On Discrete Modeling and Model Checking for Nonlinear Analog Systems

Walter Hartong, Lars Hedrich, and Erich Barke

Institute of Microelectronic Circuits and Systems, University of Hannover
Appelstrasse 4, 30167 Hannover, Germany
{hartong,hedrich,barke}@ims.uni-hannover.de
http://www.ims.uni-hannover.de

Abstract. In this contribution we present a new method for developing discrete models for nonlinear analog systems. Using an adaptive state space intersection method the main nonlinear properties of the analog system can be retained. Consequently, digital model checking ideas can be applied to analog systems. To describe analog specification properties an extension to the standard model checking language CTL and the appropriate, algorithmic modifications are needed. Two nonlinear examples are given to show the feasibility and the advantages of this method.

1 Introduction

Formal verification for digital systems has a relatively long academic tradition. However, only a few years ago the capability of these tools has been raised to real world circuit sizes. Today, there are several commercial formal verification tools available and formal verification is used in many different disciplines.

There are also some tools which have been extended from digital to hybrid systems, i.e. to digital systems connected to some analog blocks or to an analog environment [1]. For some system classes these tools are quite successful, but they remain focused on the digital part of the system. The analog behavior is mostly restricted and the verification results are not appropriate to assess the functionality of the analog part. Furthermore, model checking languages used are not able to describe analog system properties.

One important paper to be named in this context is written by R. P. Kurshan and K. L. McMillan [2]. It is focused on digital system behavior but the circuit model used is based on transistors. Using a uniform state space intersection a discrete model is developed. The transition between state space regions is generated by following trajectories in the state space. We will see that these basic ideas are quite similar to the algorithms described in this contribution. Differences of these two approaches will be discussed later in more detail. Beyond that, the reachability analysis for nonlinear differential equations has been described in [3,4]. As far as we know, there is only one approach on equivalence checking for nonlinear analog systems [5].

This contribution presents a new model checking environment and a CTL extension which enables the work on pure nonlinear analog systems. The following chapter defines analog systems used in this context. The next part presents a discrete model for an analog system. Finally, a CTL extension is developed and the resulting changes in the model checking algorithms taken from well known digital tools are described. At the end, some experimental results are presented using small nonlinear examples.

2 System Description

The systems we will consider in this context are analog systems on the one hand and transition systems - as used in actual model checking tools - on the other hand. As we will see, an analog system is a transition system but the infinite number of states and the continuously defined state transition make these systems hardly accessible in an automatic way.

2.1 Analog Systems

The standard way to describe an analog system is a set of nonlinear first order differential algebraic equations

$$\mathrm{f}(\dot{x}(t), x(t), u(t)) = 0 \tag{1}$$

where $x(t)$ is the vector of system variables and $u(t)$ denotes the vector of input variables. In general, the function $f(\cdot)$ is arbitrarily nonlinear. However, in practice the nonlinearities are restricted by the device models used. There are several ways to build such equations for example from a transistor netlist or a behavioral model. The most common method is the modified nodal analysis (MNA). The set of state variables in an analog system is given by Equation (2).

$$x_s = \{x \mid \dot{x} \in \mathrm{f}(\dot{x}(t), x(t), u(t))\} \tag{2}$$

The coding of state variables spanning the state space depends on the way to build up Equation (1). It is not unique for one system [6]. Moreover, some state variables might be linearly dependent, so that the effective number of state variables may be smaller than the size of the set given in Equation (2). However, the number of independent state variables n is constant for a given system.

The link between state variables and system variables is not necessarily obvious like in the ODE case where $\dot{x}(t) = f(x(t), u(t))$. Mathematically, it is described by the system's differential index [7]. The initial value problem for differential algebraic equations can relatively easily be solved for systems of index 0 and 1. There are some approaches in solving systems with higher index, however this topic is still under research. Since we focus on basic model checking methods, this issue will not be discussed here. We assume the equation systems to be of index 0 or 1 with n linearly independent state variables. The state space spanned by the state variables $x_s \in \mathcal{R}^n$ is a continuous infinite Euclidean space.

2.2 Transition Systems

Digital and hybrid model checking tools are often based on transition systems.

Definition 1. *A state transition system $T = (Q, Q_0, \sum, R)$ consists of*
- *a set of states Q,*
- *a set of initial states Q_0,*
- *a set of generators or events $\sum$ and*
- *a state transition relation $R \subseteq Q \times \sum \times Q$.*

In fact, the analog system given above is also a state transition system. The set of states Q can be represented by the continuous state space $\mathcal{R}^n$. The number of states $x \in Q$ is always infinity, due to the continuous definition of the state variables. The initial state Q_0 is a single point or a region in the state space. Often, but not necessarily, this is the DC operating point. There are only $i+1$ generators $\sum$ causing state transitions, namely, the time t and the i input values $u(t)$. The state transition relation $R \subseteq \left(\mathcal{R}^n \times \mathcal{R}^{i+1} \times \mathcal{R}^n\right)$ is a continuous function given by the time derivation $\dot{x}(t)$ in Equation (1). The actual state transition can be calculated by integrating this function.

$$\mathrm{g}(x(t_0), u(t), \delta t) = x(t_0) + \int_{t_0}^{t_0+\delta t} \left\{ \dot{x}(t) \,\middle|\, f\left(\dot{x}(t), x(t), u(t)\right) = 0 \right\} dt \qquad (3)$$

Equation (3) has no direct time dependency but it depends on the input signals $u(t)$. Thus, the generators $\sum$ are not time t and input values but rather a time difference δt and the input values. Without losing generality, the time difference δt might be either an infinitesimal small or a finite value.

Thus, digital and analog systems can be described by transition systems. Unfortunately, the representation of states in the two system classes is totally different, so that an easy extension of digital tools to analog system seems to be difficult. Published model checking tools are already able to access discrete and partly linear system descriptions [1]. However, since our focus is on nonlinear analog behavior this is not sufficient.

Therefore, a new method has to be developed. Some of the following algorithms have been inspired by research in the area of approximating dynamical behavior [8]. Despite the similarities, there are a lot of differences, mainly caused by the overall target of the algorithms. Algorithms from this area have not been used, but some of the ideas have been adopted. We have already mentioned the work of Kurshan and McMillan [2] which is also linked to following algorithms.

3 Discrete Model Generation

As we have seen, the continuous variables in an analog system - state values and time - have to be transfered into a discrete state space description and a state transition relation. The next sections illustrate this process.

3.1 Discrete Time Steps

In Section 2.2 we found that the transition relation $R \subseteq \left(\mathcal{R}^n \times \mathcal{R}^{i+1} \times \mathcal{R}^n\right)$ for an analog system is given as a continuous function and the actual state transition can be calculated by integrating this function. In general, Equation (3) can only be solved using numerical integration. This problem is well known in analog simulators, like Spice, Spectre, Saber, etc. During transient simulation the differential equation system is solved using discrete time steps. Given a small time step $\Delta t = t_1 - t_0$, the transition between the actual state $x(t_0)$ and the next state $x(t_1)$ is determined by a numerical integrator, e.g. the backward Euler formula:

$$\mathrm{g}_{\mathrm{num}}\left(x(t_0), u(t_1), \Delta t\right) = \left\{ x(t_1) \,\middle|\, f\left(\frac{x(t_1) - x(t_0)}{\Delta t}, x(t_1), u(t_1)\right) = 0 \right\} \tag{4}$$

Disregarding numerical problems, there will always be an error due to the finite length of Δt. To bound this error, a local step size control mechanism is needed. The algorithm used take the second derivation with respect to time for a local measurement of the integration error. If the given error threshold is exceeded, step size is reduced otherwise the transient step is accepted. This method can be used directly in the analog model checking tool. An arbitrary test point $x(t)$ in the state space is mapped to its successor state $x(t + \Delta t)$, depending on the actual step size Δt and the input signals $u(t)$. In contrast to transient simulation, there is no temporal predecessor state for a test point. A second time step has to be calculated for each point to determine the second derivation, enabling a local error control.

In general, the time step Δt will vary throughout the state space, due to the step size control. As we will see later, this makes the checking of explicit time dependencies difficult because one has to store Δt for each transition separately. To make this easier, the time step is chosen to be equal for each point within one state space region (to be defined in Section 3.3). Kurshan and McMillan [2] proposed a constant time step Δt for the whole space developed by several small numerical integration steps (segments of trajectories). Despite the advantage of a constant time step, this is not suitable for all circuits since the step size variation in terms of state variable values may be large throughout the state space.

Thus, every state space point $x(t)$ can be mapped to its successor point $\mathrm{s}(x(t)) = \mathrm{g}_{\mathrm{num}}(x(t), u_{const}, \Delta t_{length\_control})$ including a local step length control and assuming given input values. The resulting tuple of test and target point is represented by a successor vector $\mathrm{sv}(x(t)) = \mathrm{s}(x(t)) - x(t)$ in the state space.

3.2 Input Value Model

To solve Equation (3) or (4) the input value $u(t)$ is needed. Until now we have not defined this value. In principle, the input signals might be defined explicitly. However this is not really useful since the model checking result will only be true for one specific input signal and this is a contradiction the formal verification idea. Let us therefore assume some conditions for the input signal without defining it explicitly. To do this, the state space is extended by the input variables $x_{si} = \{x_s, u\} \in \mathcal{R}^{n+i}$. It is called extended state space. Thus, every state within the extended state space contains information about the actual input values. However, there is no information on the input value change with respect to time. Moreover, it is theoretically impossible to predict the input value variation because the input values are not determined by the system itself but rather from some outside systems.

There are two extreme assumptions: The input values do not change at all and the input values may change instantaneously over the whole input value range. For the first assumption, the model is build up as described before for several constant input values. There will be no transition between states with different input values. In the second approach, a state space region has not only transitions to regions at the same input level but additional transitions to the neighbor regions in terms of input values. By using the extended state space and the described input model the transition relation changes to $R \subseteq \left(\mathcal{R}^{n+i} \times \mathcal{R}^1 \times \mathcal{R}^{n+i}\right)$.

As we will see later, both of these input models are useful for certain conditions to be checked. Between these two extreme models it possible to assume the input values to vary within a given frequency range or within a maximum input voltage slope. This is the most suitable assumption for real world systems but it has not been considered in the prototype tool yet.

3.3 State Space Subdivision

To get a discrete and finite state description, the continuous and infinite state space has to be bounded and subdivided. This is done by rectangular boxes. In general, boxes are not necessarily the best choice [9], however, for implementation reasons boxes are the far most convenient data structure. Other subdivision geometries might be considered during future improvements.

The restriction to a finite region is simply done by a user defined start area, comprising the considered system behavior. This causes special border problems, which will be discussed later. However, it does not impact the correctness of the model checking result because in real world systems there is alway a natural bound for the state variable values.

Since we do not have a digital environment, a natural subdivision for the start area, given for example by threshold values of digital state transitions, is missing [1]. Furthermore, to retain the analog system behavior correctly, a sufficient number of subdivisions is needed, especially in state space regions with highly nonlinear behavior. This purpose differs from approaches focusing on digital circuit behavior [2]. However, the number of discrete regions should be kept as small as possible to reduce the total runtime.

We start with a user controlled uniform subdivision in all state space dimensions. Then, an automatic subdivision strategy is used to react on different system dynamics, depending on the actual state space region. The main target is to get a uniform behavior in each state space box. The uniformity is measured by the variation of the successor vectors (sv(·)), calculated in the state space (Section 3.1 and 3.4). Namely, vector length variation l_m and angle a_m between different vectors are considered. Equations (5) and (6) give the definition of these values. The function L(·) gives the length of the delivered vector or vector component. Input value variations are not taken into account.

$$l_m = 1 - \frac{\min_{y \in p_{test}} \mathrm{L}(\mathrm{sv}(y))}{\max_{y \in p_{test}} \mathrm{L}(\mathrm{sv}(y))} \tag{5}$$

$$a_m = \max_{\substack{y \in p_{test} \\ k \in n}} \frac{\mathrm{L}(\mathrm{sv}(y)_k)}{\mathrm{L}(\mathrm{sv}(y))} - \min_{\substack{y \in p_{test} \\ k \in n}} \frac{\mathrm{L}(\mathrm{sv}(y)_k)}{\mathrm{L}(\mathrm{sv}(y))} \tag{6}$$

Box subdivision is continued recursively until l_m and a_m drop under a given threshold or a given subdivision depth is exceeded.

Within the expected accuracy, all boxes fulfilling the l_m and a_m thresholds do not contain fix points. This is, because fix points are always surrounded by regions with nonuniform behavior in terms of Equations (5) and (6). This information is stored and used in the transition relation algorithm (Section 3.4).

Additional subdivisions are applied if the successor vectors in a region are too short in relation to the box size. This occurs mainly in regions where the system is strongly nonlinear, which implies Δt to be very small. Each box in the state space will represent a single state in the discrete model. Thus, the set of i states is given by $Q = \{box_1, box_2, ..., box_i\}$.

3.4 Transition Relation

The last step in getting a discrete system model is the transition relation between state space regions. In Section 3.1 successor points for single state space points have been defined (s(·)). Using this point to point relation, the target region r_{target} is given by the set of all target points associated with a test point within the state space region r_{test} (see Equation (7)), as illustrated by the gray regions in Figure 1. We call this exact transformation $\mathrm{T}_1(\cdot)$.

$$r_{target1} = \{\mathrm{s}(y) \mid y \in r_{test}\} = \mathrm{T}_1(r_{text}) \tag{7}$$

Since it is not practical to calculate a huge or - mathematically - infinite number of successor points for each box, a good estimation or inclusion of the target region is needed. Three different approaches will be discussed.

An inclusion $r_{target2}$ can be calculated using interval analysis [10]. This approach provides an overestimated solution. That means, the correct solution $r_{target1}$ is fully included within $r_{target2}$. However, $r_{target2}$ might be much larger than $r_{target1}$. This effect is called overestimation and might be a serious problem especially for large systems [5].

Interval analysis has already successfully been used for hybrid systems [11].

$$r_{target2} = \mathrm{T}_{interval}(r_{test}) = \mathrm{T}_2(r_{test}) \supseteq r_{target1} \tag{8}$$

A more practical but also less accurate way to approximate the target region is to choose a number of test points p_{test} within the test region (e.g. randomly, grid based, or corner values) and to calculate the dedicated target points p_{target}. The target region $r_{target3}$ can be approximated using an appropriate inclusion of these points. As we will see below, an inclusion operation is also needed while expanding the target regions to the actual state space regions. These two steps can be combined. Even a few test points may give a reasonable target approximation, but the region $r_{target3}$ might be under- or overestimated.

$$p_{test3} = \{s_1, s_2, ..., s_n\}\,,\ s_i \in r_{test} \tag{9}$$

$$r_{target3} = \{\text{inclusion}(s(y)) \mid y \in p_{test3}\} = \mathrm{T}_3(r_{test}) \simeq r_{target1} \tag{10}$$

There are two approaches making this process rigorous which means that the target region $r_{target1}$ is fully included in the target approximation. At first, it is shown in [2] that T_3 is surely overestimated if all corner values are used as test points and if s(·) can be assumed to be monotonic. Secondly, following the argumentation in [8], this is done using Lipschitz constants L in each state space dimension. Using a grid of test points, spaced by h, one can calculate an extension diameter $d_{ex} = Lh$ for the target points. Expanding each target point by this diameter d_{ex} in each dimension gives a set of boxes. The union of these boxes is an overestimated target approximation $r_{target4}$. In Figure 2 three test and target points and the dedicated extension boxes are shown.

$$r_{target4} = \left\{\text{expand}_{(Lh)}(s(y)) \mid y \in \text{grid}\,(h, r_{test})\right\} = \mathrm{T}_4(r_{test}) \supseteq r_{target1} \tag{11}$$

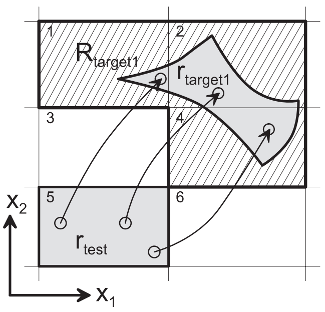

Fig. 1. State space transition relation using T_1

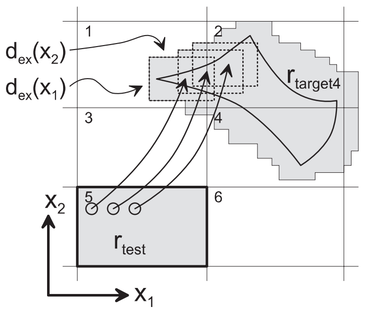

Fig. 2. Rigorous approach using Lipschitz constants

All discussed target regions ($r_{target1}$, $r_{target2}$, $r_{target3}$, and $r_{target4}$) do not fit into the state space subdivisions used. Therefore, a second step is needed to extend these regions to legal sets of boxes. For example, region $R_{target1}$ (hatched areas in Figure 1) is given by the set of all boxes having contact with the target region $r_{target1}$. Fortunately, this operation is always an overestimation and does therefore not impact the correctness of the above results. Until now, only the third operation $\mathrm{T}_3(\cdot)$ has been implemented. $\mathrm{T}_2(\cdot)$ and $\mathrm{T}_4(\cdot)$ will follow in further implementation steps.

Some additional steps are needed to optimize the transition relation for some corner cases. Namely, these are prevention of long successor vectors, resulting in a box over-jump, boxes with self-connection and boxes with no transitions to other boxes due to short successor vectors. The last two conditions are unphysical if the box does not contain fix points (Section 3.3). As we have already mentioned in Section 3.1, no explicit time relations are considered. It might be useful or necessary in future implementations to store not only the transition relation $R \subseteq Q \times Q$ but rather this relation combined with the related transition time delays $R \subseteq Q \times Z \times Q$ where Z denotes the set of all transition time delays used.

4 Model Checking Algorithms

Due to the developed discrete model, the analog system is accessible by discrete algorithms. In this contribution a CTL model checker will be discussed but the foregoing approach can also be used for other formal verification approaches.

As we will see, existing model checking tools and languages are not well suited for the generated models and the description of analog properties. In particular, the intensively used BDD structures are not helpful for this kind of models, because the set of states can not efficiently be described by binary state variable combinations. Therefore, a modified model checker has been developed, based on the basic CTL algorithms described in [12]. A simple tree structure is used to store the discrete state regions. There are some algorithmic modifications due to the special need of the analog model, explained below. The language has been extended by a minimal set of operations enabling the work on analog models. Additionally, the results are visualized graphically. The meaning of the CTL operators is the same as in digital model checkers. Table 1 gives a short syntax overview on the classical CTL language.

For example the formula $\Theta = \mathsf{AF}(state_1)$ can be read as follows: All paths starting in a state within Θ will eventually reach a state in which $state_1$ is true. To simplify matters, we do not distinguish between a CTL condition and the set of states that fulfills this condition, both will be named by capital Greek letters.

Since the domain of digital state variables is restricted to boolean values, the statements a and $\neg a$ cover the whole domain. As we have seen before, analog state variables are defined continuously. Thus, the given CTL definition is not sufficient for describing condition in analog variables. To solve this problem, we introduce a greater and smaller operator in the language definition (Table 4). Thereby, half planes can be described in a continuous space e.g. ($x_1 > -13.2546$).

Table 1. Classical CTL syntax

$\phi := a \mid \phi \circ \phi \mid \neg\phi \mid \triangleright \diamond \phi \mid \triangleright \phi \,\mathsf{U}\, \phi$

a	boolean variable	
$\circ$	boolean operators	$\vee \rightarrow$ or
		$\wedge \rightarrow$ and
$\neg$		$\neg \rightarrow$ not
$\triangleright$	path quantifiers	$\mathsf{E} \rightarrow$ on some path
		$\mathsf{A} \rightarrow$ on all paths
$\diamond$	temporal operators	$\mathsf{X} \rightarrow$ next-time
		$\mathsf{F} \rightarrow$ eventually
		$\mathsf{G} \rightarrow$ always
U		$\mathsf{U} \rightarrow$ until

Table 2. CTL syntax extension

$\phi := b * v \mid \phi \circ \phi \mid \neg\phi \mid \triangleright \diamond \phi \mid \triangleright \phi \,\mathsf{U}\, \phi \mid \mathsf{iv}\phi$

b	continuous variable	
v	real value	
$*$	analog operators	$> \rightarrow$ greater
		$< \rightarrow$ smaller
iv	inverse time	

The combination of several half planes with boolean operators enables the definition of arbitrary Manhattan polytopes in a continuous n-dimensional space. Boolean variables are left out in this definition, because only pure analog system are considered in this contribution. However, an extension to hybrid systems seems to be possible.

Moreover, we have introduced the inverse time operator iv. It simply inverses all transition relations with respect to time. By this, we assume a branching (also known as non-Ockhamist) infinite past. A collection of other past time definitions and languages can be found in [13]. As we will see in the experimental results, this operator is useful for some analog properties to be checked. Due to these extensions, the meaning of all operators has to be reviewed to find out the necessary changes in known digital or hybrid model checking algorithms.

- If the threshold values v used in the greater/smaller operations are not subdivision values in the state space, they have to be added and the transition relation has to be reconstructed due to this change before executing the CTL formula. This makes the discrete model not only dependent on the analog system but also on the CTL formula used.
- The boolean operators $\neg$, $\vee$, and $\wedge$ are obviously defined for state space regions. However, as we will see later, not only the restricted state space has to be considered but also the outside area. In this context the definition will be extended.
- In contrast to the original analog model the discrete model will be nondeterministic due to the abstraction of the state space, namely because a state space region may be connected to several successor regions even without changing the input values. This does not effect the path quantifier definition but it has to be considered while using them in a CTL formula. Since the model is nondeterministic, the inverse time operator does not change the model structure in respect thereof.
- The last group comprises temporal operators. As we have seen before, the time step Δt may vary throughout the state space. Thus, the time quan-

tifier X has only a qualitative meaning. To enable checking of quantitative time dependencies the discrete model and the language needs further extensions (e.g. $\mathsf{X}_{(3\mu s)}$, see also [14]). The other operations can be used in the same manner as in digital applications.

Obviously, this language is not very powerful describing analog design specifications. This is because we want to follow the digital model checking ideas as far as possible without major changes. It will be shown below that it is possible to check some analog properties even with this minimal set of operations.

4.1 Border Problems

Consider for example differential equation $\left\{\dot{x} = (1, 0)^T\right\}$ and CTL formula $\mathsf{EG}(\Theta)$ where $\Theta = ((x_2 > 1.0)\ \&\ (x_2 < 2.0))$. For this example Equation (3) is easy to solve symbolically $x(t_0 + \delta t) = x(t_0) + \int_{t_0}^{t_0+\delta t} \dot{x}\, dt = x(t_0) + \binom{1}{0} \delta t$.

It is obvious, that $\mathsf{EX}(\Theta) = (\Theta)$ for all δt because a time step causes only a shift in x_1 direction and since region Θ is not restricted there, the EX operator does not affect that area. Furthermore, $\mathsf{EG}(\Theta)$ is the largest fix point of the sequence $\{\Theta_0 = \Theta; \Theta_{i+1} = \Theta_i \wedge \mathsf{EX}(\Theta_i)\}$. Consequently, the theoretical result of $\mathsf{EG}(\Theta)$ is Θ. If a restricted state space - for example $([-5\ ..\ 5], [-5\ ..\ 5])^T$ - is applied to this example the result changes dramatically. Using the solution of $x(t_0 + \delta t)$ we find $\mathsf{EX}(\Theta)$ to be $([-5\ ..\ (5 - \delta t)], [1\ ..\ 2])^T$. Thus, the largest fix point for the sequence defining EG is $\emptyset$.

Is that the expected result? To answer this question the meaning of $\Psi = \mathsf{EG}(\Theta)$ has to be studied again: “For each state ψ within Ψ there is a path starting in ψ such that Ψ is invariantly true on this path.” In the given example, every path leaves the restricted state space after some time but that does not necessarely mean that Ψ is not fulfilled on that path since Ψ is also restricted to the given state space. Thus, it has to be defined whether a path leaving the restricted state space fulfills that condition or not.

In other words, it has to be defined whether the area outside the restricted state space is part of the actual region or not. In the above example we have simply assumed that the outside area is not part of the region. Under this assumption the given result is correct. But if we assume the outside area to be part of the region $\Theta = \left(([-5\ ..\ 5], [1\ ..\ 2])^T + \mathit{Outside}\right)$ we get $\mathsf{EG}(\Theta) = \Theta$.

To implement that, an outside area flag is stored for each region used. The border boxes are treated specially, they keep the value given by the border flag. As a result, the border boxes are not part of the model checking result and have to be omitted during interpretation. Every new region to be introduced has an outside flag set to false. All boolean operations are not only applied to the state space but also to the outside flag. Thus, it is possible to define all constellations.

5 Experimental Results

5.1 Biquad Lowpass Filter Example

The first example, a 2^{nd} order Biquad lowpass filter, is shown in Figure 3. The opamp model has an open loop gain of 10000. The output voltage restriction is ± 1.5 V and the maximum output current is 80 mA. Due to these restrictions the whole system is nonlinear. This circuit has two state variables, namely the capacitor voltages V_{c1} and V_{c2}. Using a charge oriented capacitor model will normally lead to the two charges as state variables. We changed this, because we found it more convenient to think in voltages than in charges. The corner frequency ω_c and the damping factor d are given by $\omega_c = \sqrt{R_1 R_2 C_1 C_2}^{-1}$ and $d = 0.5 C_1 \omega_c (R_1 + R_2)$. We use two different value sets for the resistors and capacitors, one with $\omega_c = 100\ s^{-1}$ and $d = 0.5$ and the second one at the same frequency but with $d = 2$. Thus, we get two equivalent lowpass filters differing only by the damping factor. The property to be checked in this example is the occurrence of overshooting in the two filters. Since these properties should be proved for arbitrary input signals, the appropriate input value model is chosen. The input signal range is $V_{in} = [-2\ ..\ 2]$, so that the nonlinearity due to the restricted output voltage will effect the system behavior. The state space is restricted to $V_{c1} = [-4\ ..\ 4]$ and $V_{c2} = [-2.5\ ..\ 2.5]$.

The initial state in this example is assumed to be the DC operating point at $V_{in} = 0$. The question is: Which states are reachable from this point for arbitrary input signals within the given range? Instead of a single start point a start area is used, that is for example a box surrounding the initial point. Next, operation EF is used to check which states have a path that will eventually reach the starting area. However, the direction of this operation is wrong. Inverting the time gives the correct formula $\{\Phi_3 = \mathsf{iv}(\mathsf{EF}((V_{c2} < 0.5)\ \&\ (V_{c2} > -0.5)\ \&\ (V_{c1} < 0.5)\ \&\ (V_{c1} > -0.5)\ \&\ (V_{in} < 0.5)\ \&\ (V_{in} > -0.5)))\}$.

This equation is applied to both circuits. The result for the highly damped circuit is shown in Figure 4. The black box indicates the state space borders. The input voltage axis is perpendicular to the paper. As expected, the state V_{c1}

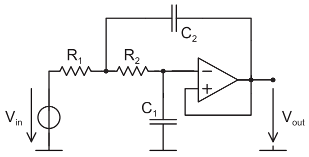

Fig. 3. 2^{nd} order lowpass filter

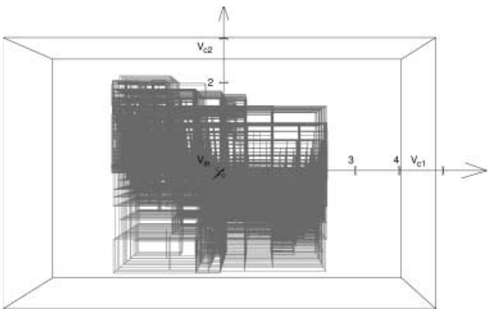

Fig. 4. Model checking result Φ_3

remains within a range of ±2 V whereas it has been shown that it reaches higher levels in the less damped case. However, the output voltage V_{out} remains within the ±1.5 V range for both circuits due to the opamp nonlinearities. It turns out that the opamp output restriction of ±1.5 V does not have an impact on this result because states V_{c1} and V_{c2} are not restricted by the opamp output.

5.2 Tunnel Diode Oscillator Example

The analog system used in our second example is a simple tunnel diode oscillator circuit shown in Figure 5. The input voltage V_{in} is set to 2.6 V. In this operating point the circuit starts an oscillation automatically. The bounded state space is given by $V_C = [\,-0.2 \mathinner{..} 4.4\,]$ and $I_L = [\,-0.2 \mathinner{..} 4.0\,]$.

According to digital systems a stable oscillation might be proved by the following CTL equation $\Phi_5 = \{\mathsf{AG}(\mathsf{AF}(I_L > 2.2))\ \&\ \mathsf{AG}(\mathsf{AF}(I_L < 1.6))\}$. The collection of boxes fulfilling this condition is shown in Figure 6 in light gray. Except of some border boxes and the middle region, the whole state space is covered.

We can conclude that nearly the whole plane will float into an stable orbit. The next question might concern the possible orbit geometry. We generate this by applying $\Phi_6 = \{\mathsf{iv}(\mathsf{EG}(\Phi_5))\}$. The result Φ_6 contains the whole orbit calculated by an ordinary simulation (black line in Figure 6).

6 Conclusion

To apply digital model checking ideas to analog systems a discrete system model is needed. The main algorithmic task is to develop a state transition model in such a manner that the main nonlinear and dynamic properties of the analog system are retained. This is done using an automatic state space subdivision method and an algorithm developing the transition relation.

The implemented CTL model checker is based on digital algorithms, but it is extended by some operations, enabling the definition of analog properties in

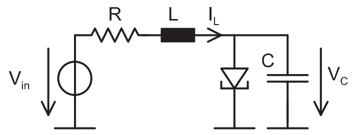

Fig. 5. Tunnel diode oscillator

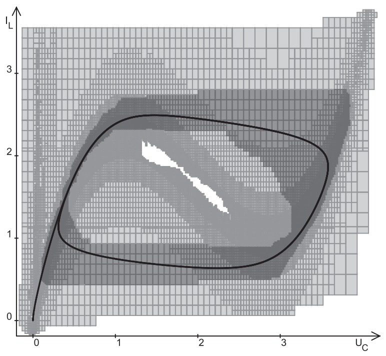

Fig. 6. Model checking results Φ_5 and Φ_6

CTL. Some applications of this language are shown by the experimental results. However, applying CTL to analog systems, seems even more uncommon than doing so in digital. It is clear that the language is not sufficient for all analog properties. Especially, explicit time dependent properties, like slew rates or delays, are not covered in the prototype implementation.

As far as we know, the presented tool is the first approach to model checking for nonlinear analog systems. That opens a wide range of possibilities in applying formal methods not only to digital and hybrid systems but also to analog systems. Therefore, it is a step towards a more formalized analog design flow.

References

1. Alur, R., Henzinger, T., Lafferriere, G., Pappas, G.: Discrete abstractions of hybrid systems. Proceedings of IEEE (2000 971-984 401, 403, 405
2. Kurshan, R., McMillan, K.: Analysis of digital circuits trough symbolic reduction. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 10 (1991 1356-71 401, 403, 404, 405, 407
3. Dang, T., Maler, O.: Reachability analysis via face lifting. HSCC '98: Hybrid Systems: Computation and Control, LNCS (1998 96-109 401
4. Asarin, E., Bournez, O., Dang, T., Maler, O.: Approximate reachability analysis of piecewise-linear dynamical systems. HSCC '00: Hybrid Systems: Computation and Control, LNCS (2000 76-90 401
5. Hedrich, L., Hartong, W.: Approaches to formal verification of analog circuits. In Wambacq, P., ed.: Low-Power Design Techniques and CAD Tools for Analog and RF Intergrated Circuits. Kluwer Academic Publishers, Boston (2001 155-191 401, 406
6. Giinther, M., Feldmann, U.: Cad-based electric circuit modeling in industry, part is Mathemetical structure and index of network equations. Suveys on Mathematics for Industry 8 (1999 97-129 402
7. März, R.: Numerical methods for differential algebraic equations. Acta Numerica (1991 141-198 402
8. Dellnitz, M., Froyland, G., Junge, O.: The algorithms behind gaio - set oriented numerical methods for dynamical systems. Ergodic Theory, Analysis, and Efficient Simulation of Dynamical Systems (eds. B. Fiedler, Springer (2001 145-174 403, 407
9. Henzinger, T., Ho, P.H.: Algorithmic analysis of nonlinear hybrid systems. CAV '95: International Conference on Computer-Aided Verification, LNCS 939 (1995 225-238 405
10. Neumaier, A.: Interval methods for systems of equations. Cambridge University Press, Cambridge (1990 406
11. Henzinger, T., Horowitz, B., Majumdar, R., Wong-Toi, H.: Beyond hytech: Hybrid systems analysis using interval numerical methods. HSCC '00: Hybrid Systems: Computation and Control, LNCS (2000 130-144 406
12. Burch, J., Clarke, E., Long, D., McMillian, K., Dill, D.: Symbolic model checking for sequential circuit verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 13 (1994 401-424 408
13. Laroussinie, F., Schnoebelen, P.: Specification in ctl+past for verification in ctl. Information and Computation 156 (2000 236-263 409

14. Emerson, E., Mok, A., Sistla, A., Srinivasan, J.: Quantitytive temporal reasoning. CAV '90: International Conference on Computer-Aided Verification, LNCS (1990 136-145 410