A Bond Graph Approach for Constraint based Verification of Analog Circuits

William Denman, Mohamed H. Zaki, Sofiène Tahar

Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., Montréal, Québec, H3G 1M8, Canada
{w_denm, mzaki, tahar}@ece.concordia.ca

Abstract. The computer-aided design community is in need of novel methodologies for the verification of analog circuits because of the growing importance of such circuits in embedded system designs. This paper demonstrates a verification flow of analog circuit functional properties. In the proposed approach, system equations are automatically extracted from an analog circuit description by means of bond graph transformations. Property verification based on abstract model checking and constraint solving are then applied to the extracted equations. The benefit of using bond graphs as a modelling framework is their representation of circuits using the concepts of energy flow, effort and conservation. Hence, allowing for several levels of abstraction. Our methodology has the advantage of avoiding exhaustive simulation normally encountered in the verification of analog circuits. To this end, we have used a set of tools (i.e., Dymola, HSolver, HybridSal and Mathematica) to implement the verification flow. We illustrate the methodology on several analog examples including Colpitts and tunnel diode oscillators.

1 Introduction

The verification of analog designs is a challenging task because of the complexity of modelling and verifying continuous-time behaviour, when compared to digital designs. For instance, digital design verification is based on the validation of abstract models that reside in a finite state-space. In contrast, the functionality of analog circuits depends on continuous electrical quantities, device parameters, in addition to parasitics and current leakage. All those factors can drastically change the behaviour of an analog circuit design making conventional finite-state verification techniques inadequate. Additionally, the dynamic behaviour of analog circuits can be generally modelled using systems of differential algebraic equations (DAE), but generating the equations from a circuit diagram and subsequently simplifying them for verification purposes is not trivial. Specifically, the DAEs must accurately describe the behaviour of the circuit while remaining simple enough to be verified using automated tools.

This paper demonstrates a verification 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 section, the circuit model is analyzed and simplified to obtain the system of ordinary differential equations (ODEs) necessary for the verification. The basic idea is to extract the circuit ODEs automatically from the corresponding analog circuit diagram, by

means of bond graph transformations [4]. Approaches based on combining predicate abstraction and constraint solving are then applied to verify the properties of interest.

In the first approach, we supply the constraint based verification with predicates that act as constraints on the state space which can enhance the state space exploration in terms of precision and computational cost. However, in case the constraint based methods fail to provide verification answers due to state space explosion, a second approach based on abstraction based verification is used. In this approach, predicate abstraction is applied to generate the abstract state space that is verified. When a property cannot be verified, a counterexample is generated identifying the reasons for the possible property violation. Validation of the generated counterexample is achieved by applying constraint solving. In case the counterexample is spurious, the information from it can be used in order to refine the abstract model.

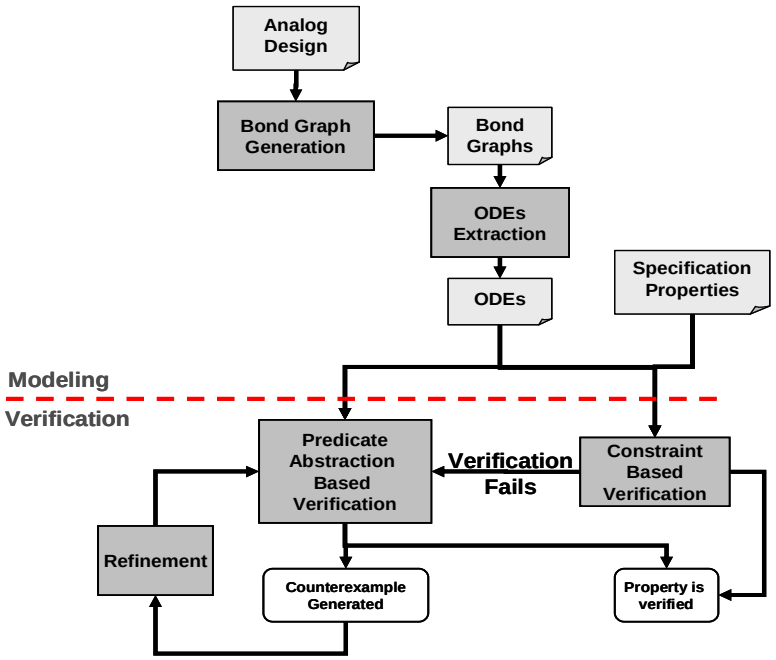


Fig. 1. Proposed Verification Flow

Bond graphs are a domain independent framework for modelling physical systems that is based on the flow of power between abstract objects. This allows for the universal treatment of different physical domains. The benefit of using bond graphs as a modelling framework is the representation of circuits using the concepts of energy flow, effort and conservation. Hence, allowing the modelling at several levels of abstraction while preserving the topological aspects of the circuit under consideration [2]. Additionally, the causality of bond graphs can be automatically generated [27], which leads to the automatic extraction of DAEs. A causality analysis can also produce an opti-

mization to the computational structure of the model that refines the extracted equations depending on the properties to be verified [27]. Moreover, since bond graphs are object oriented, larger models can be built from simpler blocks reducing the need for a complex equation layer [4]. Such characteristics allow us to compare the verification results of the design at different levels of abstraction. For instance, one abstraction of a bond graph can contain blocks that represent stray capacitances, while the other has them removed. The DAEs of the two models can be extracted and then used to verify if the capacitances play an important role in the circuit design. In summary, the ability of bond graphs to preserve the computational as well as the topological aspects of the circuits makes them an attractive tool in analog verification.

Predicate abstraction [16], 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. Recently, predicate abstraction has been extended for the verification of hybrid systems [1]. We propose using a qualitative abstraction approach for analog circuits, such that satisfaction of the property in the abstract model guarantees its satisfaction in the circuit-level model. In the proposed abstraction, the state space is initially partitioned based on the qualitative characteristics of the analog equations and constraint based methods are applied to check for property validation. When the property cannot be validated, one possible reason is because of the false negative problem due to the over-approximation of the abstraction. In case of failure, an iterative process is applied where the regions violating the property are refined and then verified again.

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 [12]. These equations are further simplified using Mathematica [36] simplification rules. HybridSal [34] is then used to obtain an abstract model which is verified using the SAL symbolic model checker (SMC) [30]. The HSolver [31] constraint solver is used alternatively for property verification and as a refinement procedure for counterexample generated by SAL-SMC. We illustrate 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 equations extraction process along with the bond graphs theory in Section 3. This is followed by an explanation of the proposed verification methodology in Section 4. Experimental results are provided in Section 5 before concluding the paper with Section 6.

2 Related Work

The presented verification methodology spans through many different research domains. Therefore we will only highlight the most crucial information including the work on bond graphs for the analysis of analog designs. A survey of the current research domains for the application of formal methods to the verification of analog designs will

be presented. Finally, we give an overview of the application of predicate abstraction and constraints solving methods for the verification of hybrid systems.

Modelling analog circuits for formal verification. One of the main challenges of the formal verification of analog designs, is the development of adequate models that preserve the required behaviour. For instance, continuous-time models can express in great detail the behaviour of a system and thus reside at the lower end of the abstraction scale. Such models are generally based on differential equations. Mathematical models that capture correct functional behaviour of the system as well as its physical characteristics are used as a means for specifying, analyzing and designing analog designs.

Significant effort is required to create an appropriate formal model for each different system. 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 behavioural analysis required at a higher level in the design process. For example, in the context of formal verification, the authors of [19] relied on the symbolic analysis toolbox AnalogInsydes to obtain the system equations necessary for the verification.

In comparison with conventional symbolic extraction methods [35], bond graph based modelling has several advantages as it provides a visual representation of the design. By construction, it checks for the consistency of the topological settings of the design. Moreover, it allows the hierarchical modelling of designs which can aid as an abstraction setting for the design. Finally, from the bond graphs, the system equations are extracted symbolically in a structured way.

Another approach that was developed recently is based on using simulation traces to generate a formal model which can be used during state space exploration. Such approach was used by Dastidar, *et al.* [9] to generate a finite state machine (FSM) from a set of simulation traces. A similar approach was proposed by Little *et al.* in [26], where they generated from simulation data, a hybrid petri net at the front-end to their model checker.

Unlike the above mentioned methods, bond graph based modelling allows a symbolic extraction of the system equations, hence providing more precise models which raises the confidence in verification.

Analog design verification. The most common trend in analog verification is using on-the-fly state space exploration techniques, where the set of reachable states correspond to the overapproximate solution of the system equations, which is obtained for a bounded period of time. An alternative approach is one where the whole state space is subdivided into regions and then computational rules define the transitions between states. Model checking algorithms are applied on the new abstract model of the system, which is generally described as a finite state automaton.

For instance, in the early work in [24], the authors constructed a finite-state discrete abstraction of electronic circuits by partitioning the continuous state space into fixed size hypercubes and computed the reachability relations between these cubes using numerical techniques. In [14], the authors tried to overcome the expensive compu-

tational method in [24], by combining discretization and projection techniques of the state space, hence reducing its dimension. While the approach in [14] is less precise due to the use of projection techniques, it is still sound. Variant approaches of the latter analysis were proposed. For instance, the model checking tools d/dt [10], *Checkmate* [18] and *PHaver* [13] were adapted and used in the verification of a biquad low-pass filter [10], a tunnel diode oscillator and a $\Delta\Sigma$ modulator [18], and voltage controlled oscillators [13]. In [19], the authors used intervals to construct the abstract state space, while using heuristics to identify possible transition between adjacent regions. The main difference with [24], is that they allow variable sized regions. Petri nets based models and algorithms have also been developed for the reachability analysis of analog designs in [25]. In [39], the authors proposed a non-linear approximation for reachable states, where the state space exploration algorithms are handled with Taylor approximations over interval domains. They used this technique along with symbolic manipulation analysis for the bounded model checking. More details about relevant related work on the formal verification of analog designs can be found in [41]

All 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 using 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 refinement procedure.

3 Analog Design Modelling

In analog design, it is convenient to model the circuits by specifying the corresponding topological description. This is generally achieved through schematic drawing or Spice netlist coding. The design equations are then extracted from the netlist. This goal is achieved in this paper through the methodology steps described in Figure 2.

At first, we require that the Spice model of circuit in question is described in Dymola [12], which can be translated automatically to the corresponding bond graph. The circuit components are then represented by generic objects that represent the same physical quantities as in the circuit diagram, but are connected by bonds that explicitly show the flow of power (the notion of bond graphs will be introduced in Section 3.1) . At this point simplification rules are applied to reduce bond graphs. For example, at the most basic level this entails combining two resistors that are in series. The modelling framework Dymola, which provides a library called BondLib, has been used to represent the bond graphs. The bond graphs are inherently acausal, but by assigning causality to the components, the system's equations can be automatically generated using Dymola BondLib [4]. Generally, the equations representing the circuits are differential algebraic equations (DAEs). Here, Dymola applies symbolic manipulation techniques in order to generate automatically the corresponding ODEs from the DAEs as described in [29]. However, this comes at the cost of introducing algebraic equations which can be then simplified or even eliminated using Mathematica simplification rules. (*Mathmodelica* [20] can also be useful at this stage).

The advantage of using Bondlib is that it preserves the behaviour of the corresponding Spice models of electrical components while allowing the modelling at several lev-

els of abstraction. For instance, MOSFETs can be represented using different Spice levels or can be specified through behavioural modelling.

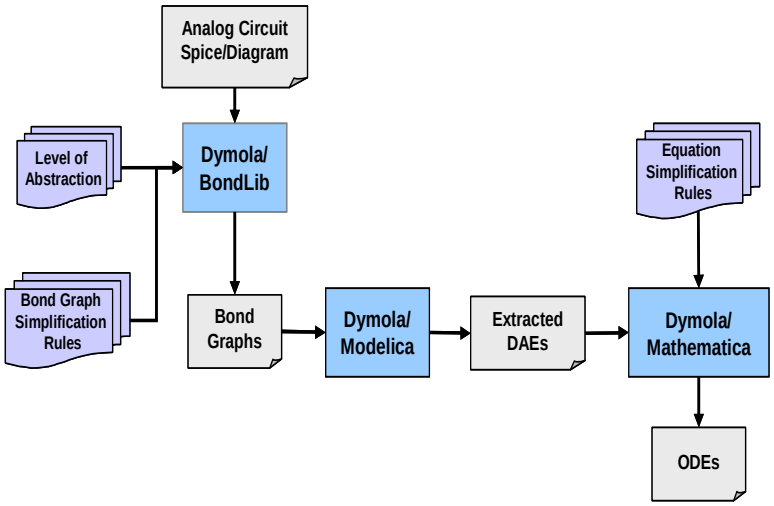


Fig. 2. Bond Graph based Modeling

3.1 Bond Graph as Model for Analog Circuits

Bond graphs were introduced by Paynter [38] 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 [3] 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 [21]. The benefit of a modelling framework based on energy flow is that different domains can be analyzed using the same methodology.

Bond graphs define a necessary and sufficient set of primitives for the modelling of a wide range of practical systems. The necessary and sufficent set of primitives consists of five elements, but normally a more practical set of nine elements is used as shown in Table 1. The storage group contains the elements for capacitive storage (C type) and inductive storage (I type). The supply group contains the sources of effort and flow. The reversible transformation group contains a transducer and gyrator. The irreversible transformation group contains the elements for thermal losses and entropy producing processes. While the distribution group contains junctions that represent the generalized domain independent KVL and KCL laws.

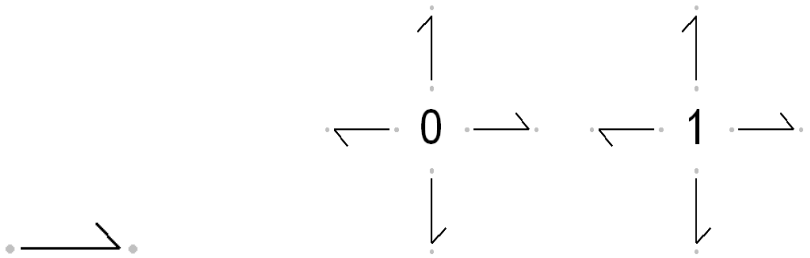
Connections. Bond graphs are based on the first principle of energy conservation. The most basic element of a bond graph is the power bond (Figure 3.a). It is the energy link between two components. It is represented graphically by a harpoon (half arrow), which

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

points in the direction of positive power flow. 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 displacement and momentum energy variables are related to the energy and flow by their time derivatives.

The next basic component is the junction, which represents a circuit node or mesh (Figure 3.b). 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.



(a) Power Bond (b) Mesh Bond

Fig. 3. Basic Bonds

Components. Using the bonds and junctions, it is possible to connect components together in a bond graph. There are different types of single and multi port interfaces that can be used to represent many configurations. 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* (See Figure 4.a).

It is possible to represent other electrical circuit components, such as transformers, gyrators and switches using two port interfaces but their application and description are beyond the scope of this paper. It is important to note though that more advanced

components exist and they can be used to model electronic components beyond simple analog ones.

(a) RC bond (b) Causality

Fig. 4. Bond Graphs Basics

We have now seen how a given bond graph and a set of constitutive relations maps to a mathematical model of the underlying system. A preferred alternative is a sequence of directed assignment statements such that unknowns can be immediately and sequentially computed from the knowns on the right hand side. Such a model is sometimes referred to as a computational model. Such a causal computational model requires the model variables to be ordered in a specific cause-effect relationship.

Causality. Causality is the determination and representation of the directional relationship between an input and an output [3] preserving the computational structure of the design. In fact, the causality concept is very important as it allows to detect any inconsistency in the circuit settings such as trying to connect two voltage sources with different voltage levels. By adding a causal bar to the end of a bond, the system equations that represent the two variables of effort and flow can be indicated explicitly. There are many rigorous explanations on how to assign the causality of a bond and how it relates to the system as a whole [3, 38, 21]. Fortunately, a simple definition exists that can be used for the direct translation of circuit diagrams. The causal stroke is attached to the side of the bond that computes the flow variable [5] (Figure 4.b). It is important for the modeler to know how to assign causality manually because it can aid in the development of complex bond graphs. However, in general causality is applied automatically using techniques like sequential causality assignment procedures (SCAP) leading to the construction of the causal bond graphs [27].

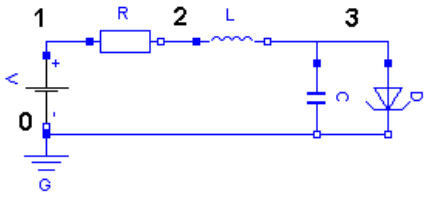
In summary, causality assignment is advantageous as it provides computational information of the system like the number of state space variables which leads to the automatic derivation of the system equations. It also aids in checking for the presence of algebraic loops during the model execution, which results in complex DAEs. Additionally, causality analysis is very useful in detecting ill posed models and can give insight to the correctness and consistency of designs.

Example 1. The tunnel diode oscillator circuit in Figure 5.a, which has been used by many researchers (e.g.,[18, 19]) as a benchmark, will be used as an example throughout the paper to demonstrate each step of our methodology. The tunnel diode exploit a phenomenon called resonant tunneling due to its negative resistance characteristic at very low forward bias voltages. This means that for some range of voltages, the current

decreases with increasing voltage. This characteristic makes the tunnel diode useful as an oscillator.

This introductory example is provided as a frame of reference for the analog designer. Figure 5.b is a HSPICE representation of the tunnel diode in Figure 5.a. Each node is represented by a number and each component is represent by an alphanumeric name. It is a one to one mapping of the circuit diagram to the HSPICE code. An external model file provides the behaviour of the tunnel diode.

Example 2. The transformation from a circuit diagram to bond graph is comparable to the previous HSPICE example. 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 6. This is preformed according to the bond graphs rules described earlier.



(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) HSPICE Code

Fig. 5. Tunnel Diode Oscillator

Simplification. There exists two levels of simplification that can be performed on bond graphs. Firstly, there are equivalence rules for the junction object. 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(Figure 7).

The second level of simplification is analogous to the concept of combining many resistances into one equivalent resistance. The similar idea can also be applied in the physical domain to two rigidly connected bodies that can be combined into a single mass [17]. 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 DAEs that are extracted from the reduced bond graph model. By using a simpler model, the number of states can be reduced, allowing for a less complex verification problem.

Example 3. Simplifications of the bond graph in Figure 6 can be made. The removal of the bonds that are connected to ground can be removed since the voltage at those nodes is zero, indicating that the power flow is zero. Since the flows at 1 junctions are equal, 1

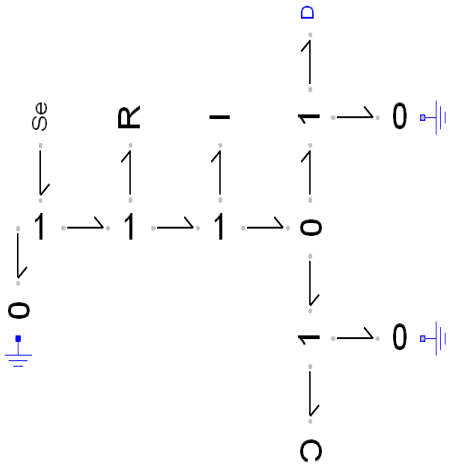


Fig. 6. Tunnel Diode Initial Bond Graph

junctions in series can be merged together. The resulting simplified bond graph is given in Figure 8.a.

As a final step to the simplification process, any junction that has only two bonds connected to it can be removed since no power that flows through a two port junction can divert to another component as shown in Figure 8.b.

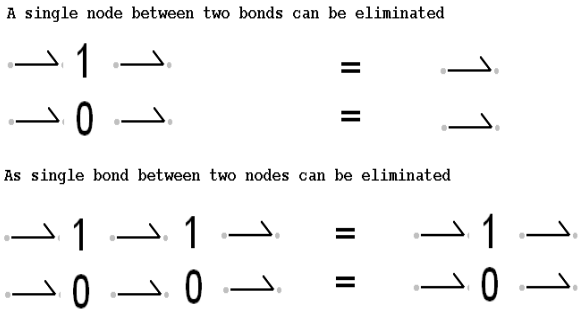
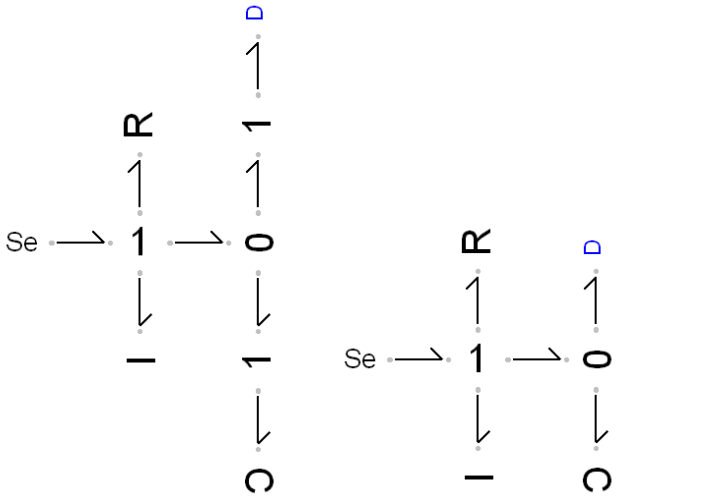


Fig. 7. Simplification Rules for Bond Graph Junctions

Example 4. The next step in the conversion process is to add a causality stroke to each bond. The stroke determines at which point the flow variable is to be calculated. Causality can be computed automatically, but it is recommended to use causal bonds since they can help in analyzing the model when designing larger systems. Certain bond graph elements only have a set number of causalities that can be assigned. For instance, at 0 junctions the efforts are equal which indicates that only one causality stroke is assigned



(a) First Simplification Pass (b) Second Simplification Pass

Fig. 8. Tunnel Diode Bong Graph Simplifications

because of the single flow equation defining the junction. Similarly, at 1 junctions the flows are equal, which indicates that there should be only one bond without a causality stroke because of the single effort equation defining the junction. For capacitors and inductors causality is chosen so that differential equations are generated. The stroke is away from capacitors and towards for inductors. The final bond graph is defined as shown in Figure 9.

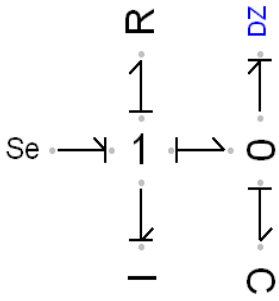


Fig. 9. Tunnel Diode Causal Simplified Bond Graph

Different levels of abstraction for verification. Bond graphs have been characterized as “the most basic graphical modelling paradigm that is fully objected-oriented” [4]. It follows that the concept of encapsulation can be applied to bond graphs to model

systems at different levels of complexity. The benefit being that there is no need for single complex equation layer to define a system.

The BondLib library developed by Cellier *et al.* [4] demonstrates the benefit of object oriented modelling with bond graphs. The transistor models for BJTs and MOSFETS are true HSPICE models that can be set to different levels of complexity [4]. At each level, parasitics, current leakages and non-ideal effects can be added to the model by specifying the correct parameter. The parameters are available to the modeler to dynamically alter the bond graph level. For instance, the difference between the MOSFET level 0 and 1 is that the capacitances between the source, drain, gate and body are set to zero. This allows for a simplification that can be used to verify system properties based on specific device configurations.

3.2 Analog Computation Modelling

Once the bond graph is built, the set of system equations can be extracted and simplified. In the current project, we use Mathematica simplification functionalities in order to remove redundant equation through rewriting techniques. The final system of equations are the computational model on which we apply the verification. In general, the analog design computational model can be described as follows:

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.* $\mathcal{X}_0 \subseteq \mathcal{X}$ *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 (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 consisting 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 5. The tunnel diode bond graph is constructed in the Dymola environment. The BondLib library contains graphical modules for bonds and nodes. The resistors, inductors and capacitors all use HSPICE based models contained in *ModelicaSpice* which is itself a sub-library of BondLib. Dymola then converts the bond graph to Modelica code. Index reduction, function tearing and further algorithms then automatically transform the DAEs to ODEs from the Modelica code. Since Dymola uses dummy variables to aid in the conversion from DAEs to ODES many extra variables are present in the final output. By constructing simplification rules in Mathematica, the system of ODEs can be simplified. The output of Dymola is shown in Figure 10.

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 5). 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 $I_d(V_C)$ describes the non-linear tunnel diode behaviour.

```
// Dummy Variables
  resistor.v := resistor.R*inductor.i;
  resistor.n.v := constantVoltage.V-resistor.v;
  inductor.v := resistor.n.v+capacitor.v;
  tunnelDiode.p.i :=
   capacitor.v^3-1.5*capacitor.v^2+0.6*capacitor.v;
  capacitor.p.i :=  -(inductor.i+tunnelDiode.p.i);

// Symbolic solution
  /*  Original equations
  inductor.L*der(inductor.i) = inductor.v;
  -capacitor.C*der(capacitor.v) =  -capacitor.p.i;
  */

  der(inductor.i) := inductor.v/inductor.L;
  der(capacitor.v) := capacitor.p.i/capacitor.C;
```

Fig. 10. System of ODEs generated by Dymola

4 Analog Design Verification

The verification proposed in this paper is based on combining predicate abstraction and constraint solvers. In predicate abstraction, the analog state space is turned into a Boolean state space over which symbolic model checking is applied. This method is then suitable for the verification of designs with a higher dimension state space. This comes at the cost of the precision of the abstraction based verification employed. To overcome such problems, refinement procedures are often associated with verification.

On the other hand, constraint solving techniques are applied directly on the continuous state space which provide more accurate verification results. However, constraint based methods are more computationally expensive and are usually limited to lower dimension systems. To take advantage of both techniques, we provide two verification settings combining predicate abstraction and constraint solving.

Approach 1: Enhancing Constraints based Verification using Predicates. In the approach shown in Figure 11, we strengthen the constraint based verification with predicates that act as constraints on the state space. This is technically practical as the addition of useful constraints usually limits the state space exploration and providing means for pruning unreachable states, which reduce the computational cost and even in some cases make some verification problem more tractable.

In this approach, HybridSal is applied 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 (e.g., using Mathematica). Verification is then applied on the newly generated abstract model.

Basically, HSolver has an internal abstraction refinement procedure. However, due to overapproximation the refinement does not terminate unless there is a bound on it which is specified when running the tool. When the bound is reached but verification does not terminate, then we get a non conclusive answer and a set of intervals that violates the property. Refinement can be done by increasing the bound or choosing tighter constraints for the abstract states. Adding constraints is done by generating more predicates using HybridSal.

Approach 2: Predicate Abstraction based Verification. The second approach illustrated in Figure 12 is an abstract model checking approach extended with a counterexample validation and refinement procedure. 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 to refine the abstract reachable states.

In the approach presented in this paper, 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. The approach shown in Figure 12 requires as input the representation of the analog circuit as system of ODEs, the initial conditions and the temporal property of interest. 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. The predicates specifying the counterexample are turned into constraints that are provided to HSolver, along with the property and the system of ODEs. HSolver tries to validate the property only in the

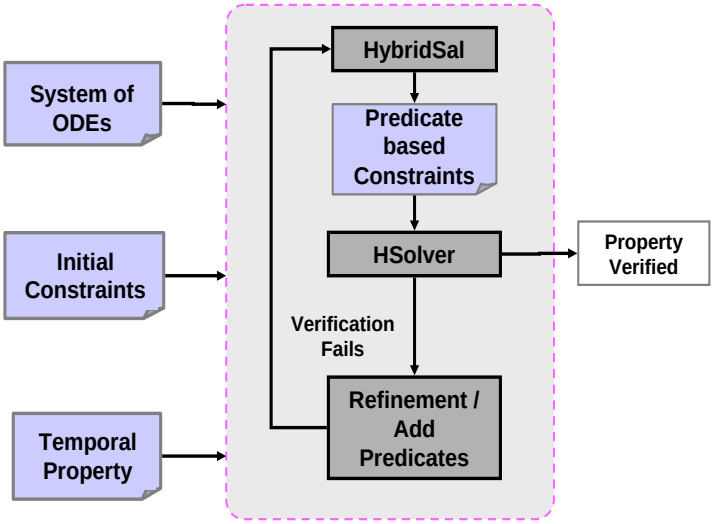


Fig. 11. Constraints based Verification

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. Unfortunately, there is no guarantee that a spurious counterexample can be refuted and the procedure might not terminate. Technically, this happens if the approximation is too loose and not precise enough, which results in behaviour that is impossible in reality. To our knowledge no efficient solution exists for such problems, however, other practical counterexample validation techniques have been proposed in [7].

4.1 Constraint Based Verification

Constraint solving is the study of systems based on constraints (relation between the variables of the system). The idea of constraint solving is to solve problems by stating constraints about the problem area and consequently, finding solutions satisfying all the constraints. Two categories of constraint solvers are identified [37]:

- Satisfiability constraint solvers: When a constraint solver pronounces the existence of a solution, the constraints are guaranteed to have a numerical solution. In addition, if a solution is produced, then it is guaranteed that this solution satisfies the constraints. One such solver is Rsolver [32] and Mathematica Capabilities like *Reduce* and *FindInstance* [36].
- Unsatisfiability constraint solvers: If a constraint solver pronounces the infeasibility of the input constraints, then this result is sound. If no solution is produced, then this means that the system is unfeasible. Realpaver [15] is an example of this category.

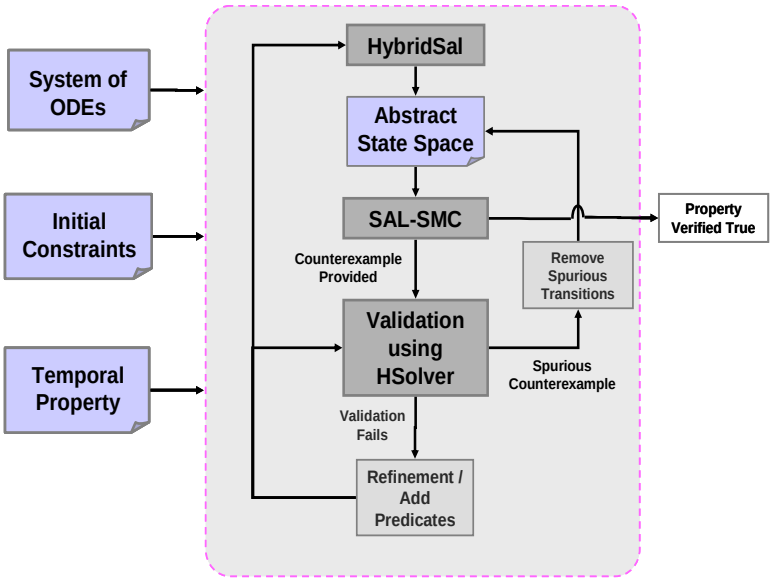


Fig. 12. Predicate Abstraction based Verification

In constraint solving techniques, the uncertainty of numerical variables are over-approximated using intervals of real numbers to make safe decisions possible. Interval based arithmetics techniques provide efficient and safe methods for solving continuous constraint satisfaction problems where real variables are constrained by equalities and inequalities. The soundness is inherited form the inclusion property of interval arithmetics [28].

Theorem 1. *Let* $f : \mathbb{R}^n \rightarrow \mathbb{R}$ *be a continuous function, then* $F : \mathbb{I}^n \rightarrow \mathbb{I}$ *is an interval extension of* f *if* $\{f(x_1,\ldots,x_n)|x_1 \in X_1,\ldots,x_n \in X_n\} \subseteq F(X_1,\ldots,X_n)$*, where* $\mathbb{I}$ *is the interval domain.*

In the context of differential equations, constraint based approaches provide safe methods for solving initial value problems which verify the existence of unique solutions and produce guaranteed bounds for the true trajectory. In this paper, for the verification purpose, we use HSolver [32]. The basic idea behind the tool is to decompose the state space into hyperboxes. Interval arithmetic is then used to check the flow on the boundary between neighboring boxes. This is done via an abstraction refinement framework in order to achieve precise results.

We make use of constraint based verification as a verification engine for two purposes. First, it is used as a verification engine to verify safety properties. Second, since more complex properties need to be verified using predicate abstraction the constraint solver tool HSolver is used to refine the abstract model by refuting invalid transitions between abstract states.

4.2 Abstraction Based Verification

The common concept between safety verification based on constraint solving and model checking based on predicate abstraction is the requirement of overapproximation for the reachable states.

Given the analog model transition system $\mathcal{T}_{\mathcal{A}}$ representing the analog behaviour and a property φ expressed in $\forall$CTL. 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 can be stated as follows [7]:

Theorem 2. *Suppose $\mathcal{T}_{\Psi}$ is an abstract model of $\mathcal{T}_{\mathcal{A}}$, then for all $\forall CTL$ 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 can be applied [7].

4.3 Safety Verification

In general, for analog designs, the kind of properties we are interested to verify can be for example: *The system behaviour will be the same for the set of initial condition*, or, *For which set of parameters values, the circuit oscillates?* Many of these properties can be stated as safety properties. Suppose that we need to verify a safety property specified in temporal logic as $\forall \mathbf{G} \neg p$ (which means always constraint p will be satisfied), we build the dual property $\exists \Diamond \neg p$ (which means that there is an execution falsifying the constraint p) and apply feasibility checking on dual property within the invariant regions of interest. If the constraints system is satisfiable, we conclude that the property might be violated otherwise, the property is verified.

Example 6. Consider the tunnel diode circuit with the set of parameters $\{C = 1000e^{-12}$, $L = 1e^{-6}$, $G = 2000e^{-3}$, $Vin = 0.3\}$ and the initial values $\{V_C = 0.131V$, $I_L = 0.055A\}$. We verify that the preceding combination of parameters and initial conditions do not produce oscillatory behaviour. The behaviour in question is stated as the safety property $\mathbf{G}v \leq 0.6$. The validation of the property ensures the non-existence of oscillation.

We apply the second verification for the verification of the tunnel diode oscillator. Once the simplified system of ODEs has been extracted as shown in Example 5, they can be used to form a hybrid system definition in the HybridSal modelling language, as in Figure 13. In general, the hybrid system definition has both discrete and continuous sections that allow the entire behaviour to be modeled.

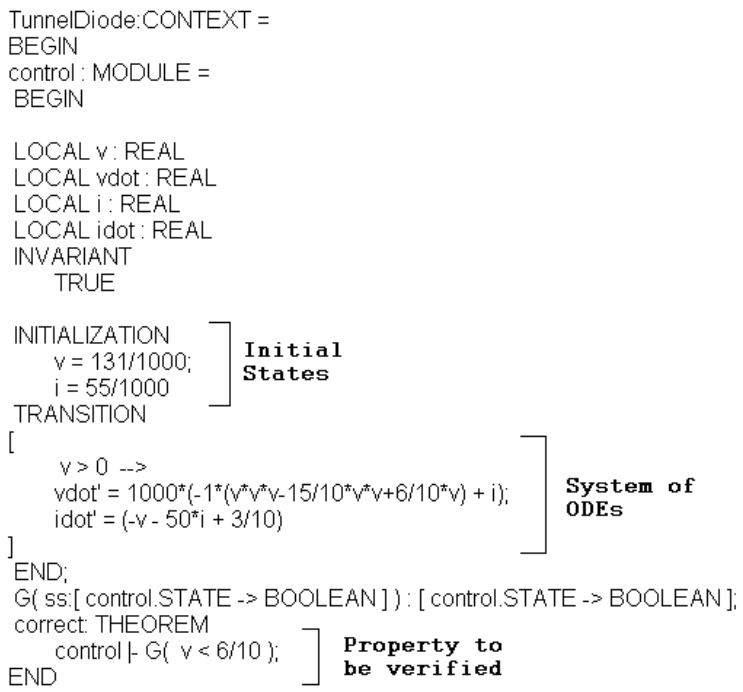


Fig. 13. HybridSal Tunnel Diode Description

4.4 Predicate Abstraction

Predicate abstraction is a method where the set of abstract states is encoded by a set of Boolean variables representing each a concrete predicate. Based on [1], 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 $\psi : \mathbb{R}^d \to \mathbb{B}$, 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 can define the abstract behaviour of the analog circuit as a transition system that overapproximates that behaviour.

Definition 3. 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* $\{b \rightsquigarrow b' | \exists x \in \Upsilon_\Psi(b), t \in \mathbb{R}^+ : x' = \Phi_x(t) \in \Upsilon_\Psi(b') \wedge x \to x'\}$*, where the concretization function:* $\Upsilon_\Psi : \mathbb{B}^n \to 2^{\mathbb{R}^d}$ *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 [34, 40].

For example, a set of predicates can be constructed using the notion of *critical forms*, which are special functions along them, the vector field direction is either vertical or horizontal. In between these forms, there can be neither vertical nor horizontal vectors. In a region (abstract state) determined by the critical forms, all vectors follow one direction. These predicates can be obtained easily by setting $\dot{\mathbf{x}} = 0$. A generalization of critical forms is the concept of *isoclines*. Isoclines are functions over which the system trajectories have a constant slope. A predicate π is an isocline of ODEs system if and only if $\exists a_i \in \mathbb{R}$ with $i = 1, \ldots d$ such that $\Sigma_{i=1}^{d} a_i \mathcal{P}_i(\mathbf{x})|_{\pi} = 0$.

Isocline and critical forms provide qualitative information about the system behaviour. Hence, such information can be used in refuting certain behaviour that is shown unreachable. For instance, by knowing different constants a_i, we deduce the direction of the flow crossing the isoclines and therefore we decide how to build transitions between abstract states. Finding different isocline predicates within an invariant region can be achieved by solving constraints on the parameters of predefined forms of an isocline predicate.

Other methods for finding useful predicates were developed in [34], where the authors proposed a way to extract predicates from polynomial ODEs by looking at higher derivatives. If $p \in P$, then add $\dot{p}$, the derivative (with respect to time) of p, to the set P unless $\dot{p}$ is a constant or a constant factor multiple of some existing polynomial in P.

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 attractive 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.

Constructing the Transition Relation The other issue in predicate abstraction is the identification of the possible transitions. In general, information from the solution of the ODEs is required to describe transitions between abstract states. In practice, each abstract transition is initialized to the trivial relation, relating all states and then step by step refined by eliminating unfeasible transitions. This guarantees that any intermediate result represents an abstraction and the refinement can be stopped at any point of time. The generated abstract state transition system can then be verified using a symbolic model checker SAL-SMC.

Several complimentary approaches can be used in order to enhance the precision of the transition system. The simplest rule to use is the *Hamming distance rule* [34]. The Hamming distance (HD) is the number of predicates for which the corresponding valuations are different in different abstract states. For instance, the Hamming distance between state $s_1 := (p_1 = 1 \wedge p_2 = 0 \wedge p_3 = 1 \wedge p_4 = 1)$ and state $s_2 := (p_1 = 1 \wedge p_2 = 0 \wedge p_3 = 0 \wedge p_4 = 1)$ is 1, written $HD(s_1, s_2) = 1$. Given two abstract states s_1 and s_2,

we say that a transition can exist between two abstract states only if $HD(s_1, s_2) = 1$. More advanced methods to refine the transition relation between abstract states where developed in the literature [34, 1, 40].

Example 7. Given the tunnel diode circuit model described in Example 6, the HybridSal tool generates the discrete abstract model illustrated in Figure 14. This abstract model is model 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 Figure 15). Specifically the property states that the predicate $g1$ must always be negative. However, the generated counterexample demonstrates a path to where the $g1$ predicate is zero. The goal is to check whether the counterexample is spurious or not.

```
%% Generated Predicates
%% g2 --> v
%% g1 --> v - 3/5
%% g0 --> -1*v^3 + 3/2*v^2 - 3/5*v + i
 .....
  control: MODULE = BEGIN
    GLOBAL
    g0: SIGN
    GLOBAL
    g1: SIGN
    GLOBAL
    g2: SIGN
    INITIALIZATION
    g2 = pos; g1 = neg; g0 = neg
    TRANSITION
    [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);
END
```

Fig. 14. SAL Code for the Abstract Model of the Tunnel Diode Circuit

Example 8. 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 the HSolver tool as shown in Figure 16, we can perform a more precise examination of the reachable states. If it is determined that the counterexample is never reached then the spurious transitions can be removed from the abstract model.

```
  INVALID, building counterexample...
  verification time: 0.03 secs
Counterexample:
=========================
Path
=========================
Step 0:
--- System Variables (assignments) ---
g0 = neg
g1 = neg
g2 = pos
-------------------------
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 property
g2 = pos
```

Fig. 15. SAL-SMC Generated Counterexample for SAL Code in Figure 14

In this case, the path of the counterexample produced by the SAL-SMC tool is never reached indicating that the counterexample is spurious. Therefore, we remove from the SAL code in Figure 14 all transitions from states where predicate $g1 = neg$ holds to states where $g1 = zero$ holds. This refinement is done by applying cone of influence [8] on the code in Figure 14. 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 code in Figure 16 are based only on the $g0$ and $g1$ predicates. The verification on the refined SAL code using SAL-SMC in this case succeeds, which means that no oscillation will occur.

5 Experimentation Results

We have applied the proposed methodology on different analog examples in order to verify certain properties related to functional behaviours. We will present below oscillator circuits, namely the Chua circuit and Colpitts oscillators.

5.1 Chua Circuit

We use the first verification approach described in Section 4 in order to verify the Circuit shown in Figure 17.a. This circuit was designed and implemented by Chua [6] to

```
VARIABLES [v,i]
MODES [m1,m2,m3,m4]
STATESPACE
m1[[-0.5,1.2],[-0.5,0.2]]
m2[[-0.5,1.2],[-0.5,0.2]]
m3[[-0.5,1.2],[-0.5,0.2]]
m4[[-0.5,1.2],[-0.5,0.2]]
INITIAL
  m1{v=0.131/\i=0.055}
FLOW
m1{v_d=1000*(-(v*v*v-1.5*v*v+0.6*v) + i)}{i_d=(-v - 50*i + 0.3)}
m2{v_d=1000*(-(v*v*v-1.5*v*v+0.6*v) + i)}{i_d=(-v - 50*i + 0.3)}
m3{v_d=1000*(-(v*v*v-1.5*v*v+0.6*v) + i)}{i_d=(-v - 50*i + 0.3)}
m4{v_d=1000*(-(v*v*v-1.5*v*v+0.6*v) + i)}{i_d=(-v - 50*i + 0.3)}
JUMP
m1->m2{[v*v*v+1.5*v*v-0.6*v+i=0]/\[i'=i/\v'=v]}
m2->m3{[v*v*v+1.5*v*v-0.6*v+i>0]/\[i'=i/\v'=v]}   Predicates
m3->m4{[v-0.6=0]/\[i'=i/\v'=v]}
UNSAFE
  m4{v>0.6}   Property
```

Fig. 16. HSolver Code for the Counterexample Validation of Figure 15

demonstrate the behaviour of chaos. This is illustrated with simulation as shown in Figure 17.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 in to three piecewise linear regions [22].

$$I = if\ (V < -V_e)\ then\ G_b(V+V_e) - G_aV_e\ else\ if\ (V > V_e)\ then\ G_b*(V-V_e) + G_a*V_e\ else\ G_a*V$$

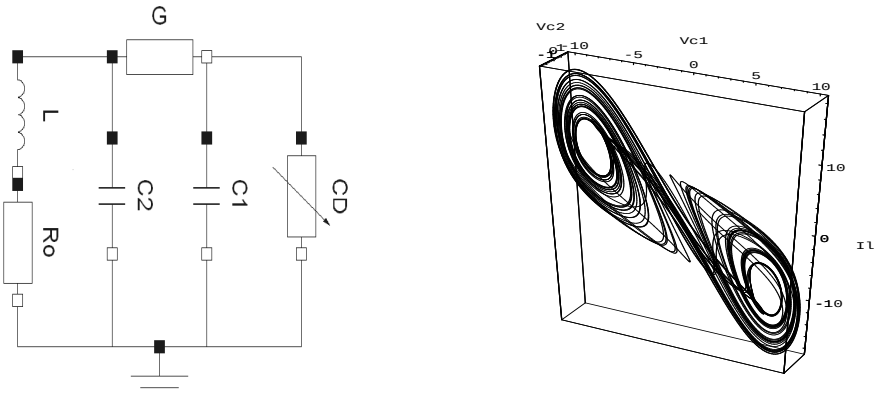
with $G_a = -0.757576$, $G_b = -0.409091$ and $V_e = 1$.

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 $\forall[G-6 \leq V_{c_1} \leq 6]$ on the voltage across the capacitor C_1 shown in Figure 17.

In order to apply the proposed verification approach, the circuit diagram in Figure 17.a is transformed to the corresponding bond graph. Simplification rules are then applied to obtain a reduced bond graph as shown in Figure 18. 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 tool. 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 code in Figure 19. 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.

5.2 MOS Colpitts Oscillator

The circuit diagram for a MOS transistor based circuit is shown in Figure 20.a. For the correct choice of component values the circuit will oscillate. This is due to the bias



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

Fig. 17. Chua Circuit

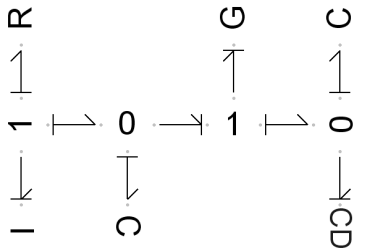


Fig. 18. Chua Circuit Bond Graph

current and negative resistance of the passive tank. The property that was analyzed was whether for the given parameters and initial conditions will the circuit die out (Not oscillate) as shown in Figure 20.b. [1] The simplified equations are described as follows:

$$Vc_1' := \frac{1.2-(Vc_1+Vc_2)}{R*C} + \frac{Il}{C} - \frac{I_{ds}}{C}, Vc_2' := \frac{-Iss}{C} + \frac{1.2-(Vc_1+Vc_2)}{R*C} + \frac{Il}{C} \text{ and } Il' := \frac{1.2-(Vc_1+Vc_2)}{L}$$

with

$$I_{ds} := \begin{cases} 0 & Vc_2 > 0.3 \\ kp * \frac{w}{l} * ((0.3-Vc_2) * (Vc_1) - 0.5 * (Vc_1)^2) & Vc_1 + Vc_2 < 0.3 \\ \frac{kp}{2} * \frac{w}{l} * (0.3-Vc_2)^2 & Vc_1 + Vc_2 \geq 0.3 \end{cases}$$

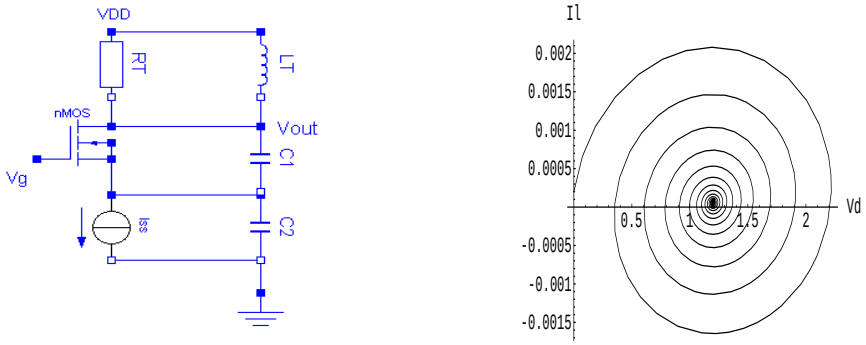
Oscillation will not occurs if the current cannot exceed a certain bound. More precisely, if verified to *true*, the property $\forall \mathbf{G} I_l > -0.004 \wedge I_l < 0.004$ is a necessary condition that implies no oscillation will occur. The system equations, the property of interest along with the required constraints were then translated into the HSolver code. The state

[1] The bond graph transformation of the circuit diagram will not be presented here to save space, details of the verification can be found in a technical report [11].

VARIABLES $[V_{c1}, V_{c2}, I_l]$
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]]
Environment Constraints
INITIAL
m3$\{V_{c1} = 4 \wedge V_{c2} = 0 \wedge I_l = 0\}$
FLOW
m1 $\{V_{c1}\_d = (0.565 * (V_{c2} - V_{c1}) + 0.409091 * V_{c1} - 0.757576) * 0.1\}$
$\{V_{c2}\_d = -(0.565 * (V_{c2} - V_{c1}) + I_l) * 0.01\}$
$\{I_l\_d = (V_{c1} - 12.5 * 10^{-3} * I_l) * 0.0555\}$
m2 $\{V_{c1}\_d = (0.565 * (V_{c2} - V_{c1}) + 0.757576 * V_{c1}) * 0.11\}$
$\{V_{c2}\_d = -(0.565 * (V_{c2} - V_{c1}) + I_l) * 0.01\}$
$\{I_l\_d = (V_{c2} - 0.0125 * I_l) * 0.0555\}$
m3 $\{V_{c1}\_d = (0.565 * (V_{c2} - V_{c1}) - 0.409091 * (V_{c1} - 1) + 0.757576) * 0.1\}$
$\{V_{c2}\_d = -(0.565 * (V_{c2} - V_{c1}) + I_l) * 0.01\}$
$\{I_l\_d = (V_{c2} - 0.0125 * I_l) * 0.0555\}$
Simplified ODEs
JUMP
$m1 -> m2\{[V_{c1} > -1] \wedge [V'_{c1} = V_{c1} \wedge V'_{c2} = V_{c2} \wedge I'_l = I_l]\}$
$m2 -> m1\{[V_{c1} <= -1] \wedge [V'_{c1} = V_{c1} \wedge V'_{c2} = V_{c2} \wedge I'_l = I_l]\}$
$m2 -> m3\{[V_{c1} > 1] \wedge [V'_{c1} = V_{c1} \wedge V'_{c2} = V_{c2} \wedge I'_l = I_l]\}$
$m3 -> m2\{[V_{c1} <= 1] \wedge [V'_{c1} = V_{c1} \wedge V'_{c2} = V_{c2} \wedge I'_l = I_l]\}$
Transition Rules and Predicates
UNSAFE
m1$\{V_{c1} < -6\}$
m3$\{V_{c1} > 6\}$
Properties

Fig. 19. Chua Circuit HSolver Code

space was split into three regions because of the different states of the MOSFET transistor within the circuit. The property was verified to be true indicating no oscillation.



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

Fig. 20. MOS Colpitts Circuit

5.3 BJT Colpitts Oscillator

In order to understand the circuit behaviour, it is important to identify the different modes of operations of the transistor when connected with other circuit components. Circuit analysis is usually done by hand as simulation data is not conclusive. We can apply constraint solving to ensure that the transistor will never go into a specific mode of operation.

Consider the BJT based Colpitts oscillator shown in Figure 21. Correct functionality ensures that the BJT will never go to saturation region [23]. In fact, the BJT will either be in the Cut-off mode or Forward active mode. 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 $Vth = 0.75$ as shown in Figure 22. For instance, the property that no transition can occur from Forward active (m_1) to Saturation (m_3), can be validated by proving that $\forall \mathbf{G}\ V_{C_2} < 0.75 \wedge V_{C_1} + V_{C_2} < 0$ is False, where V_{C_1} and V_{C_2} are voltages across the capacitors C_1 and C_2.

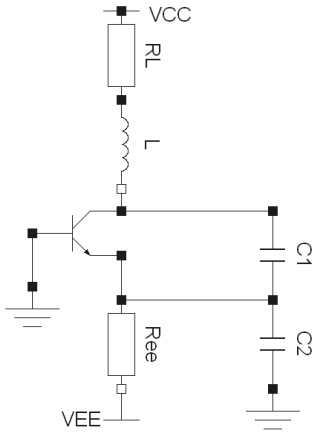


Fig. 21. BJT Colpitts Circuit

6 Conclusion

In this paper, we proposed a novel approach for the verification of analog circuits. 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. The major contributions are the following:

– By using bond graphs as a framework to represent circuits, models can be constructed automatically at several levels of abstraction. This can reduce the complexity of the system equations as well as simplify complex behaviour.

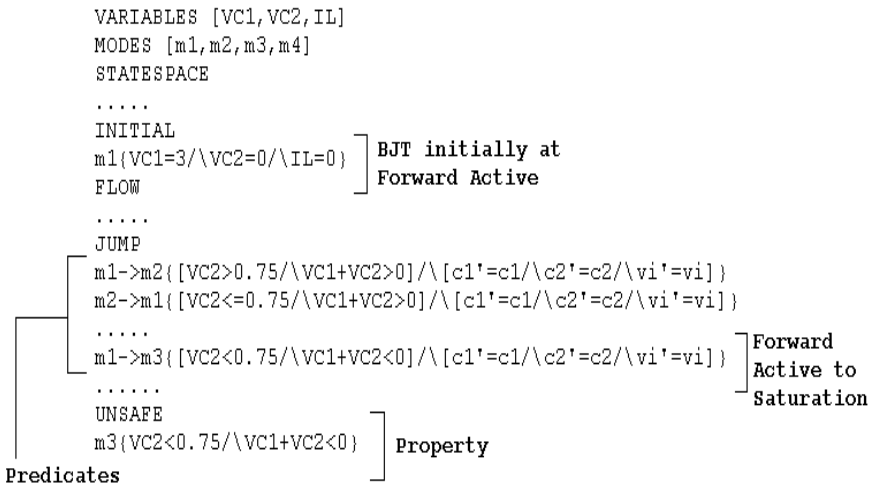


Fig. 22. HSolver Code of the BJT Colpitts Circuit

– For verification purposes, we proposed to combine predicate abstraction and constraint solving into two alternative methodologies. Our method does not require explicit representation of state space and relies on functions that prove or disapprove circuit properties.

Future Work. Main future directions include the extension of the proposed approach to analog and mixed signal designs. For this aim, we plan to represent the design using switched bond graph rather than the conventional one presented in this paper. We also need to explore more case studies and apply the verification on more interesting properties.

References

1. R. Alur, T. Dang, F. Ivancic. Reachability Analysis Via Predicate Abstraction. In Hybrid Systems: Computation and Control, LNCS 2289, pp. 35-48. Springer, 2002.
2. P.C. Breedveld. Modeling And Simulation Of Dynamic Systems Using Bond Graphs. In Control Systems, Robotics and Automation, Encyclopedia of Life Support Systems, Eolss Publishers, pp. 1-36, 2004.
3. F. Broenink. Introduction to Physical Systems Modeling with Bond Graphs, SiE Whitebook on Simulation Methodologies, 1999.
4. F.E. Cellier, C. Clauss and A. Urquia. Electronic Circuit Modeling and Simulation in Modelica, In Eurosim Congress on Modelling and Simulation, Vol. 2, pp. 1-10, 2007.
5. F.E. Cellier and A. Nebot. The Modelica bond graph library. In Modelica Conference, pp. 57-65, 2005.
6. L.O. Chua. Chuas Circuit : An Overview Ten Years Later, Journal of Circuits, Systems and Computers, 4:117-159, World Scientific, 1994.

7. E. Clarke, A. Fehnker, Z. Han, B.H. Krogh, O. Stursberg, M. Theobald. Verification of Hybrid Systems based on Counterexample-Guided Abstraction Refinement. In Tools and Algorithms for the Construction and Analysis of Systems, LNCS 2619, pp. 192-207, Springer, 2003.
8. E.M. Clarke, O. Grumberg, and D.A. Peled. Model Checking. MIT Press, 2000.
9. T. R. Dastidar, P. P. Chakrabarti, Verification System for Transient Response of Analog Circuits Using Model Checking. In IEEE International Conference on VLSI, pp.195-200, 2005.
10. T. Dang, A. Donze, O. Maler, Verification of Analog and Mixed-signal Circuits using Hybrid System Techniques. In Formal Methods in Computer-Aided Design, LNCS 3312, pp.14-17, Springer, 2004.
11. W. Denman, M. Zaki and S. Tahar. Analog Formal Verification Via Bond Graphs and Constraint Solving. Technical Report, ECE Dept., Concordia University, Montreal, Quebec, Canada, April 2008. http://hvg.ece.concordia.ca/Publications/TECH_REP/ AMS_BG_TR08
12. H. Elmqvist. Dymola - Dynamic Modeling Language, User's Manual. Lund: Dynasim AB, 1994.
13. G. Frehse, B. H. Krogh, R. A. Rutenbar. Verifying Analog Oscillator Circuits Using Forward/Backward Abstraction Refinement. In IEEE/ACM Design, Automation and Test in Europe, pp. 257-262, 2006.
14. M. R. Greenstreet, I. Mitchell: Reachability Analysis Using Polygonal Projections. In Hybrid System: Computation and Control, LNCS 1569, pp.103-116, Springer, 1999.
15. L. Granvilliers. On the Combination of Interval Constraint Solvers. Reliable Computing, 7(6):467-483, Springer, 2001
16. S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In Computer Aided Verification, LNCS 1254, pp. 72-83. Springer, 1997.
17. P.J. Gawthrop, G.P. Bevan. Bond-graph modeling. In IEEE Control Systems Magazine, 27(2):24-45, 2007.
18. S. Gupta, B.H. Krogh, R.A. Rutenbar: Towards Formal Verification of Analog Designs, In IEEE/ACM Conference on Computer Aided Design, pp. 210-217, 2004.
19. W. Hartong, K. Klausen, L. Hedrich. Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking, Advanced Formal Verification, Kluwer: pp. 205-245, 2004.
20. M. Jirstrand, J. Gunnarsson and P. Fritzson. MathModelica - A New Modeling and Simulation Environment for Mathematica. In the International Mathematica Symposium, 1999
21. D. Karnopp, R. Rosenberg. System Dynamics: a Unified Approach, Wiley, 1975.
22. M.P. Kennedy. Three Steps to Chaos - Part I: Evolution. IEEE Transactions on Circuits and Systems *I*, 40(10):640-656, 1993.
23. M.P. Kennedy. Chaos in the Colpitts Oscillator, IEEE Transactions on Circuits and Systems *I*, 41:771-74, 1994.
24. R.P. Kurshan and K.L. McMillan. Analysis of Digital Circuits Through Symbolic Reduction. IEEE Transactions on Computer-Aided Design 10:1350-1371, 1991.
25. S. Little, D. Walter, N. Seegmiller, C. Myers and T. Yoneda. Verification of Analog and Mixed-Signal Circuits Using Timed Hybrid Petri Nets. In Automated Technology for Verification and Analysis, LNCS 3299, pp. 426-440, Springer, 2004.

26. S. Little, D. Walter, K. Jones, C. J. Myers: Analog/Mixed-Signal Circuit Verification Using Models Generated from Simulation Traces. In Automated Technology for Verification and Analysis, pp. 114-128, LNCS 4762, Springer, 2007
27. T. Maehne, A. Vachoux. Proposal for a Bond Graph Based Model of Computation in SystemC-AMS. In Languages for Formal Specification and Verification, Forum on Specification & Design Languages, 2007.
28. R. E. Moore. Methods and Applications of Interval Analysis, Society for Industrial & Applied Mathematics, 1979.
29. S.E. Mattsson, H. Olsson, H. Elmqvist. Dynamic Selection of States in Dymola. In Modelica Workshop, pp. 61-67, 2000.
30. L.M. de Moura, S. Owre, H. Rue, J.M. Rushby, N. Shankar, M. Sorea, A. Tiwari: SAL 2. In Computer Aided Verification, LNCS 3114, pp. 496-500, Springer, 2004.
31. S. Ratschan, Z. She. Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement. In Hybrid System: Computation and Control, LNCS 3414, pp. 573-589, Springer, 2005.
32. S. Ratschan. Continuous First-Order Constraint Satisfaction. In Artificial Intelligence, Automated Reasoning, and Symbolic Computation, LNCS 2385, pp. 181-195, Springer, 2002
33. J.-E. Stromberg, S. Nadjm-Tehrani, J. Top. Switched Bond Graphs as Front-end to Formal Verification of Hybrid Systems. In Verification and Control of Hybrid Systems, LNCS 1066, pp. 282-293, Springer, 1996.
34. A. Tiwari and G. Khanna. Series of Abstractions for Hybrid Automata. In Hybrid Systems: Computation and Control, LNCS 2289, pp. 465-478, Springer, 2002.
35. J. Vlach, K. Singhal. Computer Methods for Circuit Analysis and Design. Kluver, 2003.
36. S. Wolfram. Mathematica: A System for Doing Mathematics by Computer. Addison Wesley Longman Publishing, 1991.
37. S. Xia, B. Divito, C. Munoz, Toward Automated Test Generation for Engineering Applications, In IEEE/ACM International Conference on Automated Software Engineering, pp. 283-286, 2005
38. Yousri El Fattah. Constraint logic programming for structure-based reasoning about dynamic physical systems. Artificial Intelligence in Engineering, 1:253-264, 1996.
39. M. Zaki, G. Al Sammane, S. Tahar, and G. Bois. Combining Symbolic Simulation and Interval Arithmetic for the Verification of AMS Designs. In IEEE International Conference on Formal Methods in Computer-Aided Design, pp. 207-215, 2007.
40. M. Zaki, S. Tahar, and G. Bois: Qualitative Abstraction based Verification for Analog Circuits; Revue des Nouvelles Technologies de l'information, 4:147-158, RNTI-SM-1, Edition Cepadues, 2007.
41. M.H. Zaki, S. Tahar, and G. Bois: Formal Verification of Analog and Mixed Signal Designs : A Survey. Microelectronics Journal, Elsevier. In Print.