Embedded Tutorial: Analog-/Mixed-Signal Verification Methods for AMS Coverage Analysis

Erich Barke[4], Andreas Fürtig[2], Georg Gläser[3], Christoph Grimm[5], Lars Hedrich[2], Stefan Heinen[6], Eckhard Hennig[8], Hyun-Sek Lukas Lee[4], Wolfgang Nebel[1,7], Gregor Nitsche[1], Markus Olbrich[4], Carna Radojicic[5], and Fabian Speicher[6]

[1]OFFIS, Institute for Information Technology, Germany
[2]Electronic Design Methodology, Dept. of Computer Science, Goethe-Universität Frankfurt a. M., Germany
[3]IMMS Institut für Mikroelektronik- und Mechatronik-Systeme gemeinnützige GmbH Ilmenau, Germany
[4]Institute of Microelectronic Systems, Leibniz Universität Hannover, Germany
[5]Design of Cyber-Physical Systems, Kaiserslautern University of Technology, Germany
[6]Chair of Integrated Analog Circuits, RWTH Aachen University, Germany
[7]Carl von Ossietzky University Oldenburg, Germany
[8]Reutlingen University, Germany

***Abstract*—Analog-/Mixed-Signal (AMS) design verification is one of the most challenging and time consuming tasks of todays complex system on chip (SoC) designs. In contrast to digital system design, AMS designers have to deal with a continuous state space of conservative quantities, highly nonlinear relationships, non-functional influences, etc. enlarging the number of possibly critical scenarios to infinity. In this special session we demonstrate the verification of functional properties using simulative and formal methods. We combine different approaches including automated abstraction and refinement of mixed-level models, state-space discretization as well as affine arithmetic. To reach sufficient verification coverage with reasonable time and effort, we use enhanced simulation schemes to avoid conventional simulation drawbacks.**

1. Introduction

Facing the complexity of modern SoCs, AMS design verification is a prerequisite for a modern design flow to prove the system's functional behaviour and avoid design bugs. Hence, to optimize time to market while ensuring safety and quality of the design, measuring the verification quality became crucial in deciding whether the regarded system is sufficiently tested or verified.

Especially in the area of safety-critical design e.g. automotive hardware and software applications coverage metrics are commonly used to evaluate the amount of the already invested verification effort. This is done by comparing the number of analyzed verification or test scenarios with the overall number of scenarios. Due to the finite and discrete nature of digital systems the overall number can either be obtained from the model of the design (Structural Coverage) or from its specification (Functional Coverage). Opposed to this, analog circuit designers are challenged by continuous quantities and physical aspects of single blocks or devices. In addition to functional properties, non-function effects like crosstalk over supply or parasitic coupling have to be investigated in industrial size designs. Moreover, several levels of abstraction have to be considered demanding methods for system level as well as transistor level circuits. Digital domain coverage metrics are not directly applicable to AMS circuits and systems. Hence, industrial use-cases demand for novel coverage-oriented modeling and verification strategies to be investigated to tackle this challenge making the quality of AMS verification measurable. Within the special session we present methods and concepts to improve the AMS verification process and to allow for the evaluation of the coverage proposing different metrics.

In the next chapter, we present an approach to integrate uncertainties into system-level analog simulations using affine arithmetic. To consider formerly not modelled properties in available models, we propose a refinement and efficient simulation scheme in the following section. The last part of this contribution deals with formal checking of AMS models based on state-space exploration and contracts.

2. Towards More Dependable Verification Using Symbolic Simulation

A particular challenge in analog/mixed-signal verification is a high sensitivity of analog parts to deviations in its parameters and quantities. These deviations can be caused by variations in a system environment or the design process itself. They are of huge impact on system verification since they can lead to violation of properties defined by system specification. However, there is still a lack of methods,

which can provide dependable verification results in the presence of these deviations.

Numerical simulation requires a high number of runs to analyze a system behavior over wide sets of parameter values. Even then, the dependability still cannot be guaranteed. On the other side, formal methods allow a comprehensive verification. However, the scalability of these methods with complexity and heterogeneity is a big question. This part of the contribution will give an overview on symbolic simulation. A literature survey provides a list of methods, which uses a symbolic approach for mixed-signal verification. However, all these methods require the use of their own tools and translation of mixed-signal designs to appropriate formal models.

In contrast to these methods, this tutorial will present a methodology, which is much closer to current design practice. The presented symbolic approach is integrated in the existing simulator where scalar values are replaced by symbolic ranges. In this way, comprehensive verification of formal methods is combined with the general applicability of simulation. Furthermore, using symbolic forms the proposed methodology allows an easy tracking of the impact of each deviation on the overall system behavior.

The proposed symbolic simulation verifies system behavior already on a high level of abstraction. A mixed-signal system is modeled as a block diagram used in common modeling languages such as Matlab/Simulink or SystemC-AMS, Verilog, VHDL-AMS, etc..

Here, we use SystemC-AMS. Analog parts are modeled by the SystemC-AMS extensions. Software is modeled by embedding C++ code in SystemC modules.

For symbolic simulation we replace the basic C++ data types double/int/bool by the new abstract data type XAAF. The objective is to allow us the use of an existing, numeric simulator for symbolic simulation. This permits an easy integration of the methodology in the existing design flow. This is possible through the steps summarized in Figure 1. The proposed methodology will be demonstrated

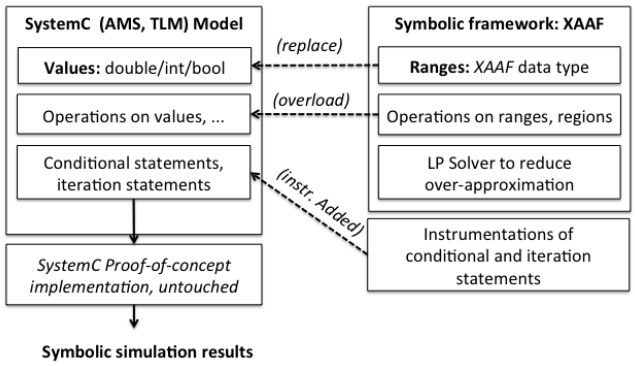


Figure 1: Overall idea of XAAF-based symbolic simulation.

for a dual-path charge-pump PLL circuit, which is used to generate a high quality signal for the local oscillators in a IEEE 802.15.4 transceiver system. The virtual prototype of the circuit in SystemC is provided by our project partner RWTH University in Aachen.

2.1. Extended Affine Arithmetic XAAF

Affine Arithmetic (AA) [1] is a mathematical approach introduced to overcome the divergence problem of Interval Arithmetic [2]. The use of AA for symbolic simulation of analog systems was firstly proposed by Grimm in 2004 [3]. A system behaviour is simulated over the ranges of parameters represented in a symbolic way:

$$\tilde{x} = x_0 + \sum_{i=1}^{n} x_i \varepsilon_i \qquad \varepsilon_i \in [-1,1] \tag{1}$$

Affine Arithmetic with the above form allows only computations with ranges in the continuous domain. However, this is not enough for simulation of mixed-signal systems which beside analog parts also contain digital parts and embedded software. In order to allow computation with ranges on the digital and software side, [4] proposes the extension of Affine Arithmetic.

An extended Affine Arithmetic form $\hat{x}$ is defined by:

$$\begin{aligned} \hat{x} &= \underbrace{x_0 + \sum_{i=1}^{m} x_i \varepsilon_i}_{\tilde{x_0}} + \sum_{k=1}^{n} \omega_k \underbrace{\left(x_{0k} + \sum_{i=1}^{m} x_{ik} \varepsilon_i \right)}_{\tilde{x_k}} \\ &= \tilde{x_0} + \sum_{k=1}^{n} \omega_k \tilde{x_k} \quad \omega_k \in \{-1,1\}. \end{aligned} \tag{2}$$

where $\tilde{x_0}$ represents the center around which ω_k split ranges/polytopes represented by AAF. Analogous to AAF where ε symbols represent ranges, ω symbols are used to represent the sets of possible ranges/regions.

Computation with XAAFs. Computation with XAAFs is allowed through overloading standard C++ arithmetic and relational operators. Since the extended affine form represents the set of affine forms, all arithmetic operations defined on affine forms [5] hold here.

Relational operators ($>, <, \leq, \geq, ==, !=$) are overloaded to allow comparison of ranges on the digital and software side. Comparison with range-based quantities can result in $false$, $true$ but also in $false$ and $true$, if the compared ranges overlap.

Instrumentation of Control Flow Statements. The overloaded relational operators allow computation of conditions in control flow statements whose conditional variables are ranges. If a condition value is unknown (`{false, true}`), both possible branches of a control flow should be executed. This is not possible with the 'default' semantics of conditional statements. In the following we show how this issue can be solved by applying the following small instrumentation rules on existing codes:

Rule 1 - Condition Rule This rule explains how the condition part of C/C++ codes should be modified to handle conditional variables as ranges. Using the XAAF approach

the result of a condition is not Boolean type (`false` or `true`), but XAAF type which can also result in both `{false, true}`. To check if the condition `cond` may result in `{false, true}`, it should be checked if the condition value is different than `false`. To do so the condition part in regular C/C++ code should be modified as following:

cond becomes cond$!=$**false**.

Rule 2 - Statement Rule The second rule explains how the actions for the `true` condition value should be modified to cover the case for which the condition results in `{false, true}`. Let the statement values for the `true` and `false` condition values be `stm1` and `stm2`, respectively. Hence, for `cond=true` the statement `stm` gets value `stm=stm1`. However, applying Rule 1 the statement will execute when `cond=true` but also when `cond={false, true}`. To cover the second case, the statement should be modified using Shannon's expansion:

stm $=$ **stm1 becomes stm** $=$ **cond** $*$ **stm1**$+!$**cond** $*$ **stm2**

where the first part computes the statement for `cond=true` and the second for `cond=false`. Note, that for `cond=true` the statement gets only `stm1` value (`!cond*stm2=0`), as it should be.

In the XAAF library the set `{false, true}` is represented by XAAF:

$$0.5+\omega 0.5; \omega \in \{-1,1\}$$

where $\omega=-1$ represents the `false` and $\omega=1$ the `true` condition value. Due to its symmetry, the implementation of negation operator $0.5+\omega 0.5$ only required to change the sign of the value which scales ω. Thus, the negation operator results in:

$$!(0.5+\omega 0.5) = 0.5-\omega 0.5; \omega \in \{-1,1\}.$$

This explains why we chose ω to be $\{-1,1\}$ instead of $\{0,1\}$.

Example: There are various variants of control flow statements: conditions such as (`if`, `if-else` and `if-else-if`) and iterations such as (`do while`, `repeat until`). Here we discuss the `if-else-if` conditional statement. The others can be modified in a similar way. Fig. 2 shows the modification of `if-else-if` applying Rules 1 and 2. Each branch should be executed only if the corresponding conditions are `true`. Therefore, we need to check if the conditions are not equal to `false` as defined by Rule 1. The statements should update their value only if the corresponding conditions are `true`. For the `false` value of the conditions the statement value should stay the same. Applying Rule 2 the condition value is then multiplied with the new statement value and its negation with the old value:

$$stm = cond1*(stm1)+!cond1*(stm)$$

The same holds also for the rest of the branches. Here, it should also be noted that in the case the conditions result in `true` the conditional statement is the same as on the left side.

```
if (cond1)                  if (cond1!=false)
{                           {
  stm=stm1;                    stm=cond1*stm1+!cond1*stm;
}                           }
else if (cond2)             else if (cond2!=false)
{                           {
  stm=stm2;                    stm=cond2*stm2+!cond2*stm;
}                           }
```

Figure 2: IF-ELSE-IF conditional statement with Boolean conditions (left) and XAAF conditions (right).

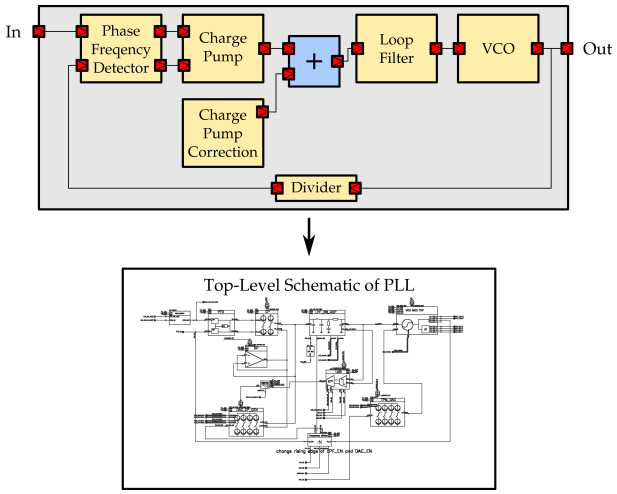


Figure 3: The structure of PLL circuit.

2.2. Simulation Results of Transceiver PLL Circuit

The method is applied on a PLL circuit of one IEEE 802.15.4 RF Transceiver. The structure of the circuit is shown in Fig. 3. The SystemC virtual prototype of the circuit was provided by our ANCONA project partner RWTH University of Aachen.

The PLL design was simulated over the range of initial node voltages of the loop filter (low-pass filter) v_0, v_1, $v_2 \in [0.5, 0.7]$. The current variations of charge pump and current generators in the correction block (see Fig. 3) inside 10 % tolerance were also take into account. The circuit was simulated for 100.000 time steps each equal to $1/f_s$, where f_s is the sampling frequency equal to 20 GHz. The reference frequency was 32 MHz and the N ratio was 110. Hence, the desired output frequency should have been close to the value 3.52 GHz.

Fig. 4 shows the output frequency calculated for the first 400 time steps. Black lines show the result of 20 behavior simulations computed for randomly values of node voltages and current variations. Red lines show the worst case behavior found by symbolic simulation. Note that the red envelope encloses the behavior of random simulations. The PLL design over the considered range of operating

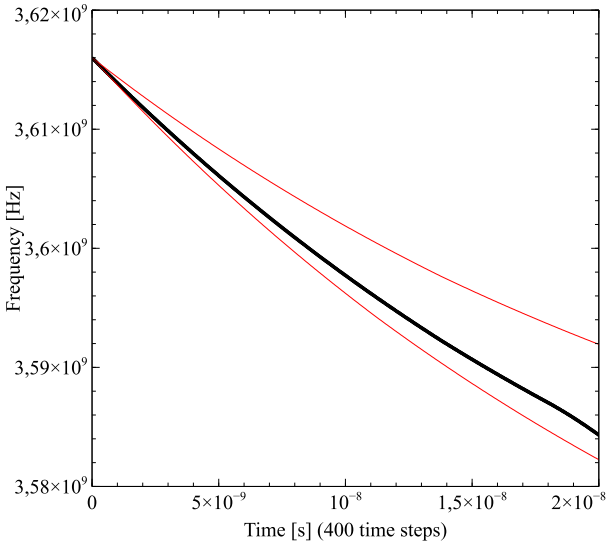


Figure 4: PLL output frequency for the first 400 time steps.

conditions locked to 3.52 GHz with the tolerance of 0.1 % after 36.000 time steps (corresponding to time 2 µs). The circuit was simulated for additional 64.000 time steps and the circuit stayed in the locked state. The total simulation took 15 minutes. The number of possible transitions was around 1.000 which were covered with 10 ω symbols. On the other side, one random simulation run took only 2 s. However, our approach is still competitive considering the fact that thousands of random simulations are required to obtain a sufficient coverage of a state space.

3. Identification of Critical Scenarios in AMS Verification

The enormously increasing complexity of AMS systems over the last years requires new design automation efforts in several areas. One of the main challenges is to perform the functional verification with respect to non-functional properties, e.g. timing and power integrity. These effects potentially decrease the system's accuracy or even disturb the overall functionality. Considering these effects makes system analysis more and more complex. Hence, a systematic, coverage-based approach is needed to measure the quality of the executed verification. This work presents a novel approach for identifying critical scenarios in AMS systems. We combine automatic annotation of non-functional properties to behavioral models with accelerated piecewise-linear analog simulations. On the basis of our abstract modeling approach the simulation avoids time consuming numerical integration, which speeds up system simulation. With this method, the evaluation of more different scenarios is possible during the same amount of time. This allows to identify the uncertain limits of system's functionality more dependably. Our framework supports the design of AMS systems by showing the critical behavior and creating verification scenarios to be verified throughout the design process. These new use cases provide a base for defining a coverage-metric to evaluate and increase the overall verification quality.

3.1. Finding System Acceptance Regions of AMS Systems Influenced by Non-Functional Properties

Virtual prototyping of AMS systems for verification became one of the major concerns in designing modern ASICs. Non-functional effects such as noise and cross-talk are often critical to the system's performance [6]. Therefore, their integration into the virtual prototype model is crucial for the design and verification process. In this section, we propose a methodology for determining system acceptance regions considering non-functional effects.

To demonstrate the methodology, we examine a hysteretic current-mode buck converter circuit for driving OLEDs shown in Figure 5 [7]. In this circuit, the output current is controlled using two reference voltages and a shuntresistor in order to sense the current flowing through the attached load. This system can be influenced by several non-functional effects possibly destabilizing the circuit, e.g. reference voltages could be distorted by noise or crosstalk aggression from supply or ground.

For verifying the stability, a SystemC-AMS [8] prototype is used. The purely analog components of the systems are modeled using a piecewise linear (PWL) scheme described in Section 3.2. To integrate the aforementioned non-functional effects in this prototype, the components outside of the PWL simulation are automatically analyzed and wrapped with additional blocks modeling the required properties. The annotation and refinement process is shown in Fig. 6. Since the effort of modeling every effect manually is significant, the model components are refined automatically. With the help of the *libclang* code analysis framework [9], the structure and the schematic behavior of the circuit can be extracted. The actual refinement is done by using this information to generate a description of the non-functional effect. This process is used to generate a specific wrapper modeling the impact on the behavior of the component. For some effects, it is necessary to connect this refined component to additional signals modeling e.g. cross-talk issues. This is so done by automatically adding new signals and connections to the source-code and replacing the model instance by a refined one in the hierarchy.

Based on these refined models, a parametric simulation is executed to identify corner cases and parameter regions with correct system behavior. These system acceptance regions provide a basis for design decisions as well as scenarios determining the critical non-functional property model parameters of a *device under verification*.

3.2. Accelerated Mixed-Level Mixed-Signal Simulator

An accelerated simulation of AMS circuits based on piecewise-linear models has been presented in a previous work [10]. Our simulation environment focuses on analog

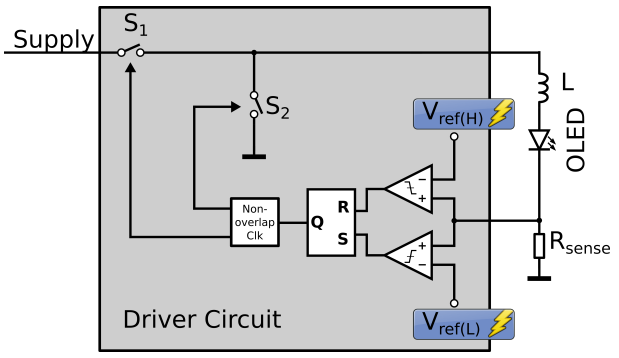


Figure 5: Hysteretic current-mode buck converter for OLED control.

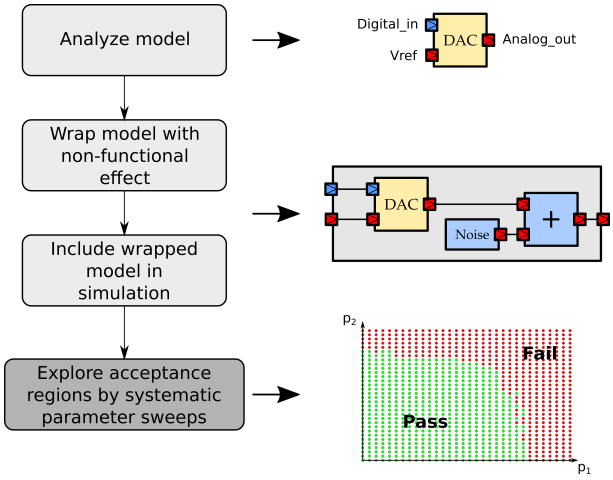
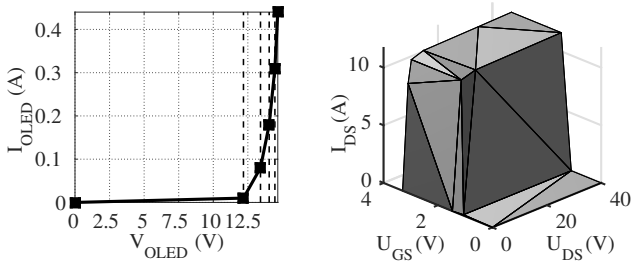


Figure 6: Annotation and simulation process for exploration of system acceptance regions.

subcircuits as shown in Figure 5. It provides a simulation kernel for an accelerated simulation of the analog part. Speedup is achieved by avoiding numerical integration and directly using the linear time-domain solution of the system. The time-domain solution is described by sums of exponential terms of the form (3), which can be efficiently evaluated during simulation.

$$\mathbf{y}(\mathbf{u},t) = \mathbf{y_0} + \sum_{i=1}^{n} \mathbf{a}_i(\mathbf{u}) \cdot \mathrm{e}^{\lambda_i t} \tag{3}$$

This approach requires piecewise-constant inputs (implicitly given by the digital part of AMS circuits) and linear or at least piecewise-linear models [11]. The models are generated by taking advantage of geometric methods (see Figure 7). The use of PWL device models results in the generation of multiple linear state-space circuit models describing the analog circuit behavior. The possible combinations of these piecewise-linear models result in a hybrid automaton [12]. It is natural that exactly one state of the automaton is valid at the same time. The modeling approach can be applied to nonlinear semiconductor devices as well as to nonlinear macro models like operational amplifiers, which



(a) OLED model (5 sections). (b) N-channel MOSFET switch (15 sections/triangles).

Figure 7: Abstract piecewise-linear behavioral models of OLED and low-side switch S_2.

makes a mixed-level simulation feasible. Approximating a circuit by a hybrid system with linear continuous dynamics has been used before, see e.g. [13], [14], [15]. It has been proven to be an applicable method to control the complexity of system-level modeling. Each state corresponds to a discrete state-space representation of the form (4) and (5).

$$\dot{\mathbf{x}}(t) = \mathbf{A}_v\mathbf{x}(t) + \mathbf{B}_v\mathbf{u}(t) \tag{4}$$

$$\mathbf{y}(t) = \mathbf{C}_v\mathbf{x}(t) + \mathbf{D}_v\mathbf{u}(t). \tag{5}$$

Switching Between State-Space Models. Switching between different PWL models is an important step for our simulation methodology. The switch over time depends on the threshold voltages and currents of nonlinear components. Several root-finding algorithms for such functions are known. We found that the Newton-Raphson method and the bisection method yield unsatisfactory results, as they often do not converge towards the first root and exhibit long runtimes. A specialized root-finding algorithm for this task has been presented in [16]. It guarantees to find the first root in a given interval.

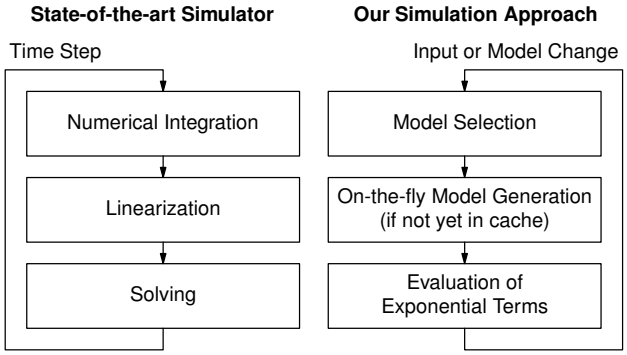


Figure 8: State-of-the-art and our analog simulation flow.

Figure 8 shows a comparison of our simulation approach with existing analog circuit simulators. Instead of performing numerical integration, linearization and solving the system of equations during each time step, our approach is only sensitive to input changes and internal model switching. A

model switch is triggered by a transition from one linear section of a PWL component to another. The active linear section, selected by the simulator kernel, remains valid as long as no change of any input occurs and the circuit does not exceed the current section due to its dynamics. In case of an input change, a new valid section must be calculated along with its new initial values to satisfy the continuity of inductor currents and capacitor voltages. In contrast to an input change, the dynamics of the circuit make switching to an adjacent linear section necessary.

SystemC Interface. To analyze an AMS system, described in Section 3.1, the accelerated simulation kernel is interfaced to SystemC. The interface enables mixed-system simulations, where SystemC serves as master simulator and as digital simulator. The interface consists of the modules shown in Figure 9. As the structure of these modules depends on

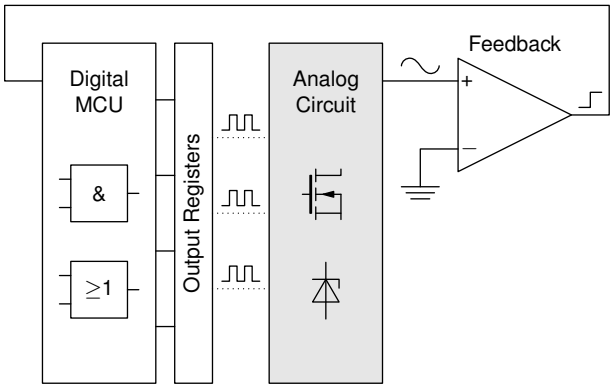


Figure 9: Mixed-signal circuit setup.

the components of the AMS system, we generate SystemC wrapper modules automatically. For this purpose only a description of the analog components and an XML file specifying input and output ports of the analog part are required. The SystemC wrapper modules encapsulate the accelerated simulation kernel as well as the hybrid automaton describing the analog circuit behavior. The module offers a simulation method which is invoked when input signals change or a model transition is detected via the event feedback loop. Thus, an event-driven simulation is feasible. Our previous work shows, that a simulation speedup of factor 30 is achievable [11]. The latter was compared to a reference simulation setup using ModelSim and Saber.

3.3. Experimental Results

For experimentally evaluating our approach, the OLED driving system shown in Figure 5 is used. In this system, we annotate several non-functional properties to the analog reference voltages $V_{\mathrm{ref(L)}}$ and $V_{\mathrm{ref(H)}}$.

In this contribution, we take the designer's point of view to find out acceptance region border values of the annotated effects to be taken into account during the following schematic design process. In a first step, abstract models for the components are created. Since the buck converter itself relies on conservative quantities, it has to be modeled in an analog way using the described accelerated PWL mixed-signal simulation scheme. Besides this analog model, a checker module observing the simulation classifies correct and incorrect behavior.

Before the actual simulation, the non-functional effects are annotated automatically to the model components. In order to cover the resulting non-functional parameter space, the parameters swept over a predefined range. In the obtained simulation results the correct and incorrect simulation points are extracted to explore the influence of the modifications and their interactions on the overall system behavior. As an example result, Figure 10 shows the impact of noise on $V_{\mathrm{ref(L)}}$ and $V_{\mathrm{ref(H)}}$. We observe, that both effects cannot be treated separately since the figure shows a clear interaction. Based on this method, the worst-case noise variance values can be determined indicating an additional check for system verification. A coverage metric describing the amount of modeled effects could be defined using these results. Moreover, advanced algorithms can be used to track the system's critical scenarios – i.e. the boundary between passing and failing simulations – even more accurately by adaptively adding additional points. In future work, we aim at reducing the number of required simulations for identifying the acceptance regions of a system in a multidimensional parameter space.

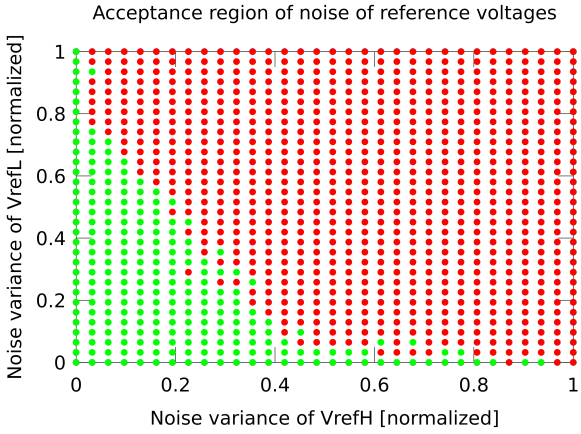


Figure 10: Acceptance region for noise impact on $V_{\mathrm{ref(L)}}$ and $V_{\mathrm{ref(H)}}$. Passed simulations are marked green, failed red.

4. AMS Leaf-Component Characterization and Satisfaction Checking vs. Electronic Circuit Schematics

The formal verification of AMS systems remains still a demanding task and a main part of complexity comes from the need to analyze and verify extra-functional properties, such as timing or power consumption and from preventing non-functional design faults, such as power cross-talk. To check the correct implementation of such designs, sophisticated designers and verification engineers are necessary.

To reduce the design flaws, we show an approach to ensure extra-functional properties of AMS systems, using contract-based design for refining the system to subsystems. Finally we verify the contracts for the implementation of the refined subsystems. Based on a discretization of their analog state space, the implementation of the much smaller subsystems can formally be checked for satisfying the contracts or containing design flaws. Furthermore, the previously defined contracts, can be checked by a systematic simulation, which is controlled by a state-space based coverage analysis to visit all states of the state space. If necessary, from that, a bottom-up adaption of the leaf components' contracts becomes possible, which then can be used for the virtual integration. As a result, a more reliable verification of the system level contracts becomes possible.

4.1. Contract Based Specification and Verification

For digital systems the formal description and refinement of functional specifications using contract and component based design (CBD) is a well-developed approach [17], [18]. Differently, for extra-functional and AMS properties the compositionality issues prohibit its direct application without further constraints [19], [20]. To continue our work in [19], [20] we now implemented the bottom-up characterization of extra-functional or AMS properties using the components' transistor level SPICE netlists. Based on the resulting discrete analog transition systems (DATS) the satisfaction of the contracts can be verified for the leaf components – i.e. components of the lowest level of refinement. In the result, we connect the contract based process of virtual integration and refinement checking to the detailed electrical design and characterization of the leaf components.

Contract and Component Based Design (CBD). Following the top-down CBD approach we consider the system M as well as the components $M_i \in M_M^*$ of its decomposition $D_M = (M_M^*, N_M)$ into subcomponents M_i and interconnections $n_i \in N_M$ as components M resp. M_i [1] , defined by:

$M, M_i = (tp(M), \chi_M, S_M, D_M, B_M)$ with:

- $\mathrm{tp}(M)$ is the component's *type* name
- $\chi_M = X_M \cup Y_M$ is the component's port *interface*
- $S_M = \bigcup_{i\in\mathbb{N}} C_i$ is the component's set of *contracts* C_i
- $D_M = (M_M^*, N_M)$ is the component's *decomposition*
- B_M is the component's *implementation* as a DATS

Assuming the behavior and the communication of the components to be compositional for some constraints, the component denotes a design element which internally encapsulates its behavior, restricting its interaction with its environment to solely its well-defined, directed port interface $\chi_M = X_M \cup Y_M$, consisting of *input ports* $x_i \in X_M$ and *output ports* $y_i \in Y_M$. Based on this interface, the *contracts* $C_{M,i} := (A_{M,i}, G_{M,i})$ of a component M declare *guarantees* $G_{M,i}$ which are promised to hold at the output ports, as the component's environment satisfies the corresponding *assumptions* $A_{M,i}$ of $C_{M,i}$ at the input ports.

Specifying assumptions and guarantees in the extended *linear time logic OTHELLO* (Object Temporal with Hybrid Expressions Linear-Time LOgic) [21], assumptions A and guarantees G can intuitively be interpreted as the sets $[\![A]\!] = \bigcup_i S_i(A)$, $[\![G]\!] = \bigcup_i S_i(G)$ of *hybrid trace* sets $S_i(A) = \bigcup_{j\in|X|} s(x_j)$ resp. $S_i(G) = \bigcup_{j\in|Y|} s(y_j)$ which for all their *hybrid traces* $s(x_j) = \{(v_0(x_j), T_0), \ldots, (v_k(x_j), T_k)\}$ resp. $s(y_j) = \{(v_0(y_j), T_0), \ldots, (v_k(y_j), T_k)\}$, with $T_0 = 0$, $k \leq \infty$, $T_i \in \{[t_k, t_k] = t_k, (t_k, t_{k+1})\}$, hold the assumption resp. guarantee for all of the traces timed values $(v_k(x_j), T_k)$ resp. $(v_k(y_j), T_k)$ at any time $t_i \in \mathbb{R}$.

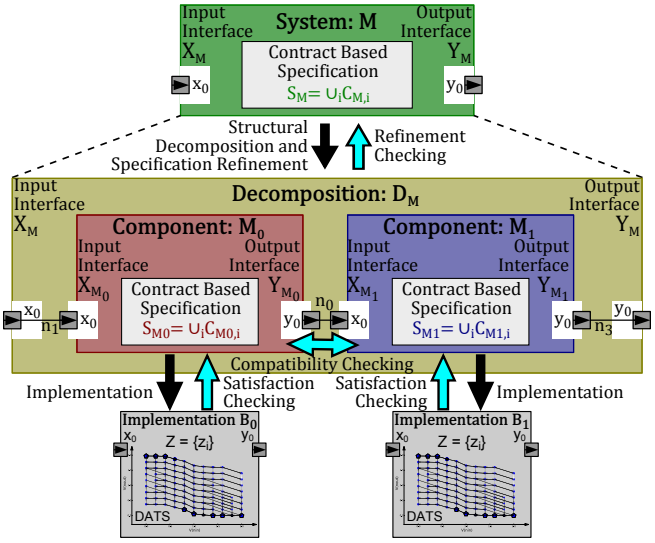


Figure 11: General concepts of contract based design.

Refining the specification S_M of a component M by a decomposition structure D_M and specifying the subcomponents $M_i \in M_M^*$ by specifications S_{M_i}, CBD allows to derive and verify the composed system behavior S_M^* of the subcomponents' *virtual integration* by systematically integrating the contracts $C_{M_i} \in S_{M_i}$ of all subcomponents $M_i \in M_M^*$ and formally comparing the resulting S_M^* with the contracts of the system specification S_M. Figure 11 outlines this processes of system *decomposition* and specification *refinement* as well as checking the *virtual integration* by *refinement checking*, *compatibility checking* and *satisfaction checking*. While refinement checking verifies if the virtual integration S_{M^*} holds the initial specification S_M, compatibility checking verifies, if for all interconnections $n_i = (p_{src}, p_{snk})$, $p_{src} \in X_M \cup \bigcup_{M_i\in M_M^*} Y_{M_i}$, $p_{snk} \in Y_M \cup \bigcup_{M_i\in M_M^*} X_{M_i}$, $n_i \in N_M$ the traces guaranteed for the net's *source* port p_{src} are acceptable traces for the assumptions concerning the *sink* port p_{snk}. To specify the components' interfaces and contracts as well as to specify and to verify the component compatibility and refinement relations, we use the *OCRA System Specification* language *OSS* of *OCRA* (OTHELLO Contract Refinement Analysis) [22]. Finally, for the leaf components of the

1. Not meaning identity, we repeatedly use i for multiple indices $i \in \mathbb{N}$. To provide detailed relations between indices, we additionally use j and k and explicitly formulate their relations.

refinement the satisfaction checking denotes the verification, if all execution paths of the component's behavioral implementation B_M satisfies the contract based specification S_M, meaning that for all execution paths the appropriate guarantees $G_{M,i}$ hold, if the corresponding assumptions $A_{M,i}$ are satisfied by the environment.

4.2. Extra-Functional and AMS Contracts

To apply contract based design for extra-functional and AMS properties, we use *virtual ports* for the node voltages (V) and branch currents (I) at the electrical ports. However, owed to their conservative nature, the AMS and extra-functional properties like timing or power consumption are not generally compositional for these ports. As a consequence, these properties can be guaranteed only with tolerance margins and with additional constraints for their environment. Hence, to apply contract based design we suggest to extend the components and contracts for specifying constraints for the wiring szenario, meaning to provide guarantees concerning its input impedance as well as assumptions for its acceptable load impedances. To relate these wiring information with the schematic and to verify their compatibility based on contracts, we use structural contracts [20]. With structural contracts we explicitly define the interconnection rules for the components' virtual ports according to a given tailoring of the component models.

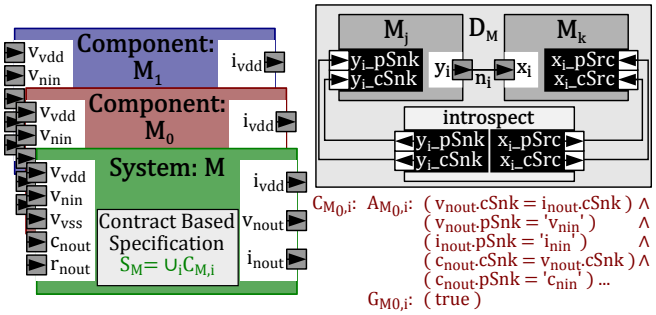


Figure 12: Outline of structural contracts to add constraints for the interconnection of virtual ports.

Application example. Figure 12 outlines a simple example, drafting a system interface M and the interfaces M_j, M_k of the refining subcomponents. For each of their ports $x_i \in X_{M_i}$, $y_i \in Y_{M_i}$ two structural ports $x_i.cSrc$, $x_i.pSrc$ resp. $y_i.cSnk$, $y_i.pSnk$ are inserted as depicted on th upper left. Connecting these ports to an additional introspection component, the information of exemplary interconnection $n_i = (M_j.y_i, M_k.x_i)$ is reflected to the components, and their structural contracts – one drafted at bottom right – can be verified.

As a simple example, we specify contracts for the functional, extra-functional and AMS properties of a small PWM output driver circuit according to Fig. 13. The actual implementation is illustrated in Figure 14 which consists of two inverter circuits with different parameters to meet the different contracts mentioned before.

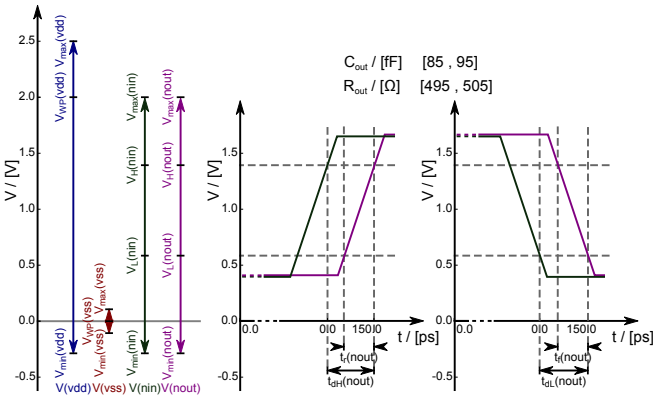


Figure 13: Exemplary specifications for a simple PWM output driver circuit.

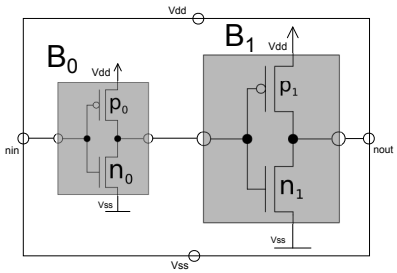


Figure 14: Implementation of the Decomposition D_M, using two inverter circuits with different parameters (see Tab. 1).

For satisfaction checking, we characterize the component based on its transistor level design and discretize its electrical behavior into a *discrete analog transition system*.

4.3. Satisfaction Checking: State-Space Coverage

To reduce the complexity of the full continuous state space of an analog circuit, we create a discrete model of the actual implementation. Starting with a netlist description of a subsystem, the underlying DAE (Differential Algebraic Equation) system of the circuit is solved. The resulting high dimensional state space is discretized using an electrical circuit simulator with full SPICE accuracy [23]. The full process is well described in [24].

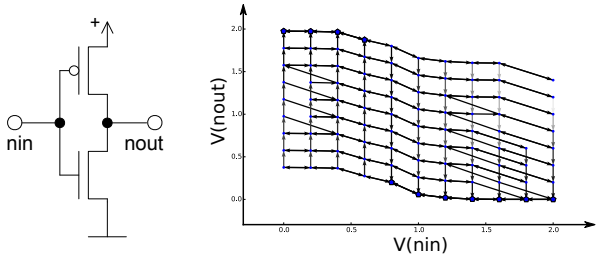


Figure 15: Simple inverter circuit (left) and the resulting DATS System (right).

With these method we can construct a discrete state-space model *DATS*:

Definition 1. Discrete Analog Transition System (DATS) *For the DATS we define a four-tuple* $M_{ATS} = (\Sigma, R, L_V, T)$ *where*

- Σ *is a finite set of states of the system.*
- $R \subseteq \Sigma \times \Sigma$ *is a total transition relation, hence for every state* $\sigma \in \Sigma$ *there exists a state* σ' *such that* $(\sigma, \sigma') \in R$.
- $L_V : \Sigma \to \mathbb{R}^{n_d}$ *is a labeling function that labels each state with the vector of* n_d *variables containing the values of the state space variables and the inputs of the DAE system.*
- $T : R \to \mathbb{R}_0^+$ *is a labeling function that labels each transition from* σ *to* σ' *with a real valued positive or zero transition time that represents the time required for the trajectory in the state space between these states.*

Figure 15 shows the results of the discretization of a simple inverter circuit.

Once the DATS is calculated model-checking techniques can be applied to the discrete model of the analog system. Using the *Analog Specification Language* (ASL) [25] simple property specifications can be verified. Additionally, a reachability analysis reduces the number of states inside the DATS, resulting in the set of states Σ_R. As seen in Def. 1, timing information T are available on each edge and information about the states are stored inside the DATS during the discretization process. Besides all node voltages also currents in every state are stored as well, allowing us to define and verify power properties.

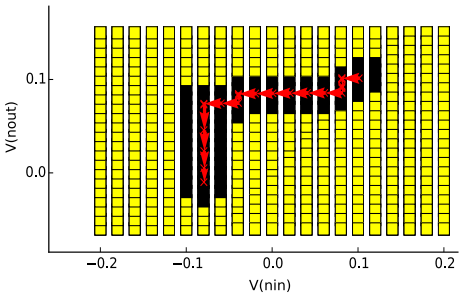


Figure 16: Coverage calculation for a transient simulation response (red) which marks a set of states inside the DATS as covered (black).

In addition to the model-checking approach, another way to ensure the implementation itself is to simulate the actual implementation. Typically, a designer creates tests to ensure the correct behavior of the implementation. For this, a set of input stimuli is created and afterwards simulated. To measure the quality of this set of stimuli, we introduce an *analog state space coverage* ζ which maps a transient simulation response with $|C|$ data points to the DATS:

$$\zeta = \frac{|C|}{|\Sigma_R|} \tag{6}$$

The calculation of an analog state-space coverage is shown in Figure 16. The trajectory (red crosses) through the DATS (yellow boxes) is the result of a simulation of a given input stimuli. A state is covered by that trajectory, if its Euclidean distance is under a given threshold. While full coverage ($\zeta = 1.0$) implies a complete tested design, the analog coverage metric can be also used to guide the designer to regions of the DATS which is not tested so far, reducing the number of possible design flaws and a faulty implementation. This method ensures that an implementation meets the specifications in any case.

Satisfaction Checking Results. The presented methods are applied to our application example. First, we created a testbench with the requested specifications (like C_{load} and R_{load}, supply voltage, etc.) and created the DATS for the second inverter B_1. Using ASL we're able to prove some characteristics of the circuit like power consumption P_{switch} during switching behavior and transition times T_{max} through the discrete state space. The following listing shows an ASL statement for checking the maximum value T_{max} on all transitions while switching the input of the inverter from a low to a high input signal.

```
in_up = Reach and nin[>1.4] and nout[<0.6];
in_down = Reach and nin[<0.6] and nout[>1.4];

numvar %max_time;
on Reach assign(%max_time, max)
  transition from in_up to in_down;
```

As seen in Table 1, all values for this component meet the specifications. The other inverter circuit B_0 is verified with the given contracts and specifications in the same way as described before. Using the analog state-space coverage method a set of stimuli are generated automatically to ensure the required specifications are met.

Table 1: Characteristics of the applied analog transistor level circuits

Inverter B_0		Inverter B_1	
PMOS p_0, w	4.5×10^{-6} m	PMOS p_1, w	9×10^{-6} m
PMOS p_0, l	180×10^{-9} m	PMOS p_1, l	180×10^{-9} m
NMOS n_0, w	2.0×10^{-6} m	NMOS n_1, w	4.0×10^{-6} m
NMOS n_0, l	180×10^{-9} m	NMOS n_1, l	180×10^{-9} m
W_{switch}	11.7×10^{-12} J	W_{switch}	28.9×10^{-12} J
T_{max}	37.6×10^{-12} s	T_{max}	86.1×10^{-12} s
C_{load}	23.0×10^{-15} F	C_{load}	92.0×10^{-15} F
R_{load}	200 Ω	R_{load}	500 Ω
TTL_{low}		0.0 V to 0.6 V	
TTL_{high}		1.4 V to 2.0 V	

5. Conclusion

This article summarizes novel methods and approaches for the verification of AMS designs from a coverage-oriented perspective.

We presented an approach allowing the mostly seamless integration of formal methods (symbolic simulation) into

an existing numerical simulator. Operator overloading and a few lines of code instrumentation are sufficient for interfacing to the conventional tool. Simulation of an industrial PLL design shows that the methodology can be successfully used with complex AMS designs. Automatic model transformation using property annotation and piecewise-linear modeling shows a way for verifying the influence of non-functional effects to the system behaviour. We demonstrated a methodology to identify system acceptance regions as well as critical points in the AMS system behaviour. Finally, we outlined our work on applying contract based design to the specification and verification of extra-functional and AMS properties. Using discrete analog transition systems, we derive a formal component characterization of the transistor level circuit, which is used for satisfaction checking the contracts.

The presented work aims at improving the speed and extending the scope of the AMS verification process. It builds the base for our ongoing work on coverage analysis in complex AMS systems.

Acknowledgments

The authors gratefully acknowledge partial financial support by the project ANCONA by the Federal Ministry of Education and Research (BMBF) under the grant number IKT 2020 16ES021.

References

[1] M. Andrade, J. Comba, and J. Stolfi, "Affine Arithmetic (Extended Abstract)," *Interval '94, St.Petersburg, Russia*, 1994.

[2] R. E. Moore, *Interval Analysis*. Eaglewood Cliffs, NJ: Prentice-Hall, 1966.

[3] C. Grimm, W. Heupke, and K. Waldschmidt, "Refinement of mixed-signals systems with affine arithmetic," in *Design, Automation and Test in Europe*, pp. 372–377, IEEE Comput. Soc, 2004.

[4] C. Radojicic, T. Purusothaman, and C. Grimm, "Symbolic Simulation of Mixed-Signal Systems with Extended Affine Arithmetic," in *Proceedings of EDAWORKSHOP 2015*, pp. 21–26, 2015.

[5] J. Stolfi and L. de Figueriedo, "An introduction to affine arithmetic," *TEMA Trend. Mat. Apl. Comput.*, vol. 4, pp. 297–312, 2003.

[6] M. Alassir, J. Denoulet, O. Romain, and P. Garda, "Signal integrity-aware virtual prototyping of field bus-based embedded systems," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 3, no. 12, pp. 2081–2091, 2013.

[7] K.-H. Kim, B.-S. Kong, and Y.-H. Jun, "Adaptive frequency-controlled ultra-fast hysteretic buck converter for portable devices," in *SoC Design Conference (ISOCC), 2012 International*, pp. 5–8, Nov 2012.

[8] M. Barnasconi and C. Grimm, eds., *SystemC AMS extension User's Guide*. OSCI, 2010.

[9] *clang: a C language family frontend for LLVM*.

[10] S. Hoelldampf, D. Zaum, M. Olbrich, and E. Barke, "Using analog circuit behavior to generate SystemC events for an acceleration of mixed-signal simulation," in *IEEE International Conference on Computer Design (ICCD)*, pp. 108–112, Oct. 2011.

[11] S. Hoelldampf, H. L. Lee, D. Zaum, M. Olbrich, and E. Barke, "Efficient generation of analog circuit models for accelerated mixed-signal simulation," in *IEEE International SOC Conference (SOCC)*, pp. 104–109, Sept. 2012.

[12] H.-S. L. Lee, M. Althoff, S. Hoelldampf, M. Olbrich, and E. Barke, "Automated generation of hybrid system models for reachability analysis of nonlinear analog circuits," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pp. 725–730, IEEE, 2015.

[13] L. O. Chua and A.-C. Deng, "Canonical piecewise-linear modeling," vol. 33, no. 5, pp. 511–525, 1986.

[14] W.-K. Chen, ed., *Feedback, nonlinear, and distributed circuits*. CRC Press/Taylor & Francis, 3rd ed., 2009.

[15] Y. Zhang, S. Sankaranarayanan, and F. Somenzi, "Piecewise linear modeling of nonlinear devices for formal verification of analog circuits," in *Formal Methods in Computer-Aided Design (FMCAD), 2012*, pp. 196–203, Oct. 2012.

[16] D. Zaum, S. Hoelldampf, M. Olbrich, E. Barke, and I. Neumann, "An accelerated mixed-signal simulation kernel for systemc," in *Specification Design Languages (FDL 2010), 2010 Forum on*, pp. 1 –6, sept. 2010.

[17] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. Larsen, "Contracts for systems design," Technical Report RR-8147, Research Centre Rennes Bretagne Atlantique, Rennes Cedex, 2012.

[18] E. A. Lee and A. L. Sangiovanni-Vincentelli, "Component-based design for the future," in *Design, Automation & Test in Europe (DATE)*, 2011.

[19] G. Nitsche and K. Grüttner, "Ams–/ef–contracts – a proposal of contracts for ams–verification and ams–coverage–analysis," in *Proceedings of the 2015 Forum on Specification and Design Languages, FDL 2015, Barcelona, Spain September 14–16, 2015*, 2015.

[20] G. Nitsche, R. Görgen, K. Grüttner, and W. Nebel, "Structural contracts – motivating contracts to ensure extra–functional semantics," in *In Proceedings of the fifth IFIP International Embedded Systems Symposium (IESS 2015)*, 2015.

[21] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta, "Validation of requirements for hybrid systems: A formal approach," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 4, 2012.

[22] A. Cimatti, M. Dorigatti, and S. Tonetta, "Ocra: A tool for checking the refinement of temporal contracts," in *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013.

[23] A. T. Davis, "An overview of algorithms in gnucap," in *University/-Government/Industry Microelectronics Symp.*, pp. 360–361, 2003.

[24] S. Steinhorst and L. Hedrich, "Trajectory-directed discrete state space modeling for formal verification of nonlinear analog circuits," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 202–209, ACM, 2012.

[25] S. Steinhorst and L. Hedrich, "Model checking of analog systems using an analog specification language," in *Proc. Design, Automation and Test in Europe DATE '08*, pp. 324–329, 10–14 March 2008.