has made the verification problem more challenging. The typical method of verifying AMS designs is to simply use detailed *transistor-level* (SPICE) simulations, the same method used in analog designs. Since analog designs typically do not have large transistor counts, performing the detailed calculations required by SPICE simulations has been viable. Adding digital components, however, greatly increases the number of transistors leading to much longer simulation times. For example, it can take weeks or even months to complete simulations for a PLL. Such long simulation times make system-level simulations difficult, if not impossible.

To improve verification efficiency, one can attempt to extend the digital methods to AMS designs. One such method is *model checking*, which checks a property over all reachable states of a circuit. Using non-determinism, model checking can make less assumptions about the environment and design parameters, making it a promising mechanism for verifying a design taking into account noise, process variations, and uncertain initial conditions. The challenge then becomes incorporating the analog behaviors into the digital formalisms. That is, to create a formal model that takes into account both the discrete nature of the digital portions of the design, as well as, the continuous nature of the analog portions. Once a formal model is created, then one can turn to the problem of creating properties to capture the desired behavior. Finally, one can use formal methods to verify that the model satisfies the given properties. A survey of recent techniques and tools for AMS verification can be found in [1].

This paper focuses on the *LPN Embedded Mixed-Signal Analyzer* (`LEMA`) which is a tool that seeks to enable the formal verification of AMS circuits. `LEMA`'s tool flow is shown in Fig. 2. `LEMA` takes the transistor-level SPICE simulation traces from a traditional analog circuit verification approach and a set of discrete thresholds, and it applies a model generator to produce a formal model that we developed called a *labeled Petri net* (LPN) [2]–[5]. The properties that `LEMA` can verify can be provided using the *Language for Analog/Mixed-Signal Properties* (LAMP), which is a simple, intuitive language for expressing AMS circuit properties [5]–[7]. `LEMA` includes a property compiler that can convert a LAMP property into an LPN. The model and property LPNs can be combined in order to check that the model satisfies the property. This checking can be done either through simulation or model checking. For simulation, `LEMA` includes a translator that can convert LPNs into a SystemVerilog model that can then be simulated using a standard SystemVerilog simulator [3]. Formal verification can also be performed by `LEMA` using one of three model checkers: an exact *binary decision diagram* (BDD) model checker [8], a *satisfiability modulo theory* (SMT) bounded model checker [8], or a conservative model checker that uses *zones* [9]. All three model checkers provide a pass/fail result, and, in the case of failure, they can provide an error trace.

This paper is organized as follows. Section II describes the model generator. Section III introduces the property compiler for LAMP. Section IV presents the translator to SystemVerilog. Section V describes `LEMA`'s three model checkers. Finally, Section VI discusses future directions.

978-1-4799-4132-2/14/$31.00 ©2014 IEEE

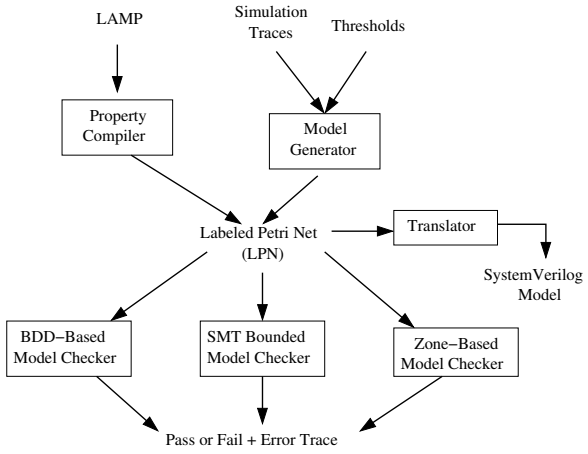

Fig. 2. LEMA's tool flow.

II. Model Generator

As indicated in the introduction, in order to use formal verification on AMS designs, one needs a method for constructing formal models that takes into account the continuous nature of analog designs. For example, before the PLL in Fig. 1 can be verified, one needs to be able to construct a formal model of the *voltage controlled oscillator* (VCO), a purely analog circuit that outputs a clock whose frequency depends on an input voltage. LEMA's chosen formalism for modeling such circuits is LPNs. An example LPN generated by LEMA is shown in Fig. 3.

An LPN consists of places and transitions along with Boolean and continuous variables. The places keep track of the current state and the transitions move the LPN from one state to another. Transitions have an enabling condition, a delay, and a set of assignments. In order for a transition to fire, the enabling condition must be true for the minimum amount of time provided by the delay and must fire before the maximum amount of delay. When the transition fires, the associated assignment statements are executed updating the corresponding variables.

Creating LPNs by hand is a tedious process and is not easy to convince AMS designers to do. Consequently, LEMA provides a model generator that takes as input the more familiar transistor-level SPICE simulations together with some threshold values and automatically constructs an LPN model [2]–[5]. The thresholds divide the space of continuous variables into regions. These regions become the places and the boundaries between the regions are indicated with enabling conditions on transitions. Furthermore, the model generator can identify some transitions as being discrete which have transitions in value after a delay.

The VCO model shown in Fig. 3 is generated using a set of three traces providing the frequency for three separate voltage values. LEMA creates a discrete variable out representing the output clock and adds delay functions $f3(ctl)$ and $f4(ctl)$ which vary based on the input control voltage. These functions produce a linear interpolation between the points of observation in the provided simulation traces. Thus, the two states, $p4$ and $p5$, create the oscillations of the output clock. When the control voltage changes, the circuit cannot instantly respond with the appropriate oscillation, as it takes some amount of time for the output to settle into the right value. During this time, the circuit is unstable and has a varying frequency of oscillation until it settles into the right value. This unstable behavior is represented in the model by the places $p2$ and $p3$ together with delay functions $f1(ctl)$ and $f2(ctl)$. When the control voltage changes, the model changes the $stable$ signal to **false** (indicating the unstable phase) and one of the transitions $pt4$ or $pt5$ fires moving the model into the left diamond. After some time, the $stable$ signal is changed to **true** (indicating the stable phase) and one of the transitions $pt6$ or $pt7$ fires moving the model into the right diamond. In order to construct this unstable period, LEMA also includes an algorithm for recognizing the unstable part of the oscillation provided in the simulation trace. This procedure is described in more detail in [3], [5].

III. Property Compiler

After a formal model has been created, the next step is to create a property to describe the desired behavior. For this purpose, LEMA provides the input language LAMP which includes the following statements [5]–[7]:

- **delay**(d) - wait for *d* time units.
- **wait**(b) - wait until expression *b* becomes true.
- **waitPosedge**(b) - wait for a positive edge on b.
- **wait**(b, d) - wait at most *d* time units for *b* to become true.
- **assert**(b, d) - ensures that *b* remains true for *d* time units.
- **assertUntil**$(b1, b2)$ - ensures that $b1$ remains true until $b2$.
- ***if-else*** statement for selections.
- **always**$(conditionsList)\{statements\}$ - continue to execute *statements* until one of the signals in the list of variables *conditionsList* changes, then break out.

Listing 1 is a LAMP property to verify that the VCO has the correct frequency response for each input voltage. This property waits a delay of 1000 time units for the signal to stabilize. Next, it waits for the positive edge of the output clock out. The frequency is then checked by the internal always block. This block asserts that the clock remains high for the appropriate amount of time, then it waits for the clock to go low in 3 time units, checks that the clock remains low for the appropriate amount of time, and finally waits for the clock to go high again within 5 time units. The property continues to check the frequency until the control voltage ctl changes. Upon a change in the control voltage, the property breaks out of the inner always and starts again at the delay. In order to check that a model satisfies a property, the property is translated into a corresponding LPN. For the LPN generation process, each statement in LAMP has a corresponding template LPN. A portion of the LPN that the property in Listing 1 compiles into is shown in Fig. 4. Note that transition prop_tFail0 is a *failure transition*. After composing the property LPN with the model LPN, a failure is indicated if this transition can fire.

IV. Translator

In order to check a property using a system-level simulation, LEMA can encode an LPN in SystemVerilog. In SystemVerilog, places become logic variables and transitions become wires. A low signal in a logic variable implies that the

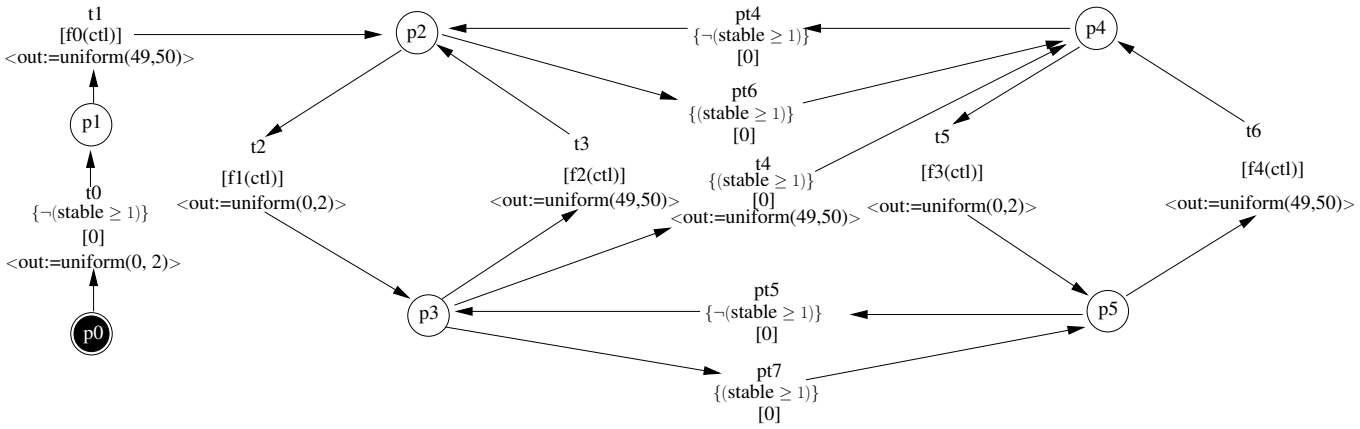

Fig. 3. LPN model for a VCO.

```
property VCO {
  real ctl;
  real out;
  always {
    delay (1000);
    waitPosedge (out>=40);
    always (ctl) {
      assert (out >= 40, f3(ctl));
      wait (out <= 30, 3);
      assert (out <= 30, f4(ctl));
      wait (out >= 40, 5);
    }
  }
}
```

Listing 1. A LAMP property for a VCO.

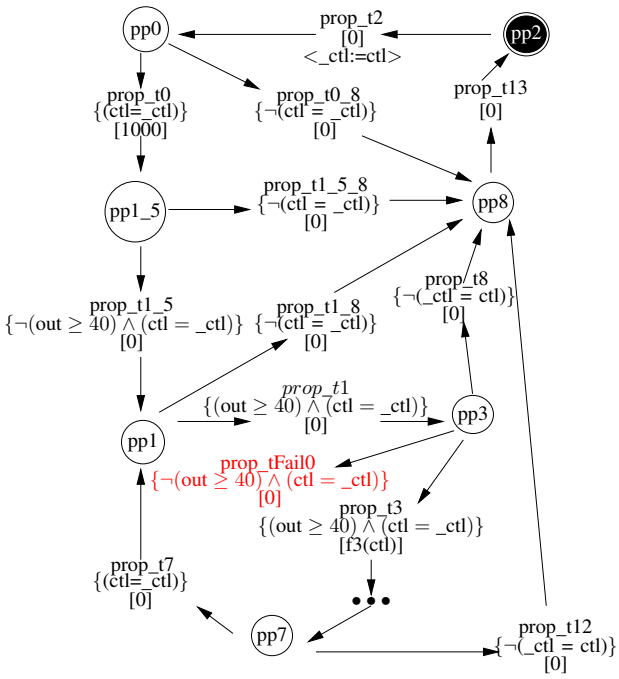

Fig. 4. A portion of the property LPN for Listing 1.

place is not marked and a high signal indicates it is marked. Initially, all places are set low, then after a delay, the initial places are marked to start the simulation. A transition fires by sending a pulse on the wire, that is, the transition wire is set high and then set back low. This process is handled by an assign statement whose delay is set by a custom function and an assignment composed of a conjunction of the marking needed for this transition and the transition's enabling condition. The custom *delay* function handles the setting of the wire high after suitable delay and resetting the wire low immediately after the transition occurs. Finally, an always statement is added that is triggered by the positive edge of the transition wire. The body of the always statement handles updating the state by setting the incoming places low, the outgoing places high, and making any necessary signal assignments. A portion of the VCO model could be translated as shown in Listing 2.

V. Model Checkers

Verification can also be performed using a model checker that determines all possible reachable states and whether or not a failure transition ever occurs. LEMA has three different model checkers. It has a BDD-based model checker that is exact, but it trades performance for memory efficiency [8]. It also has an SMT-based bounded model checker that scales better, but it can only prove there are no failure transitions in a specified number of iterations [8]. Finally, it has a conservative zone-based model checker that lies somewhere between the SMT and BDD model checkers [9]. While it is not exact, it has better performance than the BDD model checker, and it can prove that failure transitions never fire which the SMT model checker cannot. Due to space limitations, this paper focuses on the zone-based model checker.

Zone's are a subset of Euclidean space formed by intersecting half-planes associated with equations of the form $v \leq a$, $v \geq a$, or $v_i - v_j \leq a$ where v, v_i, and v_j are variables and a is a constant. In two dimensions, zones are polygons formed by using only lines forming a 90° or 45° angle with the coordinate axes. LEMA's zone-based model checker is a form of *reachability analysis*. In reachability analysis, one

```
'timescale 1ps/1fs
module VCO(input real ctl,output real out);
  wire t0,t1,t2,t3,t4,t5,t6;
  wire pt4,pt5,pt6,pt7;
  logic p0,p1,p2,p3,p4,p5;
  initial begin
    p0=0; p1=0; p2=0; p3=0; p4=0; p5=0;
    #1 p0 = 1; // initial marking
  end
  assign #(delay(~t0,0)) t0=(p0 &&
    ~(stable>=1));
            ...
  always@(posedge t0) begin
    p0 = 0; p1 = 1; out = uniform(0,2);
  end
endmodule
```

Listing 2. Portion of the SystemVerilog for the VCO model.

TABLE I. VERIFICATION RESULTS FOR A VCO CIRCUIT.

Property	Control Signals	Time (s)	States	Verifies?
Phase Checker	2	0.158	18	yes
Phase Checker	3	0.161	18	yes
Phase Checker	4	0.161	24	yes
Phase Checker	2,3,4 reg. int.	0.195	24	yes
Phase Checker	2,3,4 random	1.411	336	yes

finds all possible states that are reachable from the initial states. Of course, with continuous variables, the state space is infinite and so must be divided into equivalence classes of states. Zones are used to collect the continuous portion of a set of states together into a finite representation. The basic algorithm is a depth-first search. The algorithm starts with the initial state set and finds all possible events. An event is chosen and fired. Then the resultant state set is found, time is allowed to move forward as far as possible without causing another event, and then all possible events are found again. This process continues until one reaches a state set found before or no events are possible. At this point, the algorithm backs up to the previous state set and another event is chosen. The algorithm ends when all events have been explored. Initially, zone-based methods were used to verify *timed automata*; however, LEMA uses warping [9] to allow zones to be applied to non-rate one continuous variables in addition to clock variables. With warping, variables are scaled by their rate, turning them into rate-one variables and then the resulting figure is over-approximated by a zone.

Table I shows the results of applying the zone-based model checker to the combination of the generated VCO model LPN from Section II and the property LPN corresponding to Listing 1 as described in Section III. The first three lines show the result when the model is put into an environment that outputs a single control voltage. The fourth line allows the control voltage to change to one of three values, then the control voltage remains at that level for a fixed amount of time before allowing another level to be chosen. Finally, the fifth line has an environment that is allowed to randomly choose a time to switch to one of three control voltage levels.

VI. DISCUSSION

LEMA has made some strides into the verification of AMS circuits by providing a complete tool flow that is aimed at being easier for the non-formal methods user. The complete process of creating a model, inputting a property, and running verification can be done without knowing anything about LPNs and model checkers. Although progress has been made, LEMA still has some challenges to face. Currently, work is being done to more fully extend LEMA to general LPNs by extending the zone-based checker to handle continuous variables with ranges of rates. In addition, another model checker is being added that uses the more general polyhedral class of *octagons* [10] to reduce the number of possible false negatives.

VII. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CCF-1117515. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. In the past, this work has also been supported by an SRC Graduate Fellowship, SRC contracts 2002-TJ-1024, 2005-TJ-1357, 2008-TJ-1851, and a grant from Intel Corporation.

REFERENCES

[1] M. H. Zaki, S. Tahar, and G. Bois, "Formal verification of analog and mixed signal designs: A survey," *Microelectronics Journal*, vol. 39, no. 12, pp. 1395 – 1404, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269208002085

[2] S. Little, D. Walter, K. Jones, C. Myers, and A. Sen, "Analog/mixed-signal circuit verification using models generated from simulation traces," *The Inernational Journal of Foundations of Computer Science*, vol. 21, no. 2, pp. 191–210, 2010.

[3] S. Batchu, "Automatic extraction of behavioral models from simulations of analog/mixed-signal (AMS) circuits," Master's thesis, University of Utah, Salt Lake City, UT, USA, 2010.

[4] D. Kulkarni, S. Batchu, and C. J. Myers, "Improved model generation of AMS circuits for formal verification," *2011 Virtual Worldwide Forum for PhD Researchers in Electronic Design Automation*, Nov 2011.

[5] D. Kulkarni, "Formal verification of digitally-intensive analog/mixed signal circuits," Master's thesis, University of Utah, Salt Lake City, UT, USA, 2013.

[6] D. Kulkarni, A. N. Fisher, and C. J. Myers, "A new assertion property language for analog/mixed-signal circuits," in *Specification Design Languages (FDL), 2013 Forum on*, Sept 2013, pp. 1–8.

[7] A. N. Fisher, D. Kulkarni, and C. J. Myers, "A new assertion property language for analog/mixed-signal circuits," in *Languages, Design Methods, and Tools for electronic System Design: Selected Contributions from FDL 2013*, ser. Springer-Verlag, M.-M. Louërat and T. Maehne, Eds., vol. 311, Oct. 2014.

[8] D. Walter, S. Little, C. Myers, N. Seegmiller, and T. Yoneda, "Verification of analog/mixed-signal circuits using symbolic methods," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 12, pp. 2223 –2235, dec. 2008.

[9] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda, "Verification of analog/mixed-signal circuits using labeled hybrid petri nets," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, pp. 617–630, 2011.

[10] A. Miné, "The octagon abstract domain," *Higher-Order and Symbolic Computation*, vol. 19, no. 1, pp. 31–100, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10990-006-8609-1