A Practical Approach for Monitoring Analog Circuits

Mohamed H. Zaki
Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., H3G 1M8
Montreal, QC, Canada
mzaki@ece.concordia.ca

Sofiène Tahar
Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., H3G 1M8
Montreal, QC, Canada
tahar@ece.concordia.ca

Guy Bois
Genie Informatique, Ecole Polytechnique de Montreal
Pavillon Decelles, 5255 Avenue Decelles, H3T 2B1
Montreal, QC, Canada
guy.bois@polymtl.ca

ABSTRACT

Formal methods have been advocated for the verification of digital design where correctness is proved mathematically. In contrast to digital designs, the verification of analog and mixed signal systems is a challenging task that requires lots of expertise and deep understanding of their behavior. In this paper, we present a run-time verification methodology based on monitoring the behavior (solution flow) of analog circuits. Monitors are deterministic timed automata that can be synthesized from temporal properties. For illustration purposes, we applied our methodology on the verification of the oscillation property of a tunnel diode oscillator.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Verification*

General Terms

Verification

1. INTRODUCTION

The functionality of analog and mixed signal (AMS) circuits is defined directly in terms of continuous electrical quantities and is usually sensitive to environment factors like signal noise, current leakage, in addition to higher order physical effects when designing in deep submicron. Traditionally, simulation was used, where the evaluation of the results is often done manually or in an ad-hoc and informal fashion. The search of the state space is not complete in non-exhaustive methods. Simulation based techniques were then complemented by symbolic techniques, where the effect of parameters variations on the system behavior is analyzed. Although successful, challenging problems (like non-linear effects) make these techniques only suitable for simple designs. Researchers started lately studying the applicability of formal methods like model checking for the verification of AMS systems. However, a direct application of model checking on continuous systems is very difficult and abstraction techniques are usually required in order to achieve the verification task. Moreover, model checking (in particular in a dense time domain) is computationally more expensive and therefore suffers even more from the state-space explosion problem that makes exhaustive verification very hard, and limitations in memory and/or time resources make verification practical for only small systems.

In order to cope with these problems, new methods were developed. Among the most important ones are program monitoring (run-time verification) which were initially developed to tackle the efficiency problems of verification techniques mainly for large software with infinite state space, where formal verification as well as exhaustive simulation methods fail to achieve confidence requirements in the development process. Run-time verification is a semi-formal method that combines desirable properties from simulation and formal verification while avoiding the undesirable ones. No computational model needs to be generated prior to the verification, avoiding state space explosion. By employing logical monitors, an efficient analysis of the results is achieved, avoiding exhaustive inspection, by testing whether a given behavior satisfies a property. This process can be performed in two different fashions [15]: *Offline monitoring* starts after the whole sequence is given. *Online monitoring* is interleaved with the process of reading the sequence and is similar to the way the sequence is read by an automaton. Online monitors can detect violation or satisfaction as soon as they happen.

In this paper, we propose an online monitoring methodology for continuous systems. The methodology is based on monitoring solution flows (a set of solutions) of a system of ordinary differential equations. The rest of the paper is organized as follows: In Section 2, we present related work. In Section 3, we give a brief overview of the proposed methodology, followed by system description in Section 4. The temporal specification is then described in Section 5. The monitors description and construction are explained in Section 6. Experimental results are shown in Section 7, and finally, we conclude the paper with Section 8.

2. RELATED WORK

Moniroting based verification: Program monitoring has been applied successfully for hardware and software analysis. Motivated with the success of PSL assertion language

[2] for hardware verification, Maler *et. al* [16] proposed a simple methodology for offline monitoring the simulation of continuous signals by extending the PSL logic to support predicates over the reals (signals). The goal is to verify continuous and analog signals. Tan *et. al* [18] developed tools for monitoring real-time and hybrid systems, where timed and linear hybrid automata can be used to monitor real-time and hybrid behavior, respectively. Monitors are not built from formal logic properties, in addition only single trajectories are monitored. Recently similar ideas using hybrid automata as monitors have been integrated with the PHAVer hybrid system analysis tool [4]. This tool provides sound verification results based on linear hybrid automata approximations and was used to verify properties of piecewise models of oscillators like amplitude bounds and phase jitter. In [4], the whole state space is considered as partitioning techniques are used to construct the flow, using validated (Interval Arithmetics) method, which will generate only the reachable states reducing significantly the cost of the whole discretization. Moreover, by using temporal logic, more complex properties can be verified which in contrast to the case by case (template based) monitor building, which becomes inefficient for complex designs.

AMS formal verification: Several methods for approximating reachable sets for continuous dynamics have been proposed for the verification of AMS systems. These methods rely on the discretization of the continuous state space. In [13], the authors tried to construct a finite-state discrete abstraction of an electronic circuits by partitioning the continuous state space representing the characteristics of transistors into hypercubes using a fixed grid and computed the reachability relations between these cubes by optimizing techniques. In [7], the authors tried to overcome the expensive computational method in [13], by using discretization and projection techniques of the state space, reducing the dimension of the state space. While the approach is less precise due to projection, it is still sound. Variant approaches of polyhedral based analysis were adopted in [3, 8]. For instance, the model checking tools *d/dt* [3] and Checkmate [8] were used in the verification of a biquad low-pass filter [3], and tunnel diode oscillator and $\Delta - \Sigma$ modulator [8], respectively. Other computational techniques for AMS verification can be found in [9].

Interval Methods in AMS circuit design: Interval and affine arithmetics have been used in the analysis of analog circuits like in [10], where the authors presented an approach for equivalence checking of linear analog circuits with parameter tolerances. The goal is to prove that an actual circuit fulfills a specification in a given frequency interval for all parameter variations. In [11], affine arithmetics was used for helping in the circuit sizing.

3. VERIFICATION METHODOLOGY

Figure 1 depicts the online monitoring methodology we propose for continuous systems. Given the system description and its specification described by non-linear differential equations and timed computational temporal logic (T-CTL) formulas [1], respectively, we build a monitor which can detect bad behavior within a specified time period. During the simulation, we essentially build a deterministic automaton that incrementally monitors the prefix of state sequences that has been simulated so far. If the prefix is such that the execution can no longer lead to satisfaction of the property (a bad prefix), then the simulation can be stopped demonstrating that the property does not hold for the system.

Usually a simulation run explores a single execution of the model. This makes it suitable for the verification of linear temporal logic properties (expressing aspects of individual executions) only. To be valid, a property should hold for all executions of the system. Since this number can be large (it is often infinite), it leads to extremely low coverage. A formal candidate approach to obtain a flow (set of solutions) is using *validated methods* (proposed by Moore in [17]), where interval arithmetic analysis is used to guarantee the inclusion of required solutions for a given set of initial conditions. In this work, we chose the validated methods based AWA tool developed by Lohner [12].

Several criteria affect the choice of the temporal specification language. In thecontext of abstraction, where over-approximations of the reachable states are explored (due to interval based analysis in this paper), a variant of $\forall$CTL logic can be chosen to ensure the soundness of the verification results. In addition, as the execution of the system is time bounded, the transitions can be incomplete. However, by restricting the approach to bounded time, the soundness of the approach can be achieved. Another criteria is the possibility to synthesize a monitor from the given properties. Based on the deterministic subset identified by Maidle [14], we use the common fragment of timed logic languages T-CTL [1] and MITL [5], for which deterministic automata can be generated.

4. SYSTEM DESCRIPTION

We consider continuous systems described by non-linear ordinary differential equations (ODEs). A *continuous system* is a triple $CS = (\mathcal{X}, \mathcal{X}_0, \mathcal{P})$ with $\mathcal{X} \subseteq \mathbb{R}^d$ is a continuous state space, $\mathcal{X}_0 \subseteq \mathcal{X}$ is the set of initial states and $\mathcal{P} : \mathcal{X} \to \mathbb{R}^d$ is the continuous vector field. The behavior of a continuous system is governed by the differential equations of the form $\dot{x_k} = \frac{dx_k}{dt} = \mathcal{P}_k(x_1, \ldots, x_d)$, $k = 1, \ldots d$, d is the system dimension, t is the independent real time, $\mathcal{P}_k$ is a vector field. We assume that the differential equation has a unique solution for each initial value.

The semantics of a continuous system $CS = (\mathcal{X}, \mathcal{X}_0, \mathcal{P})$ over a continuous time period (an interval) $T_c = [\tau_0, \tau_1] \subseteq \mathbb{R}^+$ can be described as trajectory $\Phi_x : T_c \to \mathcal{X}$ for $x \in \mathcal{X}_0$ such that $\Phi_x(t)$ is the solution of $\dot{x_k} = \mathcal{P}_k(x_1, \ldots, x_d)$ with initial condition $\Phi_x(0) = x$ and $t \in T_C$ is a time point. Another representation of the behavior by means of a transition system $\mathcal{A} = (Q, Q_0, \sigma, L)$, where $q \in Q$ is a configuration (x, Γ). $x \in \mathcal{X}$ and Γ is a set of intervals, where $\cup_{i \geq 0} t_i \subseteq \mathbb{R}^+$, $t_i \in \Gamma$. We have $t_1, t_2 \in \Gamma$ for $\Phi_{x'}(t_1) = \Phi_{x''}(t_2) = x$ and $x', x'' \in \mathcal{X}_0$. $q \in Q_0$, when $t_0 \in \Gamma$ and $\Phi_x(t_0) = x$ and t_0 is the singular interval, L is an interpretation function such that $L : Q \to \mathbb{R}^n \times 2^{\mathbb{R}^+}$. Finally, $\sigma \subseteq Q \times Q$ is a transition relation such that $(q_n, q_m) \in \sigma$ iff $\exists t_n \in \Gamma_n$, $\exists t_m \in \Gamma_m$. $t_n < t_m$ and $\lim_{t_n \to t_m} \Phi_x^{q_n}(t_n) = \Phi_x^{q_m}(t_m)$, $x \in \mathcal{X}_0$. A run (trace) of the transition system can be a set of state tuples $\mu = < x_0, t_0 >< x_1, t_1 >, \ldots$ where x_j is a valuation (state) and $t_i \in \Gamma_j$ such that a transition exists between two consecutive state tuples $< x_j, t_j >< x_{j+1}, t_{j+1} >$. This corresponds to the timed word (see Section 6 for more details).

Computing the Solution Flow

Usually the analytic solution for the differential equation

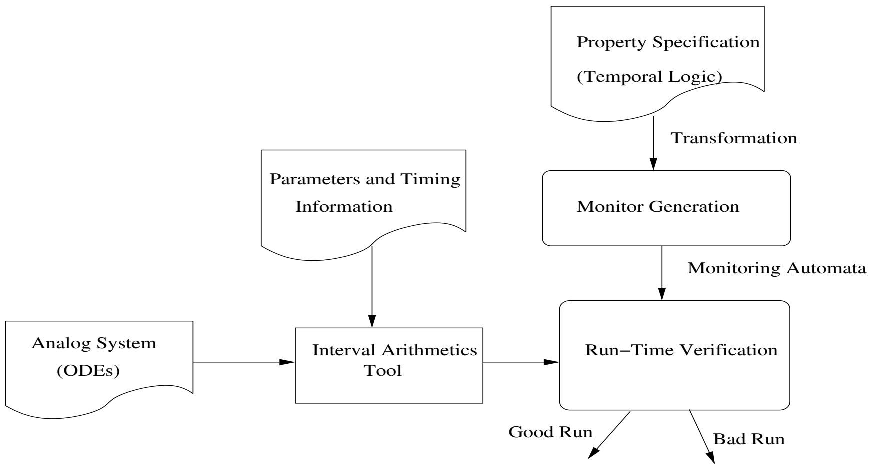


Figure 1: Verification Methodology for Continuous Systems

is not possible and approximate methods are used instead. The notion of sufficient complete discretization must be defined, which can be understood as sampling criteria that captures all different states in the given continuous evolution. In order to achieve sound discretization, we use interval arithmetics methods which are approximated methods that enclose the original solution, starting from a set (closed interval) of initial continuous states; i.e., the domain of execution is the interval domain $\mathbb{I}$, which is a conservative domain over-approximating the original one $\mathbb{R}$.

Interval arithmetic [17] is an extension of real arithmetic for real intervals which has its basic operators defined together with their evaluation rules and algebraic properties. Interval functions are introduced as the interval counterparts of real functions, which can be represented by means of interval expressions. Basic interval arithmetic operators can be defined as follows: Let I_1 and I_2 be two real intervals (bounded and closed). The basic arithmetic operations on intervals are defined by: $I_1 \Phi I_2 = \{r_1 \Phi r_2 | r_1 \in I_1 \wedge r_2 \in I_2\}$ with $\Phi \in \{+, -, , /\}$ except that I_1/I_2 is not defined if $0 \in I_2$, with:

$$\begin{array}{lcl}
[a,b]^{\iota} & \triangleq & [a+b] \\
[a,b] +^{\iota} [a',b'] & \triangleq & [a+a', b+b'] \\
[a,b] -^{\iota} [a',b'] & \triangleq & [a-b', b-a'] \\
[a,b] \times^{\iota} [a',b'] & \triangleq & [min(aa',ab',ba',bb'), \\
 & & max(aa',ab',ba',bb')] \\
1 \div^{\iota} [a,b] & \triangleq & [1 \div b, 1 \div a] if\ 0 \notin [a,b] \\
[a,b] \div^{\iota} [a',b'] & \triangleq & [a,b] \times 1 \div [a',b']
\end{array}$$

In addition, other elementary functions (i.e., *exp, ln, power, sin, cos*) can be included as basic interval arithmetic operators. For example, (*exp*) may be defined as $exp([a,b]) = [exp(a), exp(b)]$. An interval function F maps an n-ary tuple of real intervals (R-box) into the smallest real interval containing all the real values that would be obtained for each real valued combination within the interval domains of the R-box. More formally, let F be the n-ary interval function represented and B an n-ary R-box, the interval, denoted by F(B), obtained by applying the interval function F to B, is the smallest real interval containing the range $f^*(B)$ of the real function f over B; $f^*(B) = \{f(< r_1, \ldots, r_n >) | < r_1, \ldots, r_n > \in B\} \subseteq F(B) \wedge \forall [a,b] f^*(B) \subseteq [a,b] \rightarrow F(B) \subseteq [a,b]$. This interval analysis property shows how interval evaluation always gives a sound over-approximation of the real analysis.

To enhance simulation based methods, interval approaches for solving initial value problems were used by extending well known approaches like Taylor series methods and Runge-Kutta methods. The advantage of interval arithmetic is to calculate each approximation step explicitly, keeping the error term within appropriate interval bounds. Discretization and computational errors are thus encapsulated within bounds of uncertainty around the true solution function providing guaranteed enclosures of the actual solution function [17].

5. TEMPORAL SPECIFICATION: CT-CTL

We introduce a variant of temporal logic tailored for specifying desired properties of continuous systems. The logic is based on a bounded subset of the realtime logic T-CTL, augmented with a mapping from continuous domains into propositions. T-CTL is a computational tree logic (CTL) for the reasoning about real-time systems, where temporal modalities are restricted to intervals of the form $I = [a,b]$ with $0 < a < b$ and $a, b \in \mathbb{Q}_{\geq 0}$. The use of bounded temporal properties is to restrict the verification for a specific time avoiding the non-termination. As we are interested in system with real-valued state variables, we extended the T-TCTL language with predicates over real constants and real variables. We call the new language CT-CTL (Continuous Time CTL).

Syntax of CT-CTL. The basic formulae of the CT-CTL are defined by the following grammar (note: we restrict ourselves to the universal quantifier $\forall$): $\varphi := \lambda(y_1, \ldots, y_n) | \neg\varphi | \varphi_1 \vee \varphi_2 | \ \forall(\varphi_1 \mathbf{U}_I \varphi_2) | true$, where λ belongs to a set of predicates Pre and y_i is a term (that is a constant or a variable). From the basic CT-CTL operators, one can derive other standard Boolean and temporal operators, in particular the time-constrained eventually and always operators. $\forall\Diamond_I \varphi \triangleq \forall true \mathbf{U}_I \varphi$ and $\forall\Box_I \varphi \triangleq \neg\exists\Diamond_I \neg\varphi$

Semantics of CT-CTL. Given a continuous system $CS = (\mathbb{X}, \mathbb{X}_0, f)$, we define its Kripke structure which is the transition system $\mathcal{A} = (Q, Q_0, \sigma, L)$, extended with an interpretation function $[\![.]\!]$, written as $K = (\mathcal{A}, [\![.]\!])$. The semantics of the language is provided by the interpretation $[\![.]\!]$ as follows:

- For a constant C, $[\![C]\!]$ is an element of $\mathbb{R}$
- For a state variable $y \in Y$, (where Y is the set of state variables) which can be considered as a function y, $[\![y]\!]$ is a function $\mathbb{R}^+ \to \mathbb{R}$
- For n-ary predicate λ, $n \geq 1$, the meaning $[\![\lambda]\!]$ is a function $\mathbb{R}^n \to \mathbb{B}$.
  The interpretation $[\![.]\!]$ extends to arbitrary terms, inductively: $[\![\lambda(y_1, \ldots y_n)]\!] = [\![\lambda]\!]([\![y_1]\!], \ldots [\![y_n]\!])$. In addition, we have the concretisation function $\Upsilon_\lambda : \mathbb{B} \to 2^{\mathbb{R}^n}$ such that $\lambda \in$ *Pre* and $\Upsilon([\![\lambda(y)]\!]) = \Upsilon_\lambda(b) = \{y \in \mathbb{R}^n | \lambda(y) = b\}$. Intuitively, Υ_λ is a set of states, where λ holds with the condition $\Upsilon_\lambda \cap \Upsilon_{\neg\lambda} = \emptyset$

A mapping function case can be described as follows: $\nu : \mathbb{R}^n \to \mathbb{B}$, which is a function associating to each predicate $\lambda(y_1, \ldots y_n)$ an atomic proposition P. Practically, this mapping function can be used for a translation from CT-CTL to T-CTL. The satisfaction relation $s = (x, \Gamma) \models \varphi$, indicating that a state satisfies a property φ starting from position $\underline{\tau}$, where $\tau = [\underline{\tau}, \overline{\tau}]$ and $\tau \in \Gamma$ is defined inductively as follows:

- $q \models true$
- $q \models \lambda(y_1, \ldots y_n)$ iff $L_X(q) \in \Upsilon([\![\lambda(y_1, \ldots y_n)]\!])$
- $q \models \neg\varphi$ iff $q \not\models \varphi$
- $q \models \varphi_1 \vee \varphi_2$ iff $q \models \varphi_1$ or $q \models \varphi_2$
- $q \models \forall(\varphi_1 \mathbf{U}_I \varphi_2)$ iff for all paths of the transition system; $\pi = L_X(q_0), \ldots$, where $q_0 = q$, starting from position $\underline{t}$, where $t = [\underline{t}, \overline{t}]$ and $t \in \Gamma$, $\exists t' \in [\underline{t} + a, \underline{t} + b]$. $q \models \varphi_2$ and $\forall t'' \in [\underline{t}, t']$. $q \models \varphi_1$
- $q \models \forall\Diamond_I \varphi$ iff for all paths of the transition system; $\pi = L_X(q_0), \ldots$, where $q_0 = q$, starting from position $\underline{t}$, where $t = [\underline{t}, \overline{t}]$ and $t \in \Gamma$, $\exists t' \in [\underline{t} + a, \underline{t} + b]$. $q \models \varphi$
- $q \models \forall\Box_I \varphi$ iff for all paths of the transition system; $\pi = L_X(q_0), \ldots$, where $q_0 = q$, starting from position $\underline{t}$, where $t = [\underline{t}, \overline{t}]$ and $t \in \Gamma$, $\forall t' \in [\underline{t} + a, \underline{t} + b]$. $q \models \varphi$

6. MONITORING AUTOMATA

In this section, we briefly give the description of the monitors and how they can be used to detect properties of the system. We also set the main requirement on the T-CTL logic in order to construct deterministic timed automata.

A *monitoring timed automata* $\mathcal{M}$ is a tuple $\{S, S_0, V, Q, \Theta, G, T, SC, \mathcal{C}\}$ where S is a finite set of locations, $S_0 \subseteq S$ is the finite set of initial locations, V is a set of state variables, $Q : S \to 2^\Sigma$ is a labelling function assigning each location a set of atomic propositions $\{p | p \in \Sigma\}$ over state variables v_j, $v_j \in V$. Θ is the set of clocks with time valuation (timer assignment) $\delta : \Theta \to \mathbb{R}$. $G : S \to 2^{TC(\Theta)}$ assigns to each location as set of timers constraints (TC) over the location timers Θ. For example, the satisfaction of a timer constraint $x > 0$ (where $x \in \Theta$) by a timer valuation δ is defined as follows: $\delta \models x > 0$ iff $\delta(x) > 0$. $T \subseteq S \times S$ is the set of edges between locations and $SC : T \to 2^X$ associates with each edge a set of clocks that need to be set. The automata is also extended with the acceptance set $\mathcal{C}$, such that $s \in \mathcal{C} \subseteq S$ if there is no path from s to any $s' \in \mathcal{B}$, where $\mathcal{B} \subseteq S$ is the Buchi acceptance condition. A timed automata is deterministic if being at a location s_i, only one transition can be taken at a time.

A *trace* of monitoring timed automata $\mathcal{M}$ is a sequence $\pi = < s_0, \delta_0, \mathcal{I}_0 > < s_1, \delta_1, \mathcal{I}_1 > \ldots$, where δ_i are evaluations of the clocks upon entering the location s_i, $\mathcal{I}_i$ is the time interval representing the time spent at location S_i. The timer associated with each state is a decreasing timer such that we have for each pair $< s_i, \delta_i, \mathcal{I}_i > < s_{i+1}, \delta_{i+1}, \mathcal{I}_{i+1} >$, $e = (s_i, s_{i+i})$, $\delta_{i+1} = \delta_i - |\mathcal{I}_i|$ or $\delta_{i+1}(x) = t \in \mathbb{R}$ for $x \in SC(e)$ [18]. A *timed word* is a timed state sequence $\eta = \{\mathcal{V}_0, \mathcal{I}_0\}, \{\mathcal{V}_1, \mathcal{I}_1\}, \ldots$, where $\mathcal{I}_i^r = \mathcal{I}_{i+1}^l$ and $\mathcal{V}_i(t)$ denotes the evaluation of the state variables at time $t \in \mathcal{I}_i$. We say that monitoring automata $\mathcal{M}$ accepts a timed word η if $\mathcal{M}$ has an accepting run for η, or more formally, a timed word $n = \{\mathcal{V}_0, \mathcal{I}_0\}, \{\mathcal{V}_1, \mathcal{I}_1\} \ldots$ is accepted by an automata $\mathcal{M}$ if there is a trace of $\mathcal{M}$; $\pi =< s_0, \delta_0, \mathcal{I}'_0 >< s_1, \delta_1, \mathcal{I}'_1 > \ldots$, such that if $t \in \mathcal{I}_i$, then there is an l such that $t \in \mathcal{I}'_l$ and $\mathcal{V}_i(t) \models p$, $p \in Q(S_l(t))$.

We say a trace is accepting a run of the Buchi timed automata if and only if the trace is infinite π and $inf(\pi) \cap B \neq \emptyset$, where $inf(\pi)$ is the set of locations, trace π visits infinitely often. The monitoring automata accept finite words, which implies the acceptance of a finite prefix of the timed word, means that this trace will be rejected if the automata was a Buchi automata (extended with the Buchi condition) [18].

Constructing the Monitor

The idea behind the construction is to create the on-the-fly tableau automaton of the temporal logic formula ϕ, but interprets it as an automaton on finite words. If we would like to verify individual (prefixes of) timed state sequences that result for instance from a system simulation, we need the observer automaton to be deterministic. However, the determinisation of timed automata is impossible in general. Maidl [14] identified a common fragment of $\forall$CTL and LTL formulae for which deterministic automata can be constructed. We use time equivalent of the deterministic fragment identified by Maidl with restrictions of timed modalities to intervals of the form $[0.d]$ to obtain a deterministic timed automata monitor. This recent restriction was imposed by Geilen [6], to guarantee the deterministic property while preserving the dense semantics of the logic. The set of formulas of this fragment (we call it $\forall\tilde{\text{C}}$T-CTL) is as following: $\varphi := \ p |\ \neg\varphi | \varphi_1 \wedge \varphi_2 | (p \wedge \varphi_1) \vee (\neg p \wedge \varphi_2)| \forall((p \wedge \varphi_1)\mathbf{U}_I(\neg p \wedge \varphi_2))|true$. For example, $\forall(\varphi_1 \mathbf{U}_I p)$ can be expressed in $\forall\tilde{\text{C}}$T-CTL as $\forall((\varphi_1 \wedge \neg p)\mathbf{U}_I p)$. The construction of the monitor follows directly form the decision procedure developed by Geilen [6], where he used it for building deterministic timed automata from the MITL logic. Hence, we can state the following property. Let $\mathcal{A}_\phi$ be an on-the-fly tableau automata of $\forall\tilde{\text{C}}$TL formula ϕ, then $\mathcal{L}(\phi) \subseteq \mathcal{L}(\mathcal{A}_\phi)$.

7. APPLICATION: TUNNEL DIODE OSCILLATOR

To illustrate our methodology, we consider the tunnel diode oscillator (Figure 2), which has been used by many researchers (e.g.,[4, 8, 9]) as benchmark. The tunnel diodes exploit a phenomenon called resonant tunneling to provide interesting forward-bias characteristics, due to its negative resistance characteristic at very low forward bias voltages. This means that for some range of voltages, the current decreases with increasing voltage. This is in contrast with conventional diodes that have a non-linear I-V characteristic, but the slope of the curve is always positive. This characteristic makes the tunnel diode useful as oscillator. When a small forward-bias voltage is applied across a tunnel diode, it begins to conduct current. As the voltage is increased, the current increases and reaches a peak value called the peak current. If the voltage is increased a little more, the current actually begins to decrease until it reaches a low point called the valley current. If the voltage is increased further yet, the current begins to increase again, this time without decreasing into another valley.

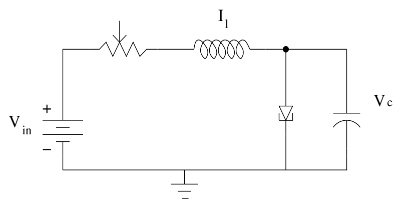


Figure 2: Tunnel Diode Oscillator

In following, we present experimental results from applying the verification methodology proposed in this paper on the tunnel-diode oscillator circuit. We focus on the current I_L and the voltage V_C across the tunnel diode in parallel with the capacitor of a serial RLC circuit (see Figure 2). The state equations of the circuits are given as $\dot{V}_C = \frac{1}{C}(-I_d(V_C) + I_L)$ and $\dot{I}_L = \frac{1}{L}(-V_C - \frac{1}{G}I_L + V_{in})$, where $I_d(V_C)$ describes the non-linear tunnel diode behavior. We analyze the circuits in two modes. The first when the circuit is in stable oscillation for a given set of parameters, the other case when this oscillation dies out. The kind of properties we are interested to verify can be for example: *The system behavior will be the same for the set of initial condition*, or, *For which set of parameters values, circuit oscillates?* We chose these two different sets of parameters values of the oscillator circuit $\{C = 1000e^{-12},\ L = 1e^{-6},\ G = 5000e^{-3},\ Vin = 0.3\}$ and $\{C = 1000e^{-12},\ L = 1e^{-6},\ G = 2000e^{-3},\ Vin = 0.3\}$ along with the set of initial values of voltages $[0.8\ V,\ 0.9\ V]$ and currents $0.04\ mA$ and the analysis region of interest $-1\ V \leq V_C \leq 1\ V$ and $0.01\ mA \leq I_L \leq 0.9\ mA$. Suppose we want to verify the following property on the set of trajectories [8]: $\forall\Box_{[0,1e^{-6}]}(\forall\Diamond_{[0,6e^{-7}]}(I_L \leq 0.02)) \ \wedge\ \forall\Box_{[0,1e^{-6}]}(\forall\Diamond_{[0,6e^{-7}]}(I_L \geq 0.06))$, which can be understood as within the time interval $[0, 1e^{-6}]$ on every computation path, whenever the I_L amplitude will reach 0.02, it will reach this value again within the time interval $[0, 0, 6e^{-7}]$, the same goes for the I_L amplitude 0.06. This property checks for oscillation behavior of the circuit.

An efficient way to build observer automata for complex properties (properties formed of the conjunction of simpler ones) is to construct an automaton for sub-properties (see Figure 3) and then use the synchronous product of these automata, with a state being marked as final (indicating a bad run).

Monitoring the interval based simulation, with the first set of parameters, we can find out that the circuit is oscillating for the given set of initial conditions (see Figure 4.a) as all the runs are accepted by the timed automata. Hence within the specified time interval, the property in verified. By following the same procedure for the system with the second set of parameters, but with the same initial conditions, we can find out that the circuit is non oscillating and the trace goes to an erroneous state (dashed state) of the left automaton in Figure 3. As the run cannot be accepted by the automata, which can be interpreted as once the trace reaches a state, it remains deadlocked there during the verification period as no state is accepting its suffix. Physically, when the circuit starts up, the energy of the system is lost due to the positive circuit resistance. Starting from any point in the analysis region, the oscillations die down to the equilibrium point (see Figure 4.b).

8. CONCLUSION

The lack of methods for computing reachable sets of continuous dynamics has been the main obstacle towards an algorithmic verification methodology for continuous systems. Unlike conventional approaches which attempt to find exact solutions and are thus limited by undecidability of most non-trivial systems, we present a practical verification approach based on an efficient method for monitoring the continuous behavior using a combination of techniques from interval based methods and automata theoretic approaches (the construction of monitor automata) to prove properties about the systems. Future work includes extending the monitoring for more complex designs, verifying mixed signal and hybrid systems, and exploring more complex case studies like phase locked loop designs.

9. REFERENCES

[1] R. Alur, C. Courcoubetis, D. L. Dill. Model-checking in dense realtime. Journal of Information and Computation, 104(1):2–34, 1993.

[2] I. Beer, *et. al.* The Temporal Logic Sugar. In Computer Aided Verification ,LNCS 2102, Springer, 2001, pp. 363-367

[3] T. Dang, A. Donze, O.Maler. Verification of Analog and Mixed-signal Circuits using hybrid system techniques. In Formal Methods in Computer-Aided Design, LNCS 3312, Springer: 14-17, 2004.

[4] G. Frehse, B. Krogh, R. Rutenbar, O. Maler Time Domain Verification of Oscillator Circuit Properties, Workshop on Formal Verification of Analog Circuits, April 9th, 2005, Edinburgh, UK.

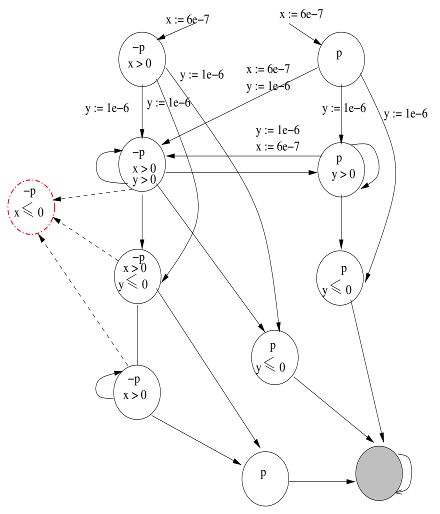
[5] M. Geilen. An Improved On-The-Fly Tableau Construction for a Real-Time Temporal Logic. In Computer Aided Verification, LNCS 2725, Springer, 2003, pp. 394-406.

[6] M.C.W. Geilen. Formal Techniques for Verification of Complex Real-time Systems, Ph.D. Thesis, Eindhoven University of Technology, 2002

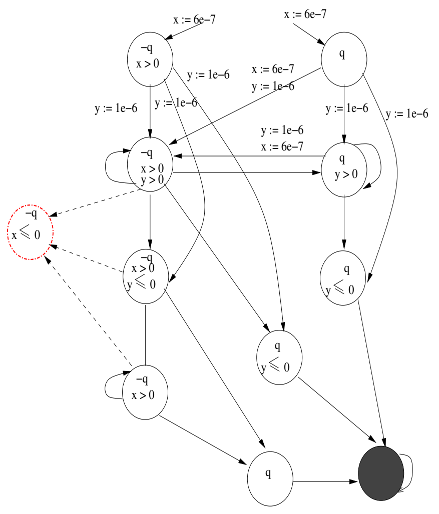
[7] M. R. Greenstreet, I. Mitchell. Reachability Analysis Using Polygonal Projections. In Hybrid Systems: Computation and Control, LNCS 1569, Springer. 1999, pp.103-116.

[8] S.Gupta, B.H. Krogh, R.A. Rutenbar. Towards Formal Verification of Analog Designs, IEEE/ACM International Conference on Computer Aided Design. 2004, pp.210 - 217.

[9] W. Hartong, R. Klausen, L. Hedrich. Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking, In Advanced Formal Verification, Kluwer, 2004, pp. 205-245.

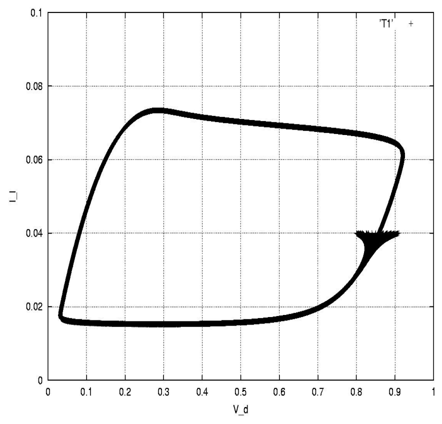


$(\forall\Box_{[0,1e^{-6}]}(\forall\Diamond_{[0,6e^{-7}]}(p = I_L \leq 0.02)))$

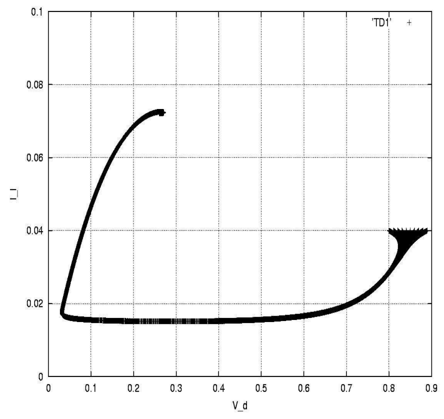


$(\forall\Box_{[0,1e^{-6}]}(\forall\Diamond_{[0,6e^{-7}]}(q = I_L \geq 0.06)))$

Figure 3: Monitor Automata



a. Oscillations



b. No Oscillation

Figure 4: Oscillator Behavior

[10] L. Hedrich and E. Barke. A Formal Approach to Verification of Linear Analog Circuits with Parameter Tolerances. IEEE/ ACM Design, Automation and Test in Europe, 1998, pp. 649-654.

[11] A. Lemke, L. Hedrich, and E. Barke, Analog Circuit Sizing Based on Formal Methods Using Affine Arithmetic. IEEE/ ACM International Conference on Computer Aided Design, 2002, pp. 486-489.

[12] R. J. Lohner, Enclosing the solutions of ordinary initial and boundary value problems, in Computer Arithmetic: Scientific Computation and Programming Language, Wiley-Teubner Series in Computer Science, Stuttgart, 1987, pp.255-286

[13] R.P. Kurshan and K.L. McMillan. Analysis of digital circuits through symbolic reduction. IEEE Transaction on Computer-Aided Design 10:pp.1350-1371, 1991.

[14] M. Maidl. The Common Fragment of CTL and LTL. Proc. Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 2000, pp.643-652.

[15] O. Maler, D. Nickovic, Amir Pnueli. Real Time Temporal Logic: Past, Present, Future. In International Conference on Formal Modelling and Analysis of Timed Systems, LNCS 3829, Springer, 2005, pp.2-16

[16] O. Maler, D. Nickovic. Monitoring Temporal Properties of Continuous Signals. In International Conference on Formal Modelling and Analysis of Timed System, LNCS 3253, Springer, 2004, pp.152-166

[17] R. E. Moore, Methods and Applications of Interval Analysis, Society for Industrial and Applied Mathematics, 1979.

[18] L. Tan, J. Kim, I. Lee. Testing and Monitoring Model-based Generated Program. In Proc. Run-time Verification, Electronic Notes in Theoretical Computer Science, 89(2), 2003