View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Concordia University Research Repository

Formal Verification of Bond Graph Modelled Analog Circuits

William Denman[1], Mohamed H. Zaki[2] and Sofiène Tahar[1]

[1] Dept. of Electrical & Computer Engineering, Concordia University
1455 De Maisonneuve Blvd. West, Montreal, Quebec, Canada, H4G 1M8
{w_denm,tahar}@ece.concordia.ca
[2] Dept. of Computer Science, University of British Columbia
2329 West Mall, Vancouver, B.C. Canada, V6T 1Z4
mzaki@cs.ubc.ca

Abstract. Analog circuits are an increasingly critical component in embedded system designs. Traditionally, simulation is used for verification, but due to the infinite state space of analog components, the 100% correctness of a design cannot be guaranteed. Formal methods, based around applying mathematical expressions and reasoning to prove correctness, have been developed to increase the verification confidence level. This paper introduces and demonstrates a methodology for formally verifying safety properties of analog circuits. In the proposed approach, system equations are automatically extracted from a SPICE netlist by means of energy conservative bond graph models. Verification based on abstract model checking and constraint solving is then applied on the extracted equation models. Our methodology avoids an exhaustive and time demanding simulation that is normally encountered during analog circuit verification. To this end, we have used a set of tools to implement the proposed verification flow and applied it on tunnel diode, Chua and Colpitts oscillators as case studies.

1 Introduction

Analog circuits are an increasingly critical component in the verification flow of embedded system designs. Embedded devices are difficult to design and verify because of the interface between the digital (discrete) and analog (continuous) domains. Because

of the unpredictable nature of the real world input, the devices are required to operate over a large number of different modes that can be particularly difficult to determine, isolate and verify. For safety critical systems, where complete verification is required to ensure that an accident will not occur, this situation is particularly problematic.

The standard method to verify analog designs is simulation. With the increasing complexity, simulations can take days or even weeks to terminate [1]. Unfortunately, the results obtained via lengthy simulations can still remain incomplete. This is because it is impossible to test the entire set of inputs and expected outputs due to the continuous nature of the external signals. Only a finite number of cases can be checked. Therefore simulation methods lack the rigor to ensure the complete correctness of a design.

To address the incomplete verification of designs via simulation, formal methods have been developed to increase the verification confidence level. Formal methods [2] are based around applying mathematical expressions and reasoning to prove the correctness of a design. A formal specification is constructed to verify a model using mathematical logic and formal reasoning.

There have been several industry level applications developed for the formal verification of digital circuits [3]. However, there has not been the same amount of progress for analog circuits. This has severely limited the application of formal methods to embedded systems and other mixed signal devices. The current modelling methodologies used are not well suited for verifying several domains together. This requires that the verification of each part of a mixed signal design to be performed separately. It will be necessary to solve this problem before any significant progress will be made in the formal verification of embedded systems.

One way to model the complex behaviour of the analog part of an embedded system is by using a system of differential equations. One challenge in the verification process is to have an adequate model that accurately represents the behaviour of the design.

Unfortunately, generating equations that accurately represent this dynamic behaviour but are also simple enough to verify automatically remains a non-trivial process.

A critical problem facing analog designs are the effects arising from the reduction in fabrication size. These effects include parasitics, current leakages and component variations that can drastically change the expected behaviour of a design. This can cause major problems for the verification engineer because it is time consuming to build an appropriate model that accounts for this additional behaviour. Additionally, a great deal of expertise is required by the designer to extract and verify the properties of interest from the newly defined models. It is therefore of great utility to both the designer and the verifier to have models at their disposal that preserve the required behaviour of a device, yet remain simple enough to be verified using tools that are available.

This paper demonstrates a flow to verify functional properties of analog circuits. The different steps of the proposed methodology are shown in Figure 1. The methodology consists of two parts; namely modelling and verification. In the modelling phase, the circuit schematic is analyzed to obtain the system of ordinary differential equations (ODEs) necessary for the verification. The idea is to extract the circuit ODEs automatically from the corresponding analog circuit diagram, by means of bond graph transformations [4]. Two complementary approaches based on combining *predicate abstraction* and *constraint solving* are then applied to validate properties of interest during the verification phase. In particular, when the constraint solving based verification fails to return a result due to the complexity of the obtained ODE model, we can apply the predicate abstraction based method to obtain a result.

Bond graphs are a domain independent modelling formalism for physical systems based on the flow of energy between abstract objects. The benefit of using bond graphs for modelling is the ability to represents circuits using flow, effort and energy conservation. There are also switched bonds that can be used to represent discrete changes in behaviour. These properties allow for the universal treatment of different physical

domains. This is particularly useful for representing the behaviour of mixed signal systems.

Predicate abstraction [5] is one of the most successful abstraction approaches for the verification of systems with an infinite state space. In this approach, the state space is divided into a finite set of regions and a set of rules is used to define the transition between these regions in a way that the generated state transition system can be verified using model checking. Model checking is defined as: given a finite state model and a property, determine automatically whether the model satisfies the property [2].

Constraint solving [6] is concerned with verifying properties based on relations between the variables of a system. Problems are solved by forming constraints around a problem definition and by consequently finding solutions satisfying them all. For the constraint solving method, we use predicates to enhance the precision and computational cost of the state space exploration. However, in case that this method fails to provide an answer due to a state space explosion, an abstraction based verification is used. In this second approach, predicate abstraction is applied to generate an abstract state space that can be subsequently verified using model checking. In our approach, we validate the counterexample by again using constraint solving. In the case of a spurious counterexample, the abstract model can be refined [7].

The proposed methodology has the advantage of avoiding exhaustive simulation usually encountered during verification. To this end, we have used a set of tools to implement the verification flow. The design equations necessary for the verification are extracted from SPICE models using Dymola [8]. These equations are further simplified using Mathematica [9] simplification rules. HybridSal [10] is then used to obtain an abstract model which is verified using the SAL symbolic model checker (SAL-SMC) [11]. The HSolver [12] constraint solver is used alternatively for property verification and as a refinement procedure for counterexamples generated by SAL-SMC. We illus-

trate the methodology on several analog examples including Colpitts and tunnel diode oscillator circuits.

The rest of the paper is organized as follows: We start with an overview of the relevant work in Section 2. After that, we describe the different phases of the equation extraction process along with the theory behind bond graphs in Section 3. This is followed by an explanation of the proposed verification methodology in Section 4. The experimental results are presented in Section 5, before concluding the paper with Section 6.

2 Related Work

The presented verification methodology spans many different research domains. Therefore we will only highlight the most important information including the work on bond graphs for the analysis of analog designs.

Modelling analog circuits for formal verification. One of the main challenges of the formal verification of analog designs, is the development of models that preserve the required behaviour. Extracting the system equations to be used in behavioural modelling is a challenging task in the analog design process. Nodal analysis techniques have been developed to this aim by extracting equations from the circuit netlist. However, the resulting equations are in general, very large and too complicated to be used for a behavioural analysis. For example, in the context of formal verification, the authors of [13] relied on the symbolic analysis toolbox AnalogInsydes to obtain the system equations necessary for the verification.

In comparison with conventional symbolic extraction methods [14] and the techniques mentioned above, bond graph based modelling allows for a symbolic extraction of the system equations. This is possible because of methods to automatically assign an input-output relation (causality) to each component, generating a compact computa-

tional structure [15], that can be used to obtain differential equations.

Analog design verification. A common trend in analog verification is to use on-the-fly state space exploration techniques, where the set of reachable states correspond to an overapproximate solution of the system equations, over a bounded period of time. In an alternative approach, the entire state space is subdivided into regions where computational rules define the transitions between states. This model is generally described as a finite state automaton, verifiable using model checking techniques.

For instance, in the early work in [16], the authors constructed a finite-state discrete abstraction of electronic circuits by partitioning the continuous state space into fixed size hypercubes and then computing the reachability relations between these cubes using numerical techniques. In [17], the authors tried to overcome the expensive computational method in [16], by combining discretization and projection techniques of the state space to reduce its dimension. Similarly, the model checking tools d/dt [18], Checkmate [19] and PHaver [20] were adapted and used in the verification of a biquad low-pass filter [18], a tunnel diode oscillator and a $\Delta\Sigma$ modulator [19], and voltage controlled oscillators [20]. In [13], the authors used intervals to construct the abstract state space, while using heuristics to identify possible transitions between adjacent regions. The main difference with [16], is that they allow variable sized regions. An exhaustive state of the art review of the formal verification of analog designs can be found in [21].

Additionally, there exists work that is concerned with transforming the analog verification problem to one that can be solved with Boolean satisfiability (SAT) techniques. In [22], the authors have developed a methodology for formulating a SPICE style simulation into a format that can then be passed to a SAT solver. In particular, this technique can capture, at the transistor level, the non-linear behavior of the design under test.

Many of the surveyed formal methods limit the verification of the circuit to a predefined time bound because they depend on explicit state exploration. In contrast, we

propose in this paper to use qualitative based methods for the construction and verification of abstract models, which overcomes the time bound requirement. In addition we extend the verification with a counterexample guided refinement procedure.

For a more in-depth review of related work and other viable methods for the modelling and formal verification of analog circuits see [23].

3 Bond Graphs as a Model for Analog Circuits

Bond graphs were introduced by Paynter [24] who hypothesized that all physical systems and the interactions between them could be modeled using energy and power alone. His work was extended later on by Karnopp and Rosenberg [25] to enable the bond graph theory to be used in practice. They developed multi-port objects that could be used with power bonds to model the flow of energy and information [26]. The benefit of a modelling framework based on energy flow is that different domains can be analyzed using the same methodology. The necessary and sufficient set of bond graph primitives consist of five elements, but normally a more practical set of nine elements is used as shown in Table 1.

Table 1. Basic Objects of Bond Graphs

Group	**Components**	**Electrical Domain Example**
Storage	Capacitive/Inertial	Capacitance/Inductance
Supply	Source of effort/Source of flow	Voltage source/Current source
Reversible transformation	Transducer/Gyrator	Transformer
Irreversible transformation	Entropy producing process	Thermal Resistance
Distribution	0 and 1 junctions	KVL, KCL

Example. The tunnel diode oscillator circuit in Figure 2(a), which has been used by many researchers (e.g.,[19, 13]) as a benchmark in formal verification research, will be used as an example throughout the paper to demonstrate each step of our methodology.

The tunnel diode exploits a phenomenon called resonant tunneling due to its negative resistance characteristic at very low forward bias voltages. For certain ranges of voltages, the current decreases with increasing voltage. This characteristic makes the tunnel diode useful as an oscillator.

Connections. Bond graphs are based on the first principle of energy conservation. The most basic element of a bond graph is the power bond. It is the energy link between two components. It is represented graphically by a harpoon (half arrow), which points in the direction of positive energy flow (see Figure 2(c)). The bond represents two variables, effort and flow. In the electrical domain, the effort variable is represented by voltage and the flow by current. It follows that the product of the effort and flow variables represents the power flowing through the bond. Additional variables can also be derived from the bonds.

The next basic component is the junction, which represents a circuit node or mesh. At the 0 or common-effort junction the efforts are equal, which is analogous to a node in a circuit. At the 1 or common-flow junction, the flows are equal, which is analogous to a mesh in a circuit.

Components. Using the bonds and junctions, it is possible to connect components together in a bond graph, as shown in Figure 2(c). Single and multi-port bond graph elements are used to represent different topologies. The single port components are described below. The first basic elements are the sources of effort or flow. They are analogous to voltage and current sources in circuit diagrams. Additional single port components are used to represent resistors, capacitors and inductors. They are denoted using the letters *R*, *L* or *C*. Other components can be defined by an input/output functional relationship. In Figure 2(c) a tunnel diode is represent by the symbol *D*.

Causality. Causality is the determination and representation of the directional relationship between an input and an output [25] preserving the computational structure of the design. The causal stroke is attached to the side of the bond that computes the flow vari-

able [27] (Figure 2(d)). In general, causality is applied automatically using a technique such as the sequential causality assignment procedures (SCAP) to produce a causal bond graph [15]. By assigning causality, computational information of the system is available so that the system equations can be automatically extracted.

The fact that causality (algebraic dependency) is defined explicitly before any equations are setup remains a great advantage over other multi-domain modelling methods. Many practical analog circuits have a mathematical model that takes the form of a system of differential algebraic equations (DAE) with an index of one. It is well known that these models can be solved numerically for simulation purposes. For formal verification we require an analytical model and not a numerical approximation. Therefore it is generally necessary to use models that are available in state space form. Borutzky [28] has developed methods that use the causality information provided by bond graphs to identify tearing variables and equations to automatically reduce the DAE system into a state space model.

3.1 Analog Modelling Methodology

In the following, we present the methodology for automatically extracting the system of ODEs from an analog circuit. By using bond graphs we are able to conveniently model the topology of an analog circuit, which can aid at both the design and verification stages. The methodology is depicted in Figure 3.

Based on what behaviour or functionality is required in the design, the analog circuit is first constructed by hand with a schematic capture program that uses common symbols to represent the necessary components. This high level abstraction is then automatically transformed into a SPICE circuit model by macros contained within the schematic capture program. Using the Dymola Modelling Laboratory [8] in conjunction with the BondLib library [4], a bond graph is created directly from the SPICE model.

At this point, the bond graph is not in its simplified form. Using the simplification rules (see below), the bond graph is reduced. With the bond graph in its reduced form we are assured that the computational complexity is at a minimum. Next, the causality is automatically assigned by Dymola. Each bond graph component can have either a fixed, preferred or free causality assignment, determined by where the flow variable is calculated. For our verification task we want differential equations to be produced instead of integrals. This might take several iterations to complete since the overall causality assignment is constrained by the stating point of the SCAP algroithm and the resulting propagation of the choice through the bong graph.

Once the simplified and causal bond graph is formed, then Dymola is used again to automatically generate the Modelica description that contains the differential equations. For smaller designs the equations can be easily read directly from it. In other cases, when the design is more complex, the Modelica description may contain redundant equations due to the conversion process from DAEs to ODEs.

Generally, the equations representing the circuits are differential algebraic equations. Here, Dymola applies symbolic manipulation techniques in order to generate automatically the corresponding ODEs from the DAEs as described in [29, 28]. However, this comes at the cost of introducing dummy algebraic equations that can be simplified or eliminated using simplification rules within Mathematica. In this case, the simplification rules in the algebraic system Mathematica are employed to automate the ODE extraction. This process is further aided by using MathModelica [30], a Mathematica interface to the Modelica library.

The advantage of using BondLib is due to the symbolic nature of bond graphs. The behaviour of the corresponding SPICE models of the electrical components is preserved using a black box abstraction. For instance, the current through a transistor can be represented by a function, e.g., $I_{ds} = f(l, w, V_{gs}, ...)$. The internal details are chosen

independently of the verification method. In general, the verification might begin with a simple model and then more complex models can be substituted when needed.

3.2 Construction of the Bond Graph

Example. The transformation from a circuit diagram to bond graph is comparable to the SPICE model given in Figure 2(b). Each circuit diagram component is transformed into its bond graph counterpart. They are then interconnected by transforming nodes into 0 junctions and meshes into 1 junctions as shown in Figure 2(c). This is performed according to the bond graphs rules described earlier.

Simplification. There exists two levels of simplification that can be performed automatically on bond graphs. First, there are equivalence rules for the junction object [26]. These rules are used to reduce the number of bonds in a circuit and are based on the simplification of the underlying power equations. The equivalence rules can be performed automatically to a bond graph.

The second level of simplification is analogous to the concept of combining many parallel capacitances into one equivalent capacitor, which reduces the state space description. By choosing to combine certain bond graph elements, it is possible to reduce the complexity of the system without affecting the overall function. This can result in simpler ODEs that are extracted from the reduced bond graph model.

Example. There are several simplifications that can be made to the bond graph in Figure 2(c). First, the bonds that are connected to ground can be deleted since the voltage at those nodes is zero, indicating that the power flow is zero. Then, since the flows at 1 junctions are equal, 1 junctions in series can be merged together. As a final step to the simplification process, any junction that has only two bonds connected is removed since no power that flows through a two port junction can divert to another component.

The next step in the conversion process is to add a causality stroke to each bond (a straight line added to one end of a bond). Since there are two variables associated with each bond, the stroke indicates at which end the flow (current in the analog circuit sense) variable is calculated. To allow for an automated extraction of system equations, causality is assigned so that differential equations are obtained. For capacitors, the causal stroke is drawn at the opposite end of the bond away from the capacitor. For inductors, the causal stroke is drawn at the inductor end of the bond. The final bond graph is defined as shown in Figure 2(d).

3.3 Obtaining the System of Equations

Once the bond graph is built, the set of system equations can be extracted and simplified. In the current project, we use rewriting techniques provided in Mathematica to remove redundant equations. This is a mostly manual process. The final system of equations are the computational model on which we apply the verification. In general, the analog design computational model is described as below:

Definition 1. Analog Design Model.

An Analog Design Model is a tuple $\mathcal{A} = (\mathcal{X}, \mathcal{X}_0, \mathcal{U}, \mathcal{F})$*, with* $\mathcal{X} = V_{c_1} \times V_{c_n} \times \ldots \times I_{l_m} \subseteq \mathbb{R}^d$ *as the continuous state space with d-dimensions, where* V_{c_i} *and* I_{l_j} *are the voltage across the capacitance* C_i *and the current through the inductance* l_j*, respectively. The resistances in this case are memoryless (non-storage) elements.* $\mathcal{X}_0 \subseteq \mathcal{X}$ *is the set of initial states (initial voltages on the capacitances and currents through the inductance).* $\mathcal{U} \in \mathbb{R}^k$ *is the set of possible input signals to the design and* $\mathcal{F} : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}^d$ *is the continuous vector field.*

The analog design can then be described by the system of ODEs as follows:

Definition 2. System of ODEs

Consider a set of variables $x_k(t) \in \mathbb{R}$*,* $i \in \{1, \ldots, d\}$*,* $t \in \mathbb{R}$*, an ODE is a system consist-*

ing of a set of equations of the form:

$$\dot{x_k} = \frac{dx_k}{dt} = \dot{x} = \mathcal{F}_k(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$$

where $\boldsymbol{x}(t)$ are variables defining the voltage across the capacitance and the current through the inductance. $\boldsymbol{u}(t) \in \mathbb{R}^m$ are variables defining the input signals, with the vector fields $\mathcal{F}_k$.

The semantics of the analog model $\mathcal{A} = (\mathcal{X}, \mathcal{X}_0, \mathcal{F})$ over a continuous time period $T_c = [\tau_0, \tau_1] \subseteq \mathbb{R}^+$ ($t_1 = \infty$ in case of complete behaviour) can be described as a trajectory $\Phi_x : T_c \rightarrow \mathcal{X}$ for $x \in \mathcal{X}_0$ such that $\Phi_x(t)$ is the solution of $\dot{x_k} = \mathcal{F}_k(x_1, \ldots, x_d)$, with initial condition $\Phi_x(0) = x$ and $t \in T_c$, is a time point.

Example. With the simplified equations, we can now focus on the current I_L and the voltage V_C across the tunnel diode in parallel with the capacitor of the serial RLC circuit (Figure 2). The extracted simplified ODEs are given as $\dot{V_C} = \frac{1}{C}(-I_d(V_C) + I_L)$ and $\dot{I_L} = \frac{1}{L}(-V_C - \frac{1}{G}I_L + V_{in})$, where $I_d(V_C) = {V_C}^3 - 1.5{V_C}^2 + 0.6V_C$ that describes the non-linear tunnel diode behaviour.

4 Analog Design Verification

This section will describe the methodology for verifying properties of analog designs using ODEs extracted from bond graphs. There are two issues that must be addressed. First, we must determine what type of properties to verify. We have chosen to focus on verifying safety properties which indicate that some bad behaviour will never occur. The second task is to determine how to verify the properties over a continuous-time ODE model. A direct analysis over the continuous domain is too computationally expensive for the available verification technologies. We have therefore chosen to use an abstraction based technique. The goal is to reduce the required computational effort while preserving critical model behaviours thus ensuring valid verification results.

4.1 Preliminaries

Examples of questions that often come up during analog verification include: “will the system’s behaviour follow the design specification for the entire range of initial operating conditions?” and “considering component variations for a specific design technology will the transistors remain in the correct operating regions?”. Such questions can be easily redefined as safety properties in a temporal logic.

LTL (Linear Temporal Logic) is a logic language that defines properties by qualitatively describing their truth over time. There are four basic temporal operators : Fp meaning “eventually p”, Gp meaning “always p”, Xp “next time p” and pUq meaning “p until q”

Example. Consider the tunnel diode circuit with the set of parameters $\{C = 1000e^{-12}$ F, $L = 1e^{-6}$ H, $R = 2\ \Omega$, $Vin = 0.3$ V and the initial values $\{V_C = 0.131V$, $I_L = 0.055A\}$. Additionally, consider that due to inconsistencies in the fabrication process, the resistance will vary 10% from its nominal value. We verify that the preceding combination of parameters, initial conditions and parameter variations do not produce oscillatory behaviour. If the circuit does not oscillate, then the voltage will never pass the upper bound of 0.6 volts. The behaviour in question is stated as the safety property $\mathbf{G}(V_C \leq 0.6)$. The validation of the property verifies the non-existence of oscillation.

We cannot verify safety properties directly on the continuous time ODE model, due to its continuous nature. Therefore we must use an abstracted model.

4.2 Abstracting a Model

Definition 3. Transition System. *A transition system [31] is a 3-tuple $TS = (S, S_0, T)$ with a set of possible states S, a set of initial states S_0 and a set of transitions T.*

Definition 4. Abstraction. *A transition system $A = (\hat{S}, \hat{S}_0, \hat{E})$ with a finite set of states $\hat{S}$ is an abstraction [31] of a transition system $C = (S, S_0, T)$ if there exists an abstraction*

function $\gamma : S \rightarrow \hat{S}$ such that each initial state in the abstracted transition system ($\hat{s}_0$) can be related to the original transition system initial states (s_0) by $\hat{s}_0 = \gamma(s_0)$. Additionally, each abstract transition must correspond to an original transition between concrete states.

One viable abstraction technique is predicate abstraction, where the set of abstract states is encoded by a set of Boolean variables representing each a concrete predicate. Based on [32], we define a discrete abstraction of the analog model $\mathcal{A}$ with respect to a given n-dimensional vector of predicates over reals where each predicate is of the form $\psi : \mathbb{R}^d \rightarrow \mathbb{B}$, with $\mathbb{B} = \{0,1\}$ and d is the state variables numbers with $\psi(x) := \mathcal{P}(x_1,\ldots,x_d) \sim 0$, where $\sim \in \{<,\geq\}$. Hence, the infinite state space $\mathcal{X}$ of the system is reduced to 2^n states in the abstract system, corresponding to the 2^n possible Boolean truth evaluates of the set of predicates. We define the abstract behaviour of the analog circuit as a transition system that overapproximates that behaviour, which is guaranteed to contain real behaviour of the concrete circuit:

Definition 5. Abstract Transition System. An abstract transition system *is a tuple* $\mathcal{T}_\Psi = (Q_\Psi, \rightsquigarrow, Q_{\Psi,0})$, *where:*

- $Q_\Psi \subset L \times \mathbb{B}^n$ *is the abstract state space for a n-dimensional vector predicate, where an abstract state is defined as a tuple* (l,b), *with* $l \in L$ *is a label and* $b \in \mathbb{B}^n$.
- $\rightsquigarrow \subseteq Q_\Psi \times Q_\Psi$ *is a relation capturing abstract transition such that* $\{b \rightsquigarrow b' | \exists x \in \Upsilon_\Psi(b), t \in \mathbb{R}^+ : x' = \Phi_x(t) \in \Upsilon_\Psi(b') \wedge x \rightarrow x'\}$, *where the concretization function:* $\Upsilon_\Psi : \mathbb{B}^n \rightarrow 2^{\mathbb{R}^d}$ *is defined as* $\Upsilon_\Psi(b) := \{x \in \mathbb{R}^d | \forall j \in \{1,\ldots,n\} : \psi_j(x) = b_j\}$
- $Q_{\Psi,0} := \{(l,b) \in Q_\Psi | \exists x \in \Upsilon_\Psi(b), x \in \mathcal{X}_0\}$ *is the set of abstract initial states.*

In general, the effectiveness of the predicate abstraction method depends on the choice of predicates and the precision of the transition relation between abstract states. Several criteria are raised for the choice of appropriate predicates. For instance, basic ideas from the qualitative theory of continuous systems can be adapted within the predicate abstraction framework as proposed in [10, 33].

Predicates related to the basic functionality of the design of interest can also be provided in a manual fashion. The conventional analysis of circuits can be an interesting direction for obtaining useful predicates. It is worth noting that the termination of the predicate generation phase is not necessary for creating an abstraction. We can stop at any point and construct the abstract model. A larger predicate set yields a finer abstraction as it results in a larger state space in the abstract model.

Given the analog model transition system $\mathcal{T}_{\mathcal{A}}$ representing the analog behaviour and a property φ expressed using LTL. The problem of checking that the property holds in this model written as $\mathcal{T}_{\mathcal{A}} \models \varphi$ can be simplified to the problem of checking that a related property holds on an approximation of the model $\mathcal{T}_{\Psi}$, i.e., $\mathcal{T}_{\Psi} \models \varphi$. More formally, the main preservation theorem is stated as follows [31]:

Theorem 1. *Suppose $\mathcal{T}_{\Psi}$ is an abstract model of $\mathcal{T}_{\mathcal{A}}$, then for all LTL state formulas describing $\mathcal{T}_{\Psi}$ and every state of $\mathcal{T}_{\mathcal{A}}$, we have $\tilde{s} \models \varphi \Rightarrow s \models \varphi$, where $s \in \gamma(\tilde{s})$. Moreover, $\mathcal{T}_{\Psi} \models \varphi \Rightarrow \mathcal{T}_{\mathcal{A}} \models \varphi$.*

If a property is proved on an abstract model $\mathcal{T}_{\Psi}$, then we are done. If the verification of $\mathcal{T}_{\Psi}$ reveals $\mathcal{T}_{\Psi} \not\models \tilde{\varphi}$, then we cannot conclude that $\mathcal{T}_{\mathcal{A}}$ is not safe with respect to $\tilde{\varphi}$, since the counterexample for $\mathcal{T}_{\Psi}$ may be spurious. In order to remove spurious counterexamples, refinement methods on the abstract model are applied [31].

4.3 Verification Methodology

We have developed a verification methodology combining predicate abstraction and constraint solving to take advantage of the best parts of both techniques. Depending on the type of property, there are two complementary verification options to choose from.

Enhancing Constraints based Verification using Predicates. If the property we want to verify can be described as some upper limit on a variable, then the best option is to use a constraint solver due to its precision in representing a variable's trajectory. On

the left branch of Figure 4, we strengthen constraint solving based verification with predicates that act as constraints on the state space. This is practical as the addition of useful constraints can limit the state space exploration by providing a means for pruning unreachable states.

In this approach, we apply HybridSal on the system equations to obtain an abstract state graph of the circuit behaviour. The satisfaction of properties is verified on these regions using constraint based methods. The abstract graph, along with the system equations and the property of interest are then used as an input to HSolver. The property verification provides the advantage of avoiding explicit computation of reachable sets. If the property cannot be verified at this stage, refinement is needed only for the non-verified regions by adding more predicates using HybridSal. Verification is then applied on the newly generated abstract model.

HSolver has an internal abstraction refinement procedure. However, due to over-approximation, the refinement does not terminate unless there is a bound on it. When the bound is reached but verification does not terminate, a non conclusive answer is returned over an interval that violates the property. Refinement can be achieved by increasing the bound or choosing tighter constraints for the abstract states. In the case that the verification still fails even with refinement, the complementary approach that uses predicate abstraction can be used.

Predicate Abstraction based Verification. If the property under consideration is described using a temporal logic such a LTL, then the best option is the approach using abstract model checking. This is to take advantage of the significant number of advanced tools that can already prove properties on LTL formulas. On the right branch of Figure 4, symbolic model checking using SAL-SMC is applied on the abstract state space generated from HybridSal. The constraint based solver HSolver is used as a counterexample validation procedure for the abstract model checking SAL-SMC. At first, the abstract model is built automatically using the predicate abstraction tool HybridSal.

If the property verification succeeds, the approach terminates, otherwise an abstract counterexample is generated.

In abstract model checking, when a property cannot be verified, a counterexample is generated identifying the reasons for the possible property violation. As the generated counterexample is an abstract one, due to the overapproximation, it is essential to validate the counterexample. In case it is spurious, the information from it can be used in order to refine the abstract reachable states. The predicates specifying the counterexample are turned into constraints that are provided to HSolver, along the property and the system of ODEs. HSolver tries to validate the property only in the regions described by the provided constraints. If the property is verified, then we deduce that the counterexample is spurious and a refinement procedure based on removing spurious transitions is applied on the abstract model and symbolic model checking is re-applied on the refined model. On the other hand, if HSolver fails to provide a decisive answer about the property validation, the abstract model is refined by abstract states splitting which results by adding more predicates.

Note. There is no guarantee that a spurious counterexample can be refuted and the procedure might therefore not terminate again. Technically, this happens if the approximation is too loose and not precise enough, resulting in impossible behaviour. To our knowledge no efficient solution exists for such problems for hybrid systems. However, other practical counterexample validation have been proposed in [31].

4.4 Verification of a Tunnel Diode Oscillator

We use the predicate abstraction option (right branch of Figure 4) for the verification of the tunnel diode oscillator. Once the simplified system of ODEs has been extracted, they can be used to form a hybrid system definition in the HybridSal modelling language.

The variables of an analog circuit lie within a continuous state space and thus pose a problem for the formal verification tools that prove properties over a finite state space. To decrease the computational complexity of the verification problem, HybridSal uses

internal abstraction methods to encode the continuous state space into a discrete one defined by a set of predicates that are either greater than, less than or equal to zero. Ideally, the abstract model that is created should preserve enough of the critical behaviour of the design to verify the safety property under question [10].

The tunnel diode circuit is first manually transformed into a HybridSal description (see Listing 1). The HybridSal tool automatically generates the discrete abstract model (see Listing 2).

```
TunnelDiode:CONTEXT =
BEGIN
control : MODULE =
 BEGIN
 LOCAL v : REAL
 LOCAL vdot : REAL
 LOCAL i : REAL
 LOCAL idot : REAL
 LOCAL r : REAL
INVARIANT
        TRUE
 INITFORMULA
        v = 131/1000 AND i = 55/1000 //Initial Values
        r > 45 AND r < 55 //Variation on r
TRANSITION //Behaviour of system described using ODEs
[
   v > 0  -->
   vdot' = 1000*(-1*(v*v*v-15/10*v*v+6/10*v) + i);
   idot' = (-v - r*i + 3/10)
]
 END;
 G( ss:[control.STATE -> BOOLEAN]):[control.STATE -> BOOLEAN];
 correct: THEOREM
        control |- G(v < 6/10}; //Property to be Verified
END
```

Listing 1. HybridSal Tunnel Diode Description

The continuous variables are indicated by the constants *v* and *i*. Their derivatives are *vdot* and *idot*. They represent the current through the inductor and the voltage across the capacitor. The *INITFORMULA* section indicates that the initial value of *v* is 0.131 V

and for *i* is 0.055 A. The constraints on the variance of the parameter R are also defined in this section. The formulas in the *TRANSITION* section describe the conditions for switching between states as well as the differential equations defined over each mode of operation. In the case of the tunnel diode, there is only one mode of operation. But for instance, a MOSFET could be defined over three modes of operation (cut-off, triode and saturation). The second to last line contains the property to be verified, defined using LTL.

In general, the hybrid system definition has both discrete and continuous sections that allow the entire behaviour to be modeled. The system of ODEs that were extracted from the bond graph model can be put directly into the TRANSITION section of the HybridSal description.

Three polynomial predicates have been used to discretize the state space and are labeled *g0, g1* and *g2*. The ODEs in the *TRANSITION* section have been converted into abstract functions dependent on these predicates. We have omitted the definitions of the abstract functions *ASSVP, ASSVD123* and *INV3*. The property to be verified itself has also been converted into an LTL definition using the predicates.

This abstract model is checked using SAL-SMC to verify the non oscillation property. In this case, the SAL-SMC tool returns that the property is not proved and gives a counterexample (see Listing 3). The counterexample shows the values of the predicates as the model checker steps through each abstract state. The abstract property states that the predicate $g1$ must always be negative. However, the generated counterexample demonstrates a path to where the $g1$ predicate is zero. At this point, it is necessary to check whether the counterexample is spurious or not.

The next step in the tunnel diode circuit verification is to validate the counterexample produced by the SAL-SMC tool. By coding the predicates and transitions specified in the counterexample into an HSolver description (see Listing 4), we can perform a more precise examination of the reachable states. If it is determined that the counterex-

```
//Generated Predicates
g2 —> v
g1 —> v − 3/5
g0 —> −1∗v^3 + 3/2∗v^2 − 3/5∗v + i

//Abstraction
TunnelDiodeABS: CONTEXT =
 BEGIN
 SIGN: TYPE = {pos, neg, zero}; // Qualitative variables
  control: MODULE = BEGIN
    GLOBAL
    g0: SIGN
    GLOBAL
    g1: SIGN
    GLOBAL
    g2: SIGN
    INITIALIZATION
    g2 = pos; g1 = neg; g0 = neg
    TRANSITION // Abstract transitions
    [g2 = pos AND INV3(g2', g1', g0')
      —>
     g2' IN ASSVP(g2, g0); g1' IN ASSVP(g1, g0);
     g0' IN ASSVD123(g0, FALSE,
                     g1=zero AND g0=neg OR g0=zero AND g1=zero,
                     g1=zero AND g0=neg OR g0=zero AND g1=zero)]
    END;
  correct: THEOREM control |− G(g1 = neg); //Abstracted Property
 END
```

Listing 2. SAL Description for the Abstract Model of the Tunnel Diode Circuit

ample is never reached then the spurious transitions can be removed from the abstract model.

The HSolver description is described as follows. The variables of the system defined in the *VARIABLES* section are *v*, *i* and *r*. The modes of the system are named *m1*, *m2*, *m3* and *m4* in the MODES section. The tunnel diode model is defined over four modes of operation in the *SATESPACE* section, which represent the 4 steps of the counterexample. The initial values and differential equations defined in the *FLOW* section are the same as defined in the HybridSal description. The differential equations are defined over the discrete modes and they have a "*_d*" appended to their variable names. The

```
INVALID, building counterexample...

Counterexample:
===========
PATH
===========
Step 0:
--- System Variables (assignments) ---
g0 = neg
g1 = neg
g2 = neg
---------------------
Step 1:
--- System Variables (assignments) ---
g0 = zero
g1 = neg
g2 = pos
---------------------
Step 2:
--- System Variables (assignments) ---
g0 = pos
g1 = neg
g2 = pos
---------------------
Step 3:
--- System Variables (assignments) ---
g0 = pos
g1 = zero //Violates the abstract property G(g1=neg)
g2 = pos
---------------------
```

Listing 3. SAL-SMC Generated Counterexample from the SAL Tunnel Diode Description

jump conditions define when the system switches modes and in this example the conditions are how predicates change value in the SAL-SMC generated counterexample. The safety constraints of the system are defined in the *UNSAFE* section.

HSolver outputs "SYSTEM SAFE" which indicates the path to the abstract state of the counterexample produced by the SAL-SMC tool is never reached. We can therefore conclude the counterexample is spurious. Therefore, we manually remove from the SAL description all transitions from states where predicate *g1= neg* holds to states where *g1= zero* holds. This refinement is valid because by applying the cone of influence [34]

```
VARIABLES [v,i,r]
MODES [m1,m2,m3,m4]
STATESPACE
m1[[-0.5,1.2],[-0.5,0.2],[40,50]]
m2[[-0.5,1.2],[-0.5,0.2],[40,50]]
m3[[-0.5,1.2],[-0.5,0.2],[40,50]]
m4[[-0.5,1.2],[-0.5,0.2],[40,50]]
INITIAL
   m1{v=0.131/\i=0.055/\r>45/\r<55}  //Constraints
FLOW
m1{v_d=1000*(-(v*v*v-1.5*v*v+0.6*v) + i)}{i_d=(-v - 50*i + 0.3)}
   {r_d=0}
m2{v_d=1000*(-(v*v*v-1.5*v*v+0.6*v) + i)}{i_d=(-v - 50*i + 0.3)}
   {r_d=0}
m3{v_d=1000*(-(v*v*v-1.5*v*v+0.6*v) + i)}{i_d=(-v - 50*i + 0.3)}
   {r_d=0}
m4{v_d=1000*(-(v*v*v-1.5*v*v+0.6*v) + i)}{i_d=(-v - 50*i + 0.3)}
   {r_d=0}
JUMP
m1->m2{v*v*v+1.5*v*v-0.6*v+i=0/\[i'=i/\v'=v]}  //Transition
m2->m3{v*v*v+1.5*v*v-0.6*v+i>0/\[i'=i/\v'=v]}  //relations
m3->m4{v-0.6=0]/\[i'=i/\v'=v]}
UNSAFE
m4{v>=0.6}   //Possible unsafe state
```

Listing 4. HSolver Description for the Counterexample Validation of the Tunnel Diode Example

on the SAL description, we find that *g1* depends only on *g0* and not *g2* through the function *ASSVP(g1, g0)*. This is the reason why the jump conditions implemented in the HSolver description is based only on the *g0* and *g1* predicates. The verification on the refined SAL description using SAL-SMC in that case succeeds, which means that no oscillation will occur.

5 Experimental Results

In this section we detail our experimental results that serve as extensions to the tunnel diode oscillator example that was developed progressively throughout the paper. In particular we apply the proposed verification methodology on a BJT Colpitts Oscillator using predicate abstraction and a Chua Circuit using constraint solving.

5.1 BJT Colpitts Oscillator

The Bipolar Junction Transistor (BJT) Colpitts oscillator (Figure 5) is another example of an oscillator circuit that has a complex behaviour, which can be properly modeled with a piecewise linear approximation consisting of two modes.

In order to fully understand the behaviour of a circuit, it is important to verify its different modes of operation. In particular, transistors can be biased in different regions depending on the required application. It is particularly important to know the mode of operation when connected with other circuit components. This type of circuit analysis is usually done by hand as simulation data cannot always be used to conclusively determine the mode over all input values. We can apply the verification methodology to ensure that the transistor will never go into an unsafe mode of operation.

Another difficult issue that arises with verifying semi-conductor devices is the variation of component values due to fabrication tolerances. In the case of a BJT, one parameter that can change across a piece of silicon is the common-emitter current gain β. For modern devices, β can vary between 50 to 1000 [35].

Verifying Oscillation. When oscillating, the BJT of Figure 5 will never go into its saturation region. In fact, the BJT will either be in the cut-off mode or forward active modes [36]. The state space is subdivided into four regions according to the BJT modes of operations (Cut-off, Reverse active, Forward active and Saturation) with threshold voltage $V_{th} = 0.75$. For instance, the property that no transition occurs from Forward active to Saturation, can be validated by proving that $G((V_{C_1} > 0)$ is True, where V_{C_1} is the voltage across the capacitors C_1.

From [36], the differential equations describing the behaviour of the BJT Colpitts oscillator are

$$\begin{aligned} C_1\dot{V}_{CE} &= I_L - I_C \\ C_2\dot{V}_{BE} &= -\frac{V_{EE}+V_{BE}}{R_{EE}} - I_L - I_B \\ L\dot{I}_L &= V_{CC} - V_{CE} + V_{BE} - I_L R_L \end{aligned}$$

The BJT can be modeled as a two-segment piecewise-linear voltage-controlled resistor with

$$I_B = \begin{cases} 0 & \text{if } V_{BE} \leq V_{TH} \\ \frac{V_{BE}-V_{TH}}{R_{ON}} & \text{if } V_{BE} > V_{TH} \end{cases}$$

Consider the BJT Colpitts circuit with the following parameters, $V_{CC} = 5$ V, $R_L = 35\,\Omega$, $C_1 = C_2 = 54$ nF, $R_{EE} = 400\,\Omega$, $V_{EE} = -5$ V, $L = 98.5$ uH, $I_s = 1.43 \times 10^{-14}$, $R_{ON} = 100\,\Omega$. Also assuming that β varies between 50 and 1000. With the ODEs and the circuit parameters we can construct the HybridSal model containing the model of the system (see Listing 5).

The *INITFORMULA* section contains constraints on the variables of the system as well as constraints on the initial conditions. The parameters of the system are defined at the beginning of the transition section.

With the system of differential equations described using the HybridSal syntax. We can run the abstraction algorithm. The generated abstract state description contains the predicates and abstract transition functions as shown in Listing 6.

Now we take the abstract description and pass it to the SAL-SMC. As expected a counterexample is generated (see Listing 7). We then convert the predicates as described in Listing 6 into constraints. As well we express the counterexample path in terms of transitions in the HSolver format (see Listing 8). By removing those predicates that do

```
INITFORMULA
vc1 > 1 AND
vc2 > -Vth AND
Bf > 50 AND Bf < 1000} //Constraint on Beta

TRANSITION
Vth = 75/100;       //Parameter List
C = 500000000/27;
Ron = 100;
Vcc = 5;
RL = 35;
Vee = -5;
Ree = 400;
L = 49/500000;
[ //System of ODEs
vc2 >= -Vth -->
        vc1dot' = 1/C*iL;
        vc2dot' = 1/C*((Vee-vc2)/Ree+iL);
        iLdot' = 1/L*(Vcc-vc1-vc2-iL*RL)
[]
vc2 < -Vth -->
        vc1dot' = 1/C*(iL-Bf*(-vc2-Vth)/Ron);
        vc2dot' = 1/C*((Vee-vc2)/Ree+iL+(-vc2-Vth)/Ron);
        iLdot' = 1/L*(Vcc-vc1-vc2-iL*RL)
]
END;

G(ss:[control.STATE -> BOOLEAN]):[control.STATE->BOOLEAN];
 correct: THEOREM
        control |- G(vc1>3/10)); //Property of Interest
```

Listing 5. HybridSal Description of the Colpitts Oscillator

not change value, we can simplify the input into HSolver. HSolver indicates that the constraints and the property are safe, meaning that the counterexample path is spurious. The transitions to the counterexample are then removed from the abstract model and then model checking is applied again. With the spurious counterexample removed, the property is proved by the SAL-SMC.

Verifying Non-Oscillation. Consider the same BJT Colpitts circuit but with $R_{EE} = 20\,\Omega$ and the other parameters unchanged. Applying predicate abstraction results in

```
%% Abstract variable to Polynomial Mapping:
%% g6 --> vc1 - 3/10
%% g5 --> vc1 - 1
%% g4 --> vc2 + Vth
%% g3 --> Bf - 50
%% g2 --> Bf - 1000
%% g1 --> -1/400*vc2 + iL - 1/80
%% g0 --> iL

correct: THEOREM control |- G(g6 = pos) //Abstracted Property
```

Listing 6. Predicate Snapshot from the Abstract Model of the Colpitts Oscillator

true counterexamples that cannot be refuted. We can then attempt our constraint based approach at verification.

If the circuit is oscillating, we know from previous designs that the voltage across C1 will vary between 2 and 6 volts. If the voltage never passes the upper bound of 2 volts, then we can deduce that the circuit is not oscillating.

Taking as input Listing 9, HSolver responds with "INPUT SAFE". This indicates that for the new resistance choice, the voltage across capacitor 1 will never increase beyond the bound of 0.5 volts. This proves conclusively that the circuit does not oscillate.

5.2 Chua Circuit Example

We use the constraint based verification verification approach (left branch of Figure 4) described in Section 4 in order to verify the circuit shown in Figure 6(a). This circuit was designed and implemented by Chua [37] to demonstrate the behaviour of chaos. This is illustrated with simulation as shown in Figure 6(b). The important component of the circuit is the non-linear resistance that is the source of the chaotic behaviour. The non-linear resistor has distinct operating modes which allow the state space to be divided up to three piecewise linear regions [38]. The capacitors are assumed to have initial voltage values, explaining the lack of a source in the circuit.

Equation 1 represents the current-voltage relationship of the non-linear resistance, where V_e is the voltage where the model switches modes. G_a and G_b are the slopes of the curve in each of the corresponding modes.

$$I_{NR}(V_{C1}) = \begin{cases} G_b(V_{C1}+V_e) - G_aV_e & \text{if } V_{C1} < -V_e \\ G_aV_{C1} & \text{if } -V_e < V_{C1} < V_e \\ G_b(V_{C1}-V_e) + G_aV_e & \text{if } V_{C1} > V_e. \end{cases} \quad (1)$$

We are interested in verifying the property that the chaos of the circuit is bounded for a given set of parameters. This can be specified using the safety property $\mathbf{G}[-6 \leq V_{c_1} \leq 6]$ on the voltage across the capacitor C_1 shown in Figure 6(a).

In order to apply the proposed verification approach, the circuit diagram in Figure 6(a) is transformed to the corresponding bond graph. Simplification rules are then applied to obtain a reduced bond graph as shown in Figure 6(c). From the reduced bond graph, we obtain using the Dymola/Modelica tool a corresponding set of equations that are further processed by Mathematica in order to obtain the simplified set of equations. The different abstract regions are formed by the predicates extracted using HybridSal. The state space was split into three operating regions to define the different modes of operation of the non-linear resistor. The system equations and the safety property are then combined into the HSolver description (see Listing 10).

As with the Tunnel Diode example, the description contains four important sections. First the *STATESPACE* section describes the environmental constraints. The *FLOW* section describes the simplified ODEs that determine the behavior in each mode. The *JUMP* section contains the transition rules and the *UNSAFE* section defines the constraints to check. The results from HSolver indicate that when the proper parameters are chosen for the components, the voltage across the conductance indeed remains bounded within -6 and 6 volts.

6 Conclusion

In this paper, we proposed a novel approach for the formal verification of analog circuits. The major contributions are the following: We demonstrated how bond graphs provide an efficient means for modelling analog circuits for formal verification. We have presented an example of a tunnel diode oscillator that was successfully translated into a bond graph, and had its ODEs automatically extracted.

For the verification, we combined predicate abstraction and constraint solving into one methodology, which does not require an explicit representation of the entire state space and relies on functions that prove or disapprove circuit properties.

To scale the methodology to larger designs will require further analysis and development of the tools that were used. In particular, even thought the Dymola Modelling Laboratory can compile and generate Modelica code in seconds, a significant amount of computational effort is needed to extract the ODEs from the Modelica code and to remove redundant equations. As well HSolver, an experimental tool, is not suitable for the verification of large examples on its own due to its computationally expensive algorithm. This fact motivated its use primarily for counterexample refutation. There is ongoing development of efficient methods to address these specific limitations.

Comparing our formal verification methodology to simulation, we see that we can reduce the required effort while increasing the reliability of the results. In the case of trying to verify a range of parameters, with simulation it would be necessary to check several test-cases at the limits of the range and at several randomly chosen points. Even with positive results, there still remains a chance that an error remains, since each value has not been checked. With formal verification, we can say conclusively that all values within the range will result in correct operation of the design. More details about the analysis and formal verification of analog circuits can be found in [23].

The greatest advantage of our methodology is the lack of the timed bound limitation associated with explicit reachability analysis methods commonly encountered in

the formal verification of analog designs.

Future Work. Main future directions include the extension of the proposed approach to analog and mixed signal designs. This is a realistic goal since bond graphs are domain independent. A more recent addition to the bond graph methodology, the switched bond graph, could be used rather than the conventional one presented in this paper. The switched bonds allow for the modelling of systems where switching occurs such as in delta-sigma converters.

By moving to the mixed signal domain, it will be necessary to extend the verification methodologies to analyze the discrete parts of the state space. There currently exists a good amount of formal tools that can analyze moderately sized digital designs. The difficulty will be linking the tools presented in this paper with those that already exist. This will additionally require an exploration of case studies that are based around interesting functional properties. This will include the verification of the behaviour of transistors (verifying the mode of operation) that could be further extended to more complex properties such as the gain of filters.

References

1. Chang H, Kundert K. Verification of Complex Analog and RF IC Designs. Proceedings of the IEEE. 2007;95(3):622–639.
2. Kropf T. Introduction to Formal Hardware Verification. Springer; 1999.
3. Abrial JR. Faultless Systems: Yes We Can! Computer. 2009 September;42(9):30–36.
4. Cellier FE, Clauss C, Urquia A. Electronic Circuit Modelling and Simulation in Modelica. In: Proc. Eurosim Congress on Modelling and Simuulation. vol. 2; 2007. p. 1–10.
5. Graf S, Saidi H. Construction of Abstract State Graphs with PVS. In: Computer Aided Verification. vol. LNCS 1254. Springer; 1997. p. 72–83.
6. Ratschan S. Continuous First-Order Constraint Satisfaction. In: Artificial Intelligence, Automated Reasoning and Symbolic Computation. vol. LNCS 2385. Springer; 2002. p. 181–195.
7. Clarke E, Grumberg O, Jha S, Lu Y, Veith H. Counterexample-Guided Abstraction Refinement. In: Computer Aided Verification. vol. LNCS 1855. Springer; 2000. p. 154–169.
8. Dassault Systemes. The Dymola Modelling Laboratory;. Available from: `http://www.dymola.com/index.htm`.

9. Wolfram S. Mathematica: A System for Doing Mathematics. Computer Addison Wesley Longman Publishing; 1991.
10. Tiwari A. Series of Abstractions for Hybrid Automata. In: Hybrid Systems: Computation and Control. vol. LNCS 2289. Springer; 2002. p. 465–478.
11. de Moura LM, Owre S, RueB H, Rushby JM, Shankar N, Sorea M, et al. SAL 2. In: Computer Aided Verification. vol. LNCS 3114. Springer; 2004. p. 496–500.
12. Ratschan S, She Z. Safety Verification of Hybrid Systems by Constraint Propagation based Abstraction Refinement. ACM Transactions in Embedded Computing Systems. 2007;6(1):1–23.
13. Hartong W, Klausen K, Hedrich L. Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking. In: Advanced Formal Verification. Kluwer; 2004. p. 205–245.
14. Vlach J, Singhal K. Computer Methods for Circuit Analysis and Design. Kluver; 1993.
15. Maehne T, Vachoux A. Proposal for a Bond Graph Based Model of Computation in SystemC-AMS. In: Proc. Languages for Formal Specification and Verification, Forum on Specification and Design Languages; 2007. p. 25–31.
16. Kurshan RP, McMillan KL. Analysis of Digital Circuits Through Symbolic Reduction. IEEE Transactions on Computer-Aided Design. 1991;10(11):1356–1371.
17. Greenstreet MR, Mitchell I. Reachability Analysis Using Polygonal Projections. In: Hybrid System: Computation and Control. vol. LNCS 1569. Springer; 1999. p. 103–116.
18. Dang T, Donze A, Maler O. Verification of Analog and Mixed-Signal Circuits using Hybrid System Techniques. In: Formal Methods in Computer-Aided Design. vol. LNCS 3312. Springer; 2004. p. 14–17.
19. Gupta S, Krogh BH, Rutenbar RA. Towards Formal Verification of Analog Designs. In: Proc. IEEE/ACM International Conference on Computer Aided Design; 2004. p. 210–217.
20. Frehse G, Krogh BH, Rutenbar RA. Verifying Analog Oscillator Circuits Using Forward/Backward Abstraction Refinement. In: Proc. IEEE Design Automation and Test in Europe; 2006. p. 257–262.
21. Zaki M, Tahar S, Bois G. Formal Verification of Analog and Mixed Signal Designs: A Survey. Microelectronics Journal. 2008;39(12):1–10.
22. Tiwary SK, Gupta A, Phillips JR, Pinello C, Zlatanovici R. First Steps Towards SAT-based Formal Analog Verification. In: Proc. IEEE/ACM International Conference on Computer-Aided Design; 2009. p. 1–8.
23. Denman W. Towards the Automated Modelling and Formal Verification of Analog Designs [Master's Thesis]. Concordia University. Montreal, Quebec, Canada; 2009.
24. Fattah YE. Constraint Logic Programming for Structure-Based Reasoning About Dynamic Physical Systems. Artificial Intelligence in Engineering. 1996;1:253–264.
25. Broenink F. Introduction to Physical Systems Modelling with Bond Graphs. University of Twente; 1999.
26. Karnopp D, Rosenberg RC. Analysis and Simulation of Multiport Systems: The Bond Graph Approach to Physical System Dynamics. The MIT Press; 1968.

27. Cellier FE, Nebot A. The Modelica Bond Graph Library. Swiss Federal Institute of Technology; 2007.
28. Borutzky W. Supporting the Generation of a State Space Model by Adding Tearing Information to the Bond Graph. Simulation Pracite and Theory. 1999;7:419–438.
29. Mattsson SE, Olsson H, Elmqvist H. Dynamic Selection of States in Dymola. In: Modelica Workshop; 2000. p. 62–67.
30. Jirstrand M, Gunnarsson J, Fritzson P. A New Modeling and Simulation Environment for Mathematica. In: Proc. International Mathematica Symposium; 1999. Available at `http://www.modelica.org`.
31. Clarke E, Fehnker A, Han Z, Krogh BH, Stursberg O, Theobald M. Verification of Hybrid Systems based on Counterexample-Guided Abstraction Refinement. In: Tools and Algorithms for the Construction and Analysis of Systems. vol. LNCS 2619. Springer; 2003. p. 192–207.
32. Alur R, Dang T, Ivancic F. Reachability Analysis Via Predicate Abstraction. In: Hybrid Systems: Computation and Control. vol. LNCS 2289. Springer; 2002. p. 35–48.
33. Zaki M, Tahar S, Bois G. Qualitative Abstraction based Verification for Analog Circuits. Revue des Nouvelles Technologies de l'information. 2007;4:147–158.
34. Clarke EM, Grumberg O, Peled DA. Model Checking. MIT Press; 1999.
35. Sedra AS, Smith KC. Microelectronic Circuits. Oxford University Press; 2004.
36. Kennedy MP. Chaos in the Colpitts Oscillator. IEEE Transactions on Circuits and Systems. 1994;41(11):771–774.
37. Chua LO. Chua's Circuit : An Overview Ten Years Later. Journal of Circuits, Systems and Computers. 1994;4:117–159.
38. Kennedy MP. Three Steps to Chaos - Part I: Evolution. IEEE Transactions on Circuits and Systems *I*. 1994;41:771–774.

```
Counterexample:
=========================
Path
=========================
Step 0:
--- System Variables (assignments) ---
g0 = neg
g1 = neg
g2 = neg
g3 = pos
g4 = pos
g5 = pos
g6 = pos
---------------------------
Step 1:
--- System Variables (assignments) ---
g0 = neg
g1 = zero
g2 = neg
g3 = pos
g4 = zero
g5 = zero
g6 = pos
---------------------------
Step 2:
--- System Variables (assignments) ---
g0 = zero
g1 = zero
g2 = neg
g3 = pos
g4 = zero
g5 = neg
g6 = zero //Violates the property G(g6=pos)
-----------------
```

Listing 7. SAL-SMC Generated Counterexample for the Colpitts Oscillator

```
VARIABLES [vc1,vc2,iL]
MODES [m1,m2,m3]
STATESPACE
m1[[-1,6],[-1,2],[-0.01,0.05]]
m2[[-1,6],[-1,2],[-0.01,0.05]]
m3[[-1,6],[-1,2],[-0.01,0.05]]
INITIAL
  m1{vc2+0.75>0/\vc1-1>0} //Predicate initial values
FLOW //System of ODEs
m1{vc1_d=18518518*iL}{vc2_d=18518518*(0.0025*(-5-vc2)+iL)}
  {iL_d=10204*(5-vc1-vc2-iL*35)}
m2{vc1_d=18518518*iL}{vc2_d=18518518*(0.0025*(-5-vc2)+iL)}
  {iL_d=10204*(5-vc1-vc2-iL*35)}
m3{vc1_d=18518518*iL}{vc2_d=18518518*(0.0025*(-5-vc2)+iL)}
  {iL_d=10204*(5-vc1-vc2-iL*35)}
JUMP //Counterexample path
m1->m2{[-0.0025*vc2+iL-0.0125=0]/\[vc2+0.75=0]/\[vc1-1=0]/\
       [iL'=iL/\vc1'=vc1/\vc2'=vc2]}
m2->m3{[iL=0]/\[vc2+0.75=0]/\vc1-1<0/\vc1-0.3=0/\
       [iL'=iL/\vc1'=vc1/\vc2'=vc2]}
UNSAFE
  m3{vc1<0.3}
```

Listing 8. HSolver Counterexample Validation of the Colpitts Oscillator

```
VARIABLES [vc1,vc2,iL,Bf]
MODES [m1,m2]
STATESPACE
m1[[0,6],[-1,3],[-0.02,0.08],[25,1025]]
m2[[0,6],[-1,3],[-0.02,0.08],[25,1025]]
INITIAL
  m1{vc1=0/\vc2=-1/\Bf>50/\Bf<1000}  //Initial Conditions
                                     //Constraints on Beta
FLOW //System of ODEs
m1{vc1_d=18518518*iL-200*0.01*(-vc2-0.75)}
  {vc2_d=18518518*(0.05*(-5-vc2)+iL+0.01*(-vc2-0.75))}
  {iL_d=10204*(5-vc1-vc2-iL*35)}
  {Bf_d=0}
m2{vc1_d=18518518*iL}
  {vc2_d=18518518*(0.05*(-5-vc2)+iL)}
  {iL_d=10204*(5-vc1-vc2-iL*35)}
  {Bf_d=0}
JUMP
m1->m2{[vc2>-0.75]/\[iL'=iL/\vc1'=vc1/\vc2'=vc2]}
m2->m1{[vc2<=-0.75]/\[iL'=iL/\vc1'=vc1/\vc2'=vc2]}
UNSAFE
  m1{vc1>0.5}  //Conditions on vc1
  m2{vc1>0.5}
```

Listing 9. HSolver Description for Proving Non-oscillation of the BJT Colpitts Oscillator

```
VARIABLES [c1,c2,vi]
MODES [m1,m2,m3]
STATESPACE
m1[[-7,-1],[-0.5,0.5],[0,1]]
m2[[-1,1],[-0.5,0.5],[0,1]]
m3[[1,7],[-0.5,0.5],[0,1]]
INITIAL
      m3{c1=4/\c2=0/\vi=0}  // Initial Conditions
FLOW
      m1{c1_d=(0.565*(c2-c1)+0.409091*c1 - 0.757576)*0.1}
        {c2_d=-(0.565*(c2-c1) + vi)*0.01}
        {vi_d=(c2-12.5*10^-3*vi)*0.0555}
      m2{c1_d=(0.565*(c2-c1) + 0.757576*c1)*0.11}
        {c2_d = -(0.565*(c2-c1) + vi)*0.01}
        {vi_d=(c2-0.0125*vi)*0.0555}
      m3{c1_d=(0.565*(c2-c1) - 0.409091*(c1 - 1) + 0.757576)*0.1}
        {c2_d = -(0.565*(c2-c1) + vi)*0.01}
        {vi_d = (c2-0.0125*vi)*0.0555}
JUMP
m1->m2{[c1>-1]/\[c1'=c1/\c2'=c2/\vi'=vi]}  //Transitions
m2->m1{[c1<=-1]/\[c1'=c1/\c2'=c2/\vi'=vi]} //between
m2->m3{[c1>1]/\[c1'=c1/\c2'=c2/\vi'=vi]}   //each
m3->m2{[c1<=1]/\[c1'=c1/\c2'=c2/\vi'=vi]}  //mode
UNSAFE
m1{c1<-6}
m3{c1>6}
```

Listing 10. HSolver Description of the Chua Circuit

List of Figures

1 Proposed Verification Flow 38
2 Tunnel Diode Oscillator Example 39
3 Bond Graph Modelling Methodology 40
4 Overview of the Verification Methodology 41
5 BJT Colpitts Oscillator 42
6 Chua Circuit Example 43

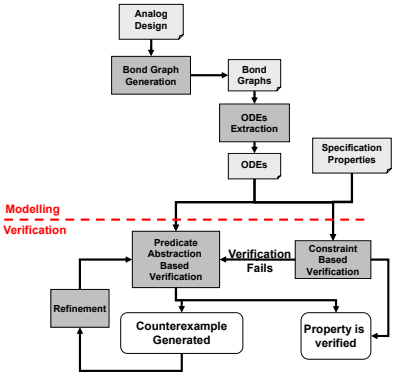
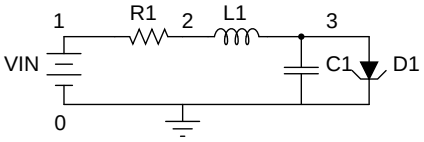


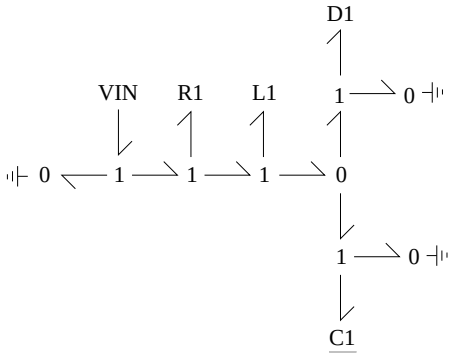
Fig. 1. Proposed Verification Flow



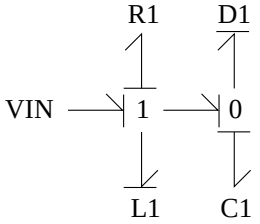
(a) Circuit Diagram

```
.MODEL TunnelDiode TD

VIN 1 0 DC 0.3
R1  1 2 50
L1  2 3 1u
C1  0 3 1p
D1  0 3 TunnelDiode

.END
```

(b) SPICE Code



(c) Initial Bond Graph



(d) Simplified Bond Graph

Fig. 2. Tunnel Diode Oscillator Example

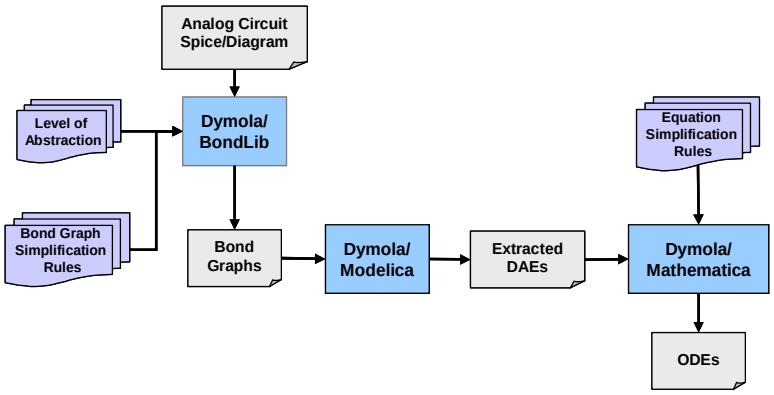


Fig. 3. Bond Graph Modelling Methodology

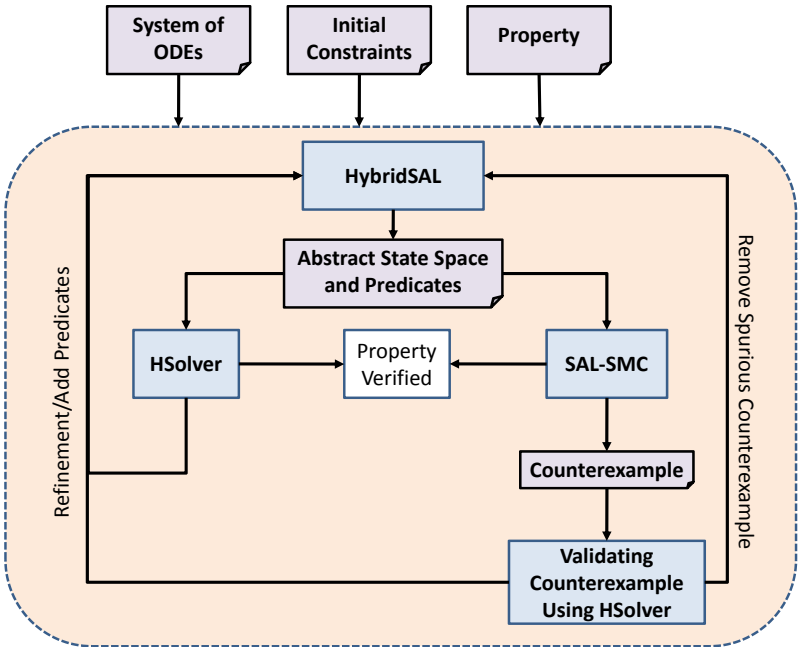


Fig. 4. Overview of the Verification Methodology

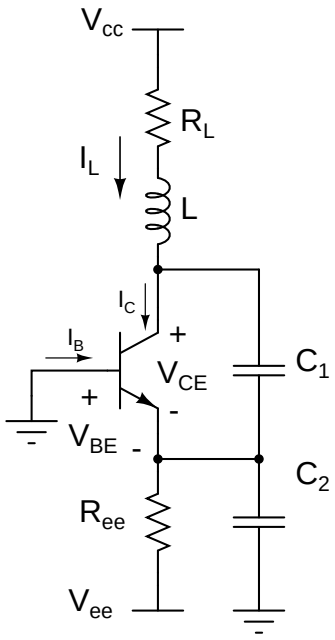
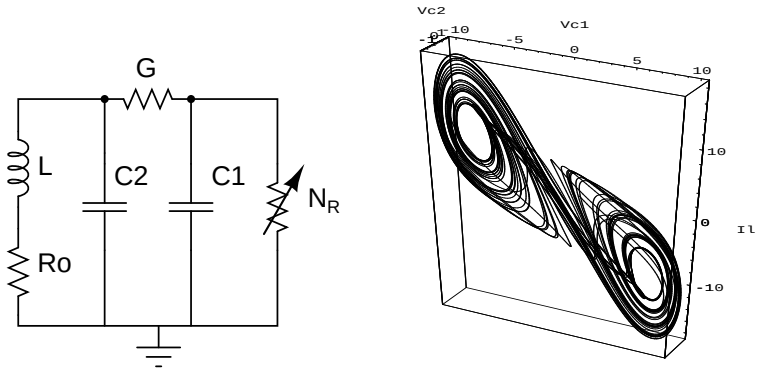
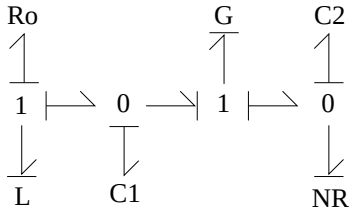


Fig. 5. BJT Colpitts Oscillator



(a) Circuit Diagram (b) Simulation (Mathematica)



(c) Bond Graph Model

Fig. 6. Chua Circuit Example