Targeting the Analog Verification Gap: State Space-based Formal Verification Approaches for Analog Circuits

Sebastian Steinhorst · Lars Hedrich

Abstract Analog circuit verification lacks systematization and formalization. The major weakness of todays verification flows is their uncertainty about verification coverage. This contribution demonstrates how novel approaches can overcome this coverage problem by considering the complete analog state space for verification. An approach to model checking proves the absence or existence of errors with respect to a machine-readable property specification. By injecting particles into the continuous vector field representing the state space of analog circuits and inspecting their simulated parallel motion according to the circuit's dynamics, another complete and therewith formal verification approach is given. The application to a circuit example shows the feasibility of both approaches.

1 Introduction

Due to the increasing system complexity and decreasing time to market, verification of analog circuit designs has become a more and more crucial part of the design flow. While formal verification methods are established in the digital domain, industrial analog circuit design flows are lacking formal or at least formalized verification methodologies.

Analog circuit verification still depends on the designer's experience and expertise to manually define appropriate test benches for simulation-based design flows and to select the right input signals in order to detect possible design errors. Driven by the perennial demand for higher design efficiency, new approaches offering more automation of the verification process are of vital importance.

This contribution describes two approaches to analog circuit verification that cover the complete state space of the circuits and therewith are considered as formal verification.

Sebastian Steinhorst · Lars Hedrich
Electronic Design Methodology
Institute for Computer Science
Goethe University of Frankfurt/Main
Robert-Mayer-Str. 11-15
D-60325 Frankfurt/Main, Germany
E-mail: [steinhorst, hedrich]@em.cs.uni-frankfurt.de

2 Quantifying the Analog Verification Gap: Verification Coverage

There has been a significant progress in several areas of electronic design automation (EDA) for analog circuits. Some complex tasks such as placement, sizing, and design centering have been addressed by EDA-vendors, now being available as automated tools which are fully integrated into the design flow. These tools are exploiting algorithmic concepts which by far outperform manual approaches.

By contrast, the area of analog design verification is not systematically covered by existing tools. While approaches to assertion-based simulation are emerging, which are mainly automating previosly manual efforts, they are not targeting the fundamental problem of analog circuit verification: verification coverage. Todays established common verification methodology is analyzing the circuit's behavior by simulation using test benches. Specification conformance is checked by performing several transient simulations with input signals which are considered representative for the future operating conditions of the circuit. Although this approach to discover design errors has been working for decades, redesigns have occurred frequently due to missing some critical behavior of the circuit during simulation.

The approaches to formal verification of analog circuits have in common that they target the verification coverage problem of user-defined transient simulations. Figure 1 illustrates the problem of conventional transient simulations not covering the complete reachable state space of a circuit and summarizes the main characteristics of this approach. In contrast, formal verification approaches cover the complete state space of the circuit, resulting in absolute confidence concerning the verification results, with the downside of requiring new verification paradigms.

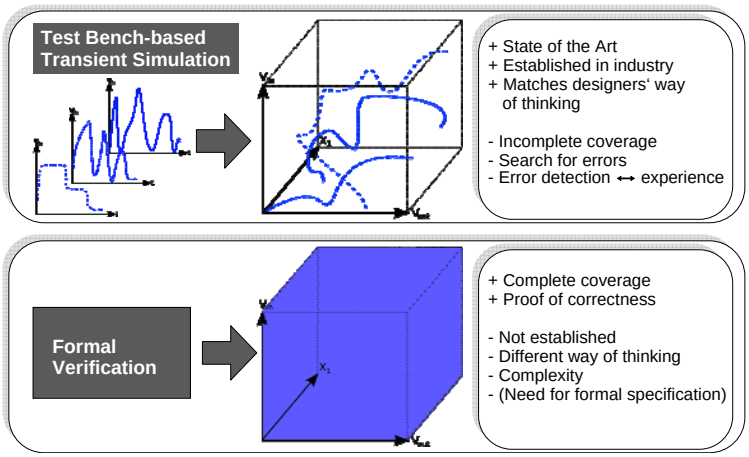


Fig. 1 Comparison of Verification by Testbench-based Transient Simulation and Formal Verification.

3 State Space Representation

The key to the approaches of formal verification of analog systems presented in the following is to create a state space representation of the circuit's behavior. By applying modified nodal analysis (MNA) to the circuit netlist, the circuit is represented by a nonlinear first order differential algebraic equation (DAE) system

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

The state space is spanned by the input vector $u(t)$ and the vector of the system variables $x_e(t)$, representing the energy-storing elements of the circuit, such as capacitances and inductances.

For a better understanding of the state space representation, the transient response for I_L of a tunnel diode oscillator circuit, shown in Figure 2, is plotted over time in Figure 3(a). For both system variables I_L (y-axis) and V_C (x-axis) of this circuit, the transient response is plotted in Figure 3(b) in state space representation with transition time implicitly shown with a color mapping. Figure 3(c) shows the complete discrete vector field of the tunnel diode oscillator's state space, which is defined by the system variables' derivatives $\dot{x}_e(t)$.

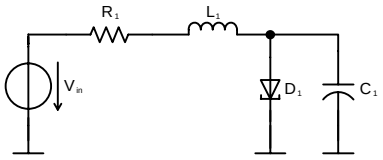
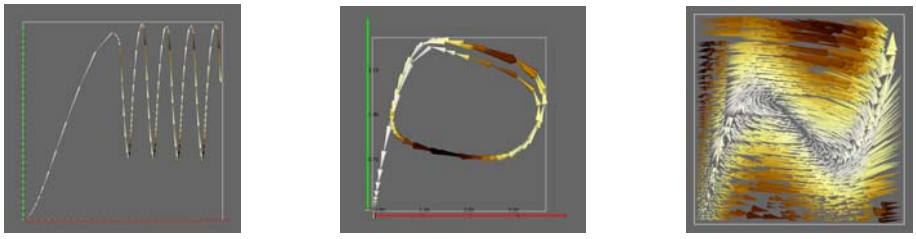


Fig. 2 Schematic of tunnel diode oscillator circuit.



(a) (b) (c)

Fig. 3 Transient response of the oscillator's system variable I_L plotted over time (a). Transient response of the two system variables I_L and V_C in state space representation (b). Vector field representation of complete state space over I_L and V_C (c).

4 Model Checking

The objective of Model Checking is to check automatically whether or not specified circuit properties meet the specification, written in a machine-readable specification

language. For analog circuits, specifications are written in informal documents, describing the required operating performances of the circuit.

To apply model checking algorithms, the state space representation is transferred into a discrete graph structure. Therefore, the continuous state space is divided into hypercubes of homogeneous behavior by comparing length and angle of the system variables' derivatives $\dot{x}_e(t)$. These derivatives also define the transition relation of the hypercubes and the transition time. By considering each hypercube as a vertex of a graph, connected according to the transition relation with the corresponding transition times, a graph data structure is generated from this information. The vertices of the graph are labeled with the state space variable values at the center of the corresponding hypercube. Figure 4 illustrates the discrete modeling process, performed by the analog modeling tool *amcheck* [1].

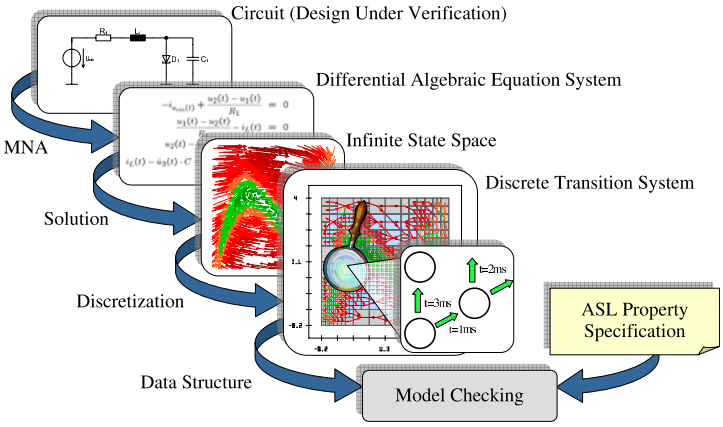


Fig. 4 Generation of a discrete model for nonlinear analog systems for model checking.

To the generated graph structure, analog model checking algorithms in conjunction with a special Analog Specification Language (ASL) [2] are applied. Compared to approaches using temporal logic specification [3,4], the number of verifiable circuit properties such as Offset, Gain, CMRR, PSRR, Slew Rate, Overshoot, Startup Time, Oscillation, and VCO Input Sensitivity is increased significantly.

4.1 Analog Specification Language

Up to now, property specification for formal verification approaches of analog circuits was based on extended versions of temporal logics such as RTCTL [5] or CTL-AT [6]. Due to its origin in temporal logics, CTL is not capable of offering a designer-oriented specification methodology. In order to gain acceptance for formal approaches in

analog verification, a new methodology of property specification is necessary. While the Property Specification Language (PSL) [7] offered this step towards designer-oriented specification in the digital domain, an approach to analog specification with PSL [8] covers signal-based properties for assertion-based verification but is not designed for describing properties of analog systems in the state space. The approach of Mixed-Signal Assertions (MSA) [9] extends temporal logic specification with operations for describing properties of mixed signal systems but again has not been designed for analog state space specification.

ASL syntax is designed to be semantically deductive and therewith reduces the time needed for understanding existing specifications. By providing the possibility of creating parameterized macro functions involving a macro preprocessor, specification code can be sourced out to macro libraries allowing encapsulation and reuse of specification code. The application of ASL specifications and algorithms is not restricted to state space models generated with the approach described in the preceding section, so it can be adopted to other finite state machine-based modeling approaches like [10].

From the point of view of analog circuit developers, analog properties are represented by continuous physical values and their alteration over time. Thus, it is necessary to select states by calculations on their state space parameter values. Whether a state belongs to a set is decided by comparing the result of an arithmetic calculation to a specified interval. Extended path operations abstract from the reachability analysis concept of temporal logics and allow examination of more complex properties on paths within state space. In table 1 the analog circuit properties specifiable with ASL compared to the possibilities given by CTL-AT are summarized. Oscillation and startup time properties are explained in the next sections in more detail.

The following subsection defines an Extended Backus Naur Form (EBNF) grammar for the syntax of ASL with terminal symbols printed in bold capital letters. User-defined variables are printed italic.

Table 1 Analog circuit properties specifiable and verifiable with ASL compared to CTL-AT. $\surd$ means fully specifiable and verifiable, ($\surd$) means verifiable only by manual iterative checking of specified assumptions.

Circuit property	**ASL**	**CTL-AT**
Reachable Area	$\surd$	$\surd$
Offset	$\surd$	($\surd$)
Gain	$\surd$	
CMRR	$\surd$	
PSRR	$\surd$	
Slew Rate	$\surd$	($\surd$)
Overshoot	$\surd$	$\surd$
Startup Time	$\surd$	
Oscillation	$\surd$	($\surd$)
VCO Input Sensitivity	$\surd$	

4.2 EBNF Grammar of ASL

ASL_Specification :=

Spec_Sequence **QUIT**; | **QUIT**;

Spec_Sequence :=

Spec_Expression | Spec_Sequence Spec_Expression

Spec_Expression :=

| **SETVAR** *Set_Variable*;
| **NUMVAR** *Number_Variable*; | *Set_Variable* = Set_Expression;
| *Number_Variable* = Number;
| **FOR** Set_Expression **ASSERT** Set_Expression;
| **FOR** Number **ASSERT** Interval;
CALCULATION *Calc_Name* ("Calc_Expression");

Set_Expression :=

ON Base_Set Operation_Set | Operation_Set

Base_Set :=

Elementary_Set | *Set_Variable* | *State_Space_Variable* Interval
| (Base_Set) | **NOT** Base_Set | Base_Set **AND** Base_Set
| Base_Set **OR** Base_Set

Operation_Set :=

Elementary_Set | *Set_Variable* | *State_Space_Variable* Interval
| (Operation_Set) | **NOT** Operation_Set
| Operation_Set **AND** Operation_Set
| Operation_Set **OR** Operation_Set
| Operation_Set Operation_Set
| **SELECT** Operation_Set
| *Calc_Name* (Parameter) Interval
| **VALUE** (*State_Space_Variable*) Interval
| **ASSIGN** (*Number_Variable*, Assign_Type)
| **OSCILLATION** | Temporal_Logic_Expression Operation_Set
| **DELTA_COMPARE** (*State_Space_Variable*) Interval **FROM** Operation_Set **TO** Operation_Set
| **TRANSITION FROM** Operation_Set **TO** Operation_Set

Elementary_Set :=

ALL | **STEADYSTATES**

Interval :=

[Number, Number] | [< Number] | [<= Number] | [> Number] | [>= Number]

Temporal_Logic_Expression :=

Temporal_Logic_Operator Direction Operation_Set
| **ALWAYS** Direction Operation_Set **UNTIL** Operation_Set
| **A** Direction Operation_Set **U** Operation_Set
| **EXISTS** Direction Operation_Set **UNTIL** Operation_Set
| **E** Direction Operation_Set **U** Operation_Set

Temporal_Logic_Operator :=

UNIVERSALLY | **AG** | **EVENTUALLY** | **AF**
| **STAY** | **EG** | **REACH** | **EF**

Direction :=

ε | **FROM**

Number :=

Floatingpoint_Constant | *Number_Variable*
| (Number) | Number Math_Operator Number

Assign_Type :=

MAX | **MIN** | **AVERAGE** | **RANGE**

4.3 Semantics of ASL operations

In the following, a brief description of the semantics of the main ASL operations is given.

value: With the operation `value`, a set of states can be selected on the whole state space or on another set by a state space parameter constrained to a specified interval. Algorithmically, for each state is decided whether it is included within the given interval by a simple comparison of the actual value of the state space parameter and the interval boundaries.

transition: Previous approaches to property specification of analog systems were directly derived from temporal logics and mostly perform some kind of reachability analysis. Although CTL-AT operations are still possible in ASL, the abstract reachability operation of ASL is called `transition` and selects states on paths between two state areas. It determines the minimum, maximum, average, and the range of the transition times detected on the transition paths. These values can be assigned to numeric variables using the `assign` command. The sum of edge weights on a path between two vertices i, j is defined as distance and is calculated by Dijkstra's algorithm either as shortest path or as longest path by inversion of the edge weights. A pseudo-code definition is given as follows.

```
ON base_set TRANSITION FROM start_set TO dest_set{
  for each vertex i in (start_set & base_set){
    for each vertex j in (dest_set & base_set){
      if (distance(i->j) < infinity){
        add distance(i->j) to transition_times;
        add vertices on path i->j to transition_set;}}}}
```

oscillation: The operation `oscillation` identifies states on cycles in state space and calculates the coresponding oscillation period. The pseudo-code definition is given as follows.

```
ON base_set SELECT OSCILLATION{
  for each pair (i,j) of vertices in base_set{
    if ((distance(i->j) < infinity) and
    (distance(j->i) < infinity)){
      add distance(i->j) + distance(j->i)
       to oscillation_times;
      add vertices on path i->j->i
       to oscillation_set;}}}
```

delta_compare: The operation `delta_compare` evaluates $\frac{\Delta value}{\Delta time}$ between all pairs of consecutive states of detected paths within a given transition area. Hence, it is possible to measure and verify the rate of change of a state space variable value over time on paths.

calculation: By defining a calculation formula, a set of states can be selected by evaluating an arithmetic property on the state space parameter values of each considered state. Additionally, the numerical calculation results can be used by following operations. Algorithmically, the formula is calculated on the parameter values of each state and therewith decided if it meets the given value constraint while recording the determined calculation results.

steadystates: The operation `steadystates` returns the steady states of the state space. For different input values, the corresponding DC operating points determine

the set of steady states of the system. This set is directly identified by the discretization process.

assign: The operation `assign` allows to assign a numerical value returned by another ASL operation to a number variable. The possible values are minimum, maximum, average, and range of the set of single values determined on a set of states.

assert: To complete the verification of a property, it is necessary to include assertions in the specification code. For sets, the operation `assert` checks whether a given set is the subset of the reference set. Numeric assertions check values determined during the verification process with respect to a given interval.

4.4 Application to a Ring Oscillator Circuit

The example circuit illustrated in Figure 5 is a modified ring oscillator with an even number of inverter stages and cross-coupling [11]. Due to the bridges β, this circuit will

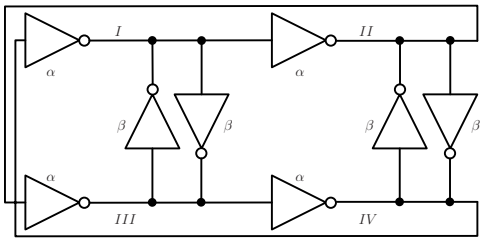


Fig. 5 Modified ring oscillator with an even number of inverter stages and cross-coupling.

oscillate if there is a ratio α/β of the transistor sizes in the feedback chain to those of the bridges within the interval [0.4, 2.0]. The crucial point of this circuit is its proneness to initial conditions, avoiding it to oscillate when the α/β ratio reaches or exceeds the interval boundaries. While several simulation runs for this critical transistor ratios with random initial conditions can show perfect oscillation, particular initial conditions exist which are preventing this circuit from oscillating.

A formal verification of the circuit using a property specification with the Analog Specification Language (ASL) and the ASL-MCT model checking tool [2] was performed. The ASL specification of the oscillation properties is given in Listing 1 and when applied to the circuit model, for transistor ratio 1.0 no initial conditions violating the oscillation behavior are existent. Figure 6(a) shows the vector field of the oscillation area automatically detected by the ASL verification algorithms described in section 4.3 with an α/β ratio of 1.0 projected to the state space variables V_I, V_{II}, and V_{III}. The initial state space model generation with 1970 states takes 54 seconds on a single core of Pentium 4 D with a clock frequency of 2.8 GHz and 2 GB of RAM. The ASL model checking algorithms run 3 seconds.

In contrast to the perfect oscillation with transistor ratio 1.0, Figure 6(b) shows a larger area of vectors belonging to possible oscillations for transistor ratio 2.1. Hence, for transistor ratio 2.1, initial conditions are detected for which the circuit will not run into an oscillation. This area of bad initial conditions is illustrated in Figure 7.

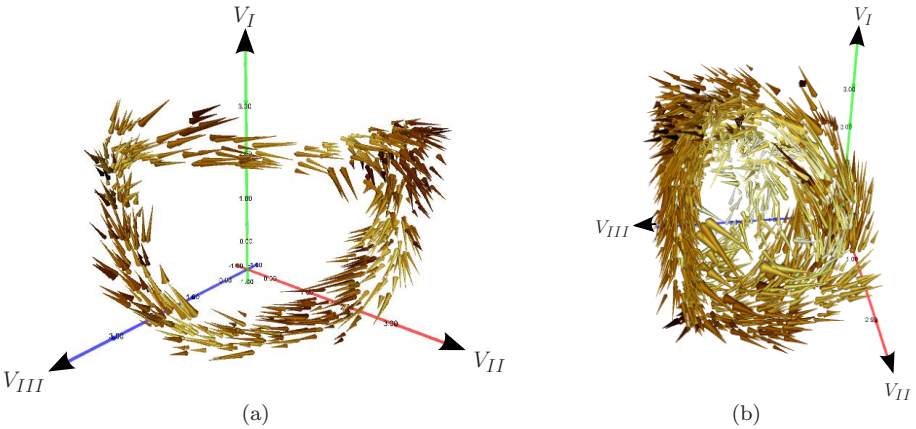


Fig. 6 Vector field representation of the oscillation area for transistor ratio $\alpha/\beta = 1.0$ (a) and for transistor ratio $\alpha/\beta = 2.1$ (b).

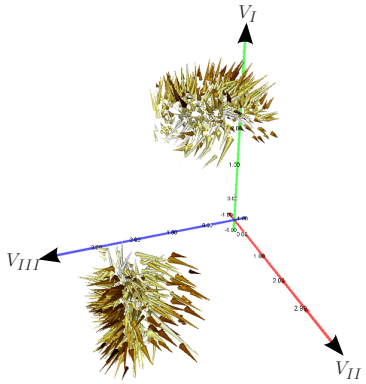


Fig. 7 Vector field of initial conditions leading into the non-oscillating steady states for transistor ratio $\alpha/\beta = 2.1$.

5 State Space Particle Simulation

By visualizing the vector field resulting from the state space sampling described in section 3, the dynamic behavior of the system under investigation can be analyzed. Vector field visualization is only possible with a restriction to 3 dimensions, while state space dimensions of common analog circuits can vary between 2 and more than 4. Therefore, a selection of the main dimensions has to be made to project to the 3-dimensional view.

Particle simulation is a common approach for vector field visualization and mature algorithms have been developed [12]. In contrast to a static approach such as line integral convolution, time-dependent motion of the particles visualizes the transient behavior of the circuit in an animation sequence.

```
# Get the oscillation set
osci_set = select oscillation;

# Check if oscillation exists (bool. result to %has_osci)
$set_is_not_empty(osci_set, %has_osci);

# Check if circuit has steady states (bool. result to %has_steady)
$set_is_not_empty(steadystates,%has_steady);

# Assertion: %has_osci shall be true
for %has_osci assert true;

# Assertion: %has_steady shall be false
for %has_steady assert false;

# Which states lead into steady states?
into_steady = reach steadystates;

# Which states can reach osci_set?
into_osci = reach osci_set;

# Which states run into steady states and not into the oscillation?
# These states are initial conditions that will never run into an
# oscillation, while states that are in the set into_steady and in
# into_osci can possibly run into steady states even if they can
# run into the oscillation area
bad_initial_conditions
  = into_steady and not into_osci;
```

Listing 1 ASL specification for oscillator verification.

We consider a vector field $V : \mathbb{R}^n \rightarrow \mathbb{R}^n$ on which for the discrete set $Q = \{q_1, ..., q_m\}$ of m sample points q_i in the state space, generated from equation 1, the discrete vector field $V_D : Q \rightarrow V_D$ is defined:

$$V_D(q_i) = \left\{ v_i | \exists q_i \in Q : v_i = \frac{\partial q_i}{\partial t} \right\} \tag{2}$$

In other words, the discrete vector field V_D is represented by position vectors q_i determining the sample points in the state space and the direction vectors $v_i = V_D(q_i)$ giving the motion direction and speed within V at position q_i.

For the injected particles $p_i \in \mathbb{R}^n$, their tangent vector $V(p_i) = \frac{\partial p_i}{\partial t}$ has to be approximated with respect to the discrete vector field V_D. Hence, a mapping is necessary which assigns a nearest sample point $q_j \in Q$ to each particle $p_i \in P$ from the set of particles P, as illustrated in Figure 8:

$$M(p_i) = \left\{ arg\ min_{q_j \in Q} ||p_i - q_j||_2 \right\} \tag{3}$$

Therewith, for each particle p_i, its next position can be calculated according to a time step Δt and the nearest direction vector $v_j = V_D(M(p_i))$:

$$p_i(t + \Delta t) = p_i(t) + v_j \cdot \Delta t \tag{4}$$

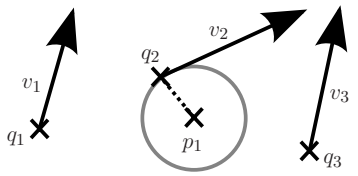


Fig. 8 Determining nearest sample point q_2 in state space for particle p_1 within discrete vector field V_D.

When starting the particle simulation, an equally distributed amount of particles is inserted into the vector field of the state space and for each particle, the nearest vector regarding euclidean distance determines its direction and speed of movement as stated above.

Algorithm 1 recapitulates the introduced particle simulation algorithm.

Algorithm 1: Particle Simulation Algorithm

```
while animation running do
    foreach each particle p_i in state space do
        detect nearest sample point q_j = M(p_i) with respect to euclidean distance;
        get direction vector v_j = V_D(q_j);
        p_i.position = p_i.position + v_j · Δt
    end
end
```

Each of the particles represents an independent simulation run with the starting position indicating its initial condition. While the visualization is projected to a 3-dimensional representation, the motion vector of the particles is calculated with full dimensionality. Thus, the motion is determined by all dimensions of the state space, revealing additional information exceeding the 3-dimensional plot.

For the tunnel diode oscillator circuit's state space with two state space dimensions, the particle animation is illustrated in four steps in Figures 9(a) to 9(d).

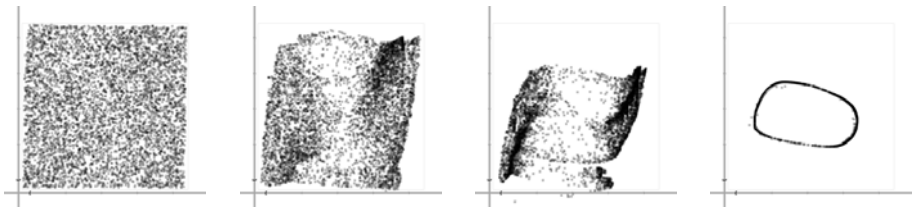


Fig. 9 Particle simulation for tunnel diode oscillator circuit with increasing time from (a) to (d).

For the ring oscillator circuit considered in section 4.4, a particle simulation has been performed. As expected from the model checking results, for transistor ratio

$\alpha/\beta = 1.0$, a perfect oscillation is detected as illustrated in Figure 10(a). In contrast, for transistor ratio $\alpha/\beta = 2.1$, the particle simulation clearly reveals the two aggregation areas where the system reaches a steady state and will not oscillate when initial conditions lead into this area as illustrated in Figure 10(b). These results match with those of the model checking approach. In terms of runtime, the state space vector field generation would take the same time as for model checking, but the model can be reused once it is generated, so no additional time for vector field sampling was consumed. The particle simulation for 5000 particles runs in real-time, allowing user-interactive zooming and rotation of the visualization during the simulation.

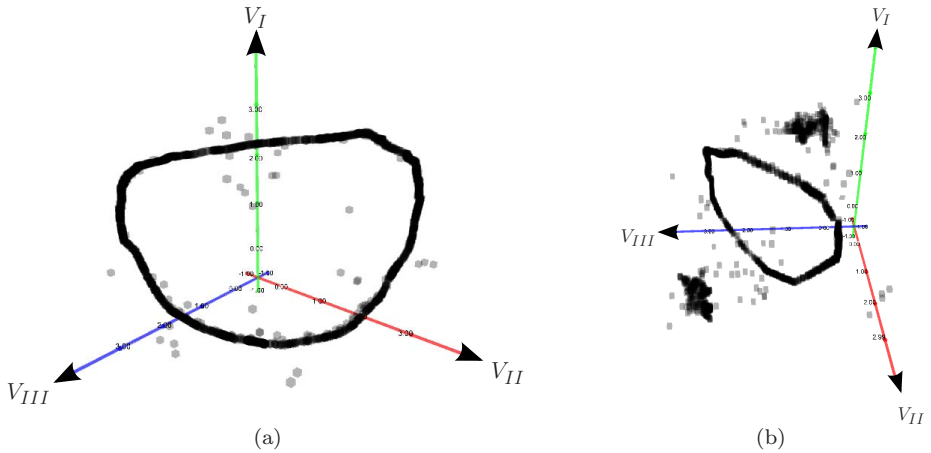


Fig. 10 Particle simulation of the oscillator circuit's dynamic behavior for transistor ratio $\alpha/\beta = 1.0$ (a) and for transistor ratio $\alpha/\beta = 2.1$ (b).

6 Conclusions

In this contribution two analog verification approaches covering the complete state space of the circuit under verification have been presented. A hidden design error of a ring oscillator was detected by model checking algorithms incorporating a property specification in the Analog Specification Language (ASL), as well as with a particle simulation covering the complete state space dynamics of the circuit under verification. In contrast to transient simulation, both approaches cover the complete state space of the circuit under verification and hence are not depending on manual selection of appropriate simulation test-benches for error identification. Future work will focus on the investigation of how to apply the proposed algorithms to more complex circuits.

7 Acknowledgments

The authors would like to thank Georg Denk and Carsten Hammer (Titan Simulator Development, Qimonda AG, Munich, Germany) and Peter Jores (AE/EIM3, Robert

Bosch GmbH, Reutlingen, Germany), and Victor Konrad for very fruitful discussions and their suggestions to this work.

References

1. W. Hartong, L. Hedrich, and E. Barke. Model checking algorithms for analog verification. In *Proceedings of the 39th conference on Design automation (DAC '02)*, pages 542–547, 2002.
2. S. Steinhorst and L. Hedrich. Model Checking of Analog Systems using an Analog Specification Language. In *Proc. of the Conference on Design, Automation and Test in Europe 2008 (DATE'08)*, pages 3247–329, 2008.
3. W. Hartong, R. Klausen, and L. Hedrich. Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking. *Advanced Formal Verification, R. Drechsler, ed., Kluwer Academic Publishers, Boston*, pages 205–245, 2004.
4. Darius Grabowski, Daniel Platte, Lars Hedrich, and Erich Barke. Time constrained verification of analog circuits using model-checking algorithms. *Electronic Notes in Theoretical Computer Science*, 153(3):37 – 52, 2006. Proceedings of the First Workshop on Formal Verification of Analog Circuits (FAC 2005).
5. E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Syst.*, 4(4):331–352, 1992.
6. D. Grabowski, D. Platte, L. Hedrich, and E. Barke. Time Constrained Verification of Analog Circuits using Model-Checking Algorithms. In *FAC 2005: Proceedings of the First Workshop on Formal Verification of Analog Circuits*, pages 37–52, 2005.
7. H. Foster, E. Marschner, and Y. Wolfsthal. IEEE 1850 PSL: The Next Generation. 2005. URL: `http://www.pslsugar.org/papers/ieee1850psl-the_next_generation.pdf`.
8. Dejan Nickovic and Oded Maler. Amt: A property-based monitoring tool for analog systems. In Jean-François Raskin and P. S. Thiagarajan, editors, *FORMATS*, volume 4763 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 2007.
9. S. Laemmermann, A. Jesser, R. Weiss, J. Ruf, L. Hedrich, T. Kropf, and W. Rosenstiel. An Assertion-Based Verification Methodology for SystemC-AMS Designs. In *The 15th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI'09)*, pages 434–439, Okinawa, Japan, March 2009.
10. Martin Freibothe, Jens Schönherr, and Bernd Straube. Formal verification of the quasi-static behavior of mixed-signal circuits by property checking. *Electr. Notes Theor. Comput. Sci.*, 153(3):23–35, 2006.
11. Kevin D. Jones, Jaeha Kim, and Victor Konrad. Some 'real world' problems in the analog and mixed signal domains. In Gordon J. Pace and Satnam Singh, editors, *Seventh International Workshop on Designing Correct Circuits: Budapest, 29–30 March 2008: Participants' Proceedings*, pages 15–29. ETAPS 2008, March 2008. A Satellite Event of the ETAPS 2008 group of conferences.
12. D. L. Darmofal and R. Haimes. An analysis of 3d particle path integration algorithms. *J. Comput. Phys.*, 123(1):182–195, 1996.