Towards the Automated Modelling and Formal Verification of Analog Designs

William Denman

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

April 2009

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **William Denman**

Entitled: **Towards the Automated Modelling and Formal Verification of Analog Designs**

and submitted in partial fulfilment of the requirements for the degree of

Master of Applied Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Dr. Dongyu Qiu

_____ Dr. Anjali Awasthi

_____ Dr. Glenn Cowan

_____ Dr. Sofiène Tahar

Approved by _____

Chair of the ECE Department

_____ 2009 _____

Dean of Engineering

ABSTRACT

Towards the Automated Modelling and Formal Verification of Analog Designs

William Denman

The verification of analog circuits remains a very time consuming and expensive part of the design process. Complete simulation of the state space is not possible; a line is drawn by the designer when it is deemed that enough sets of inputs and outputs have been covered and therefore the circuit is "verified". Unfortunately, bugs could still exist and for safety critical applications this is not acceptable. As well, a bug in the design could lead to costly recalls and a loss of revenue. Formal methods, which use mathematical logic to prove correctness of a design have been developed. However, available techniques for the formal verification of analog circuits are plagued by inaccuracies and a high level of user effort and interaction. We propose in this thesis a complete methodology for the modelling and formal verification of analog circuits. Bond graphs, which are based on the flow of power, are used to automatically extract the circuit's system of Ordinary Differential Equations. Subsequently, two formal verification methods, one based on automated theorem proving with MetiTarski, the other on predicate abstraction based model checking with HybridSal, are then used to verify functional properties on the extracted models. The methodology proposed is mechanical in nature and can be made completely automated. We apply this modelling and verification methodology on a set of analog designs that exhibit complex non-linear behaviour.

ACKNOWLEDGEMENTS

There are two people who I attribute my success at Concordia. Firstly, my supervisor, Dr. Sofiène Tahar who gave me the chance of a lifetime to join his research group even though I had no research experience. His continued support goes above and beyond what is required by a thesis supervisor and has been critical to my success and enjoyment at the post graduate level. Secondly, I would not be at this point without the help of my colleague, Dr. Mohamed Zaki, who took me under his wing when I entered into the program. Not even four months into my research, he allowed me to collaborate with him as well as be first author on one of our submitted work. I know now that this is not the norm in academia, and for his trust and confidence in me I will forever be in debt.

I would like to thank Dr. Behzad Akbarpour for his great support of the MetiTarski tool. Thank you Naeem Abbasi for taking the time to go over the thesis multiple times.

I would like also thank the members of my thesis committee, Dr. Glenn Cowan, Dr. Anjali Awasthi and Dr. Dongyu Qiu for their comments and feedback on the thesis.

I would like to thank my girlfriend for her continued emotional support as well as keeping me alive during the thesis writing.

Lastly but not least, I would like to thank my parents for putting me on the path to success. Without them, I could have never made it this far.

To Mom, Dad, Teresa,
and the memory of my grandparents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AMS	Analog and Mixed-Signal
BJT	Bipolar Junction Trasistor
CTL	Computational Tree Logic
DAE	Differential Algebraic Equation
DC	Direct Current
EDA	Electronic Design Automation
FSM	Finite State Machine
IGBT	Insulated Gate Bipolar Transistor
KCL	Kirchoff Current Law
KVL	Kirchoff Voltage Law
LTL	Linear Temporal Logic
MOC	Model Of Computation
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
ODE	Ordinary Differential Equation
PVS	Property Verification System
PWL	Piecewise Linear
QEPCAD	Quantifier Elimination Procedure by Cylindrical Algebraic Decomposition
RCF	Real Closed Fields
SAL	Symbolic Analysis Laboratory
SAT	Boolean Satisfiability
SCAP	Sequential Causality Assignment Procedure
SMC	Symbolic Model Checking
SPICE	Simulation Program with Integrated Circuit Emphasis
SRI	Stanford Research Institute

TPTP	Thousands of Problems for Theorem Provers
VHDL	Very High Speed Integrated Circuits Hardware Design Language

Chapter 1

Introduction

1.1 Motivation

Embedded systems have become an important part of many devices that we use every day including mobile phones, television set-top boxes and digital cameras. They are also responsible for controlling systems that protect us on a daily basis. Devices such as traffic light, airplane landing gear and elevator break controllers are all implemented using embedded systems. These devices are increasingly becoming complex to design because of the necessary interaction with the physical world. Because of the unpredictable nature of this outside influence, the devices are required to operate over a high number of different modes that can be particularly difficult to determine, isolate and verify. For safety critical systems, where verification is required to ensure that an accident will not occur, this situation can be particularly problematic.

Beyond the problems of verifying the combinations of user input to a device, another critical problem facing the current generation of embedded systems are the effects arising from the reduction in fabrication size. Parasitics, current leakage and signal noise change the functionality of analog designs in unexpected ways. 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.

The Electronic Design Automation (EDA) industry has developed sophisticated tools to aid engineers in the design and verification of digital circuits. This has allowed digital designs to grow in size and complexity without putting a larger burden on the designers knowledge of the lower level functionality. For analog designers, there has not been the same amount of progress on their tools or methodologies. The design flow has remained essentially the same for the past twenty years. A schematic capture program is used to hand design abstract models, a netlist is extracted and then a circuit simulator is used to verify the design. This is repeated until the desired specifications are met [56]. For the moment, this methodology is adequate since there is ongoing work to make simulators faster and more efficient. Unfortunately, this cannot go on for ever. What happened with digital circuits in the late 1990's is starting to occur again with analog circuits. The complexity of some basic circuit elements is starting to overwhelm engineers and errors at the initial design stage are increasing [14].

Traditionally, simulation has been used to verify analog designs. Unfortunately, verification by simulation is inherently informal because the state space search (set of inputs and expected outputs) is incomplete due to the continuous range of parameter values. Even when using reduced-accuracy simulators, large circuit transistor-level simulations require days or even weeks to complete [14]. As a consequence, simulation methods lack the rigor to ensure the complete correctness of the design.

Before a circuit can be simulated, a test-bench must be constructed where a

set of input and expected outputs are chosen. In the best case scenario, the designer will understand the design perfectly and will choose the test points that represent the limit of operation of the design. But this is far from reality, the questions that the verifier will have trouble answering are "How do we choose the test set?" and "How do we know when we have covered enough of the state space?". One viable option is to use random points for simulation, but again there is no absolute way to know if enough test points have been analyzed to verify a design to a proper level of conformity.

To address the incomplete verification of designs via simulation, formal methods have been developed to increase this confidence level. Formal methods [42] are based around applying mathematical expressions and reasoning to prove correctness of a design. A formal specification is constructed and is used to verify a model using mathematical logics and formal reasoning. There are two main areas of formal verification: model checking and theorem proving. In model checking [18] there is an exhaustive search of the state space. For large designs that contain many variables, most model checking techniques fail to produce an answer because of what is commonly called a "State Space Explosion" [10]. This is where the amount of computer memory required to hold the state information is too large. In theorem proving [49], a complete proof is constructed by hand using a base set of axioms and conjectures. Incredibly powerful, it can be theoretically used to solve any logical problem. Unfortunately, great manual effort is required on part of the verifier to construct the proofs since the method is interactive and thus labour intensive.

With all of its advantages, formal verification seems like the ideal method to solve all possible verification problems. But in fact, they can only assure correctness of a design with relation to a formal specification. The final circuit can still fail because there is no guarantee that the formal specification is correct. As well, errors can also be produced because of defects in fabrication. Notice though, that these problems are also encountered with simulation. Therefore, formal verification

should ideally be combined with simulation methods to increase the confidence level of design beyond of what is presently capable by simulation alone.

To take advantage of what formal methods offer to the verification of analog circuits we address two broad goals:

- Appropriately model the analog circuit so that its continuous-time behaviour can be easily extracted for verification.
- Simplify and automate the methodology for the formal verification of analog designs. The limitations due to the state space explosion problem with model checking and the high level of user interactivity required by theorem proving must be addressed.

To address the first goal, we want to develop in this thesis a modelling method for the automatic extraction of the system of equations from an analog circuit that will aid the flow from the design to the verification stage. For the second goal, we want to develop formal verification techniques that address the stated limitations. In the next section, an overview of the related work in the domains of analog modelling and formal verification will be presented and then our methodology will be outlined in detail.

1.2 Related Work

1.2.1 Formal Modelling of Analog Circuits

One of the main challenges for the formal verification of analog designs, is the development of models that preserve the behaviour of real devices. One precise way to model analog behaviour is via mathematical systems of equations that are defined over a continuous state space. Nodal analysis techniques have been developed to extract equations from a circuit netlist. However, the resulting equations are in

general, very large and too complicated to be used for a behavioural analysis. For example, the authors of [57] relied on the symbolic analysis toolbox Analog Insydes [29] to obtain the system of equations necessary for verification. In their case, they relied on several iterations of algebraic simplifications that introduce errors in the final result.

Another approach used by Dastidar [22] generated a finite state machine (FSM) from a set of simulation traces to define a formal model. A similar approach was proposed by Little *et al.* in [44], where they generated from simulation data a hybrid petri net at the front-end to their verification program. The issue of concern with their method is that the model cannot be automatically produced, thresholds must still be defined making the specification only semi-formal in nature.

In this thesis we will use the Bond Graph [50] modelling framework to model analog circuits. Bond graphs are domain independent and they can be used to model any system that has flow of power. The primary benefit of using bond graphs to model analog circuits for subsequent formal verification is that the connections between components are related by the concept of energy conservation. By keeping track of power, models can be easily specified at multiple levels of abstraction, while preserving the topological organization of the design under consideration. In comparison with conventional symbolic extraction methods [66] and the techniques mentioned above, bond graph based modelling allows for a precise symbolic extraction of the system equations thus raising the confidence in verification.

1.2.2 Formal Verification of Analog Designs

Theorem Proving

In an early attempt at using theorem proving for the formal verification of synthesized analog circuits, Ghosh and Vermuri [32] prove the equivalence of analog designs that contain linear components and components with behaviour that can

be represented by piecewise linear (PWL) models. The PVS higher-order logic theorem prover is then used to prove the implication between implementation and behavioural specification built in VHDL-AMS [16].

In a similar work with theorem provers, Hanna [37] uses formal logic to define the behaviour of predicates over voltage and current waveforms. The basic behaviour of components such as resistors, power supplies and transistors are defined and then used to verify the behaviour of a NOT gate.

These early attempts are mostly based around heuristics for constructing the circuit component models and for determining the specification of the observed behaviour. They cannot be automated and are therefore not suited for larger applications. The methodology we present in this thesis uses a newly developed automated theorem prover called MetiTarski [4] and therefore could be applied to more than just basic academic problems.

Model Checking

Promising approaches for the formal verification of analog circuits consist of using heuristics to subdivide the reachable state space and then using functions and computational methods to represent the transitions between them. Since the continuous state space is being transformed into a discrete representation, model checking tools can be use to verify the resulting model.

For instance, in the early work in [43], Kurshan and McMillan extract finite state models from an analog circuit using what they call *homomorphic* functions. Their techniques attempt to reduce the computational complexity, yet at the same time preserve the behaviour of the real circuit. However, their method is only applicable to circuits with a conservative size because of the expensive space requirements. In [38], Hartong, Klausen and Hedrich introduce an extension to Computational Tree Logic (CTL) called CTL-A which defines additional operators that take into account the continuous behaviour of analog circuits. They use a similar method to [43], by

using intervals to construct the abstract state space, while using heuristics to identify transition relations. They apply their methodology on a Schmitt trigger and tunnel diode oscillator. Their techniques too are effected by a *state space explosion*. In [35], Greenstreet and Mitchell proposed a solution to the space requirements of [43] and [38], by reducing the dimension of the state space. Their methods are sound, but at the cost of the precision of the verification results. Building on these results, several model checking tools including d/dt [19], *Checkmate* [36] and *PHaver* [28] have been developed. They have been used to verify several examples including voltage controlled oscillators, a biquad low-pass filter [19], and a $\Delta\Sigma$ modulator [36]. Methodologies using Petri nets have been developed [46] for modelling and computing the transitions between abstract states, with promising results. In [67], the authors proposed a non-linear approximation for the state space. Taylor approximations are then used for the state space exploration algorithm to verify properties of a voltage controlled oscillator. As with many of the formal analog verification techniques, their methods are limited to circuits of minimal size and complexity.

The most recent research on analog formal verification work is concerned with transforming the analog verification problem to one that can be solved with Boolean satisfiability (SAT) solvers. In [64], 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 formal methods mentioned above limit the verification of the circuit to a set amount of time because of an explicit state exploration. In contrast, we propose in this thesis to use a predicate abstraction and symbolic model checking based method for the construction and verification of abstract models, which is valid over all time. In addition we enhance the symbolic model checking with a counterexample refinement procedure using constraint solving. Further details on related work on analog and mixed signal designs can be found in [68].

1.3 Proposed Methodology

In this thesis we propose a framework for the automated modelling and formal verification of analog designs. As a general guideline, we use a syntax and semantic that is familiar to the analog designer, so that the methodology could be adopted as quickly and painlessly as possible. Therefore, the starting point of the methodology (see Figure 1.1) is a circuit described using a SPICE deck [55]. This SPICE description is then systematically translated into a bond graph using the Dymola Modelling Laboratory [21] and the BondLib library [12]. Simplification rules are applied to reduce the bond graphs into their most optimal form and causality is automatically assigned. The system of equations can then be extracted from the bond graph using Dymola. If a set of ordinary differential equations (ODEs) is obtained, then we can move on to the verification step. Generally speaking, the equations representing the continuous time behaviour of an analog circuit are differential algebraic equations (DAEs). In this case, the DAEs must be transformed into their corresponding ODEs using symbolic manipulation.

Next, the optimal ODE model that was extracted from the bond graph, is used to determine the properties of interest to verify them. We propose two verification methods in this thesis, one based on theorem proving the other on model checking. Normally, proofs generated using theorem proving require a great deal of effort to conduct a proof. MetiTarski, an automatic theorem prover for real-valued analytical functions, can automatically use deduction to prove properties over inequalities in terms of trigonometric and exponential functions.

We first convert any non-linear components into their piecewise linear approximation. Using the piecewise linear (PWL) approximation, we can then generate a closed form solution of each mode using an inverse Laplace transform. Then the property of interest is turned into an inequality over the closed form solution. MetiTarski [4] can then indicate whether the inequality is true and if so will generate a full proof of its claim. In the case a closed form solution for the system cannot be

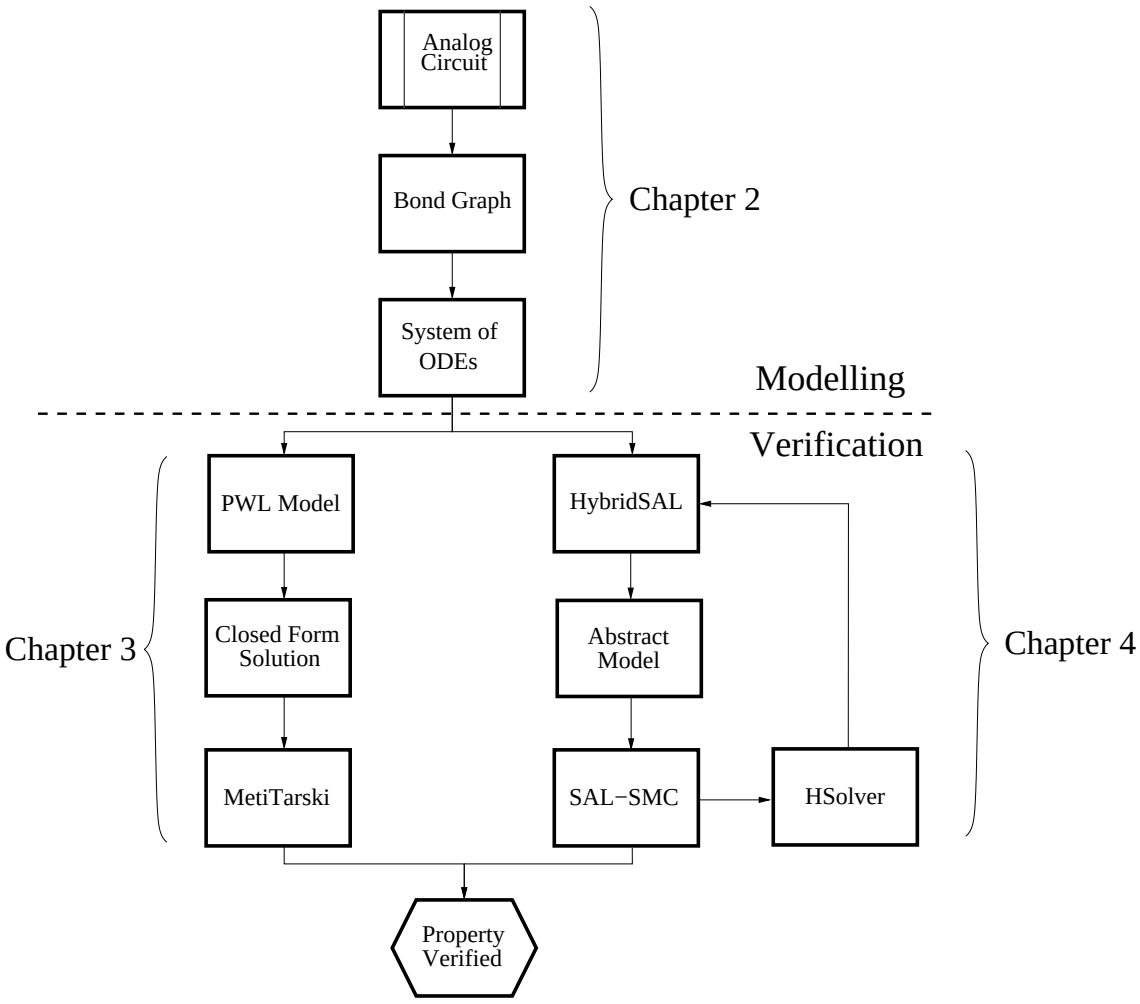


Figure 1.1: Modelling and Verification Methodology

found we can consider models that are not defined in terms of ODEs but continuous time equations. In this case, we can automate the paper-and-pencil analysis that is usually done on such models. In the case that the proof does not complete because of extreme values of the special functions, we must turn to the second method.

For the model checking method, we combine predicate abstraction and constraint solving. Using the HybridSAL [63] abstractor, the continuous analog state space is turned into a discrete Boolean state space which is verified using the SAL Symbolic model checker. This method is suitable for designs which fail normal model checking techniques due to a state space explosion. This abstraction comes at the

cost of the precision of the verification that can lead to counterexamples that do not exist in the real circuit. To eliminate the possible false negatives we use the HSolver [54] constraint solver to perform a counter-example check verification.

Two methods are used in this thesis because neither technique is perfect for all verification cases. The theorem proving method can successfully automate very specific problems, such as the determination of the mode of operation of a transistor, because it supports a large set of functions. In the case of our predicate abstraction method, HybridSal can only support polynomial functions. Additionally, an automated theorem prover will provide a complete logical proof of its claims. On the other hand, if our theorem proving methodology is not successful, then no further information is returned thus making it impossible to say where the problem with the model lies. As well, the MetiTarski tool is limited to specifying basic properties as inequalities over closed form solutions. The HybridSAL tools allow for models defined using differential equations that can be used directly after being extracted from the bond graph model and we can define more complex properties using Linear Temporal Logic (LTL). In the case of verification failure, a counterexample is returned giving a direct way to determine the error in the design. The potential problem is that the generated counterexample could be caused by the over approximation of the abstracted model. Even with a counterexample and refinement strategy, this methodology might still not terminate.

1.4 Thesis Contribution

In this thesis we have developed a complete methodology for the automated and formal verification of analog designs. The contribution of the thesis can be summarized as follows:

- A method that uses bond graphs to model analog circuits to aid in the extraction of formal specifications. We subsequently used it to generate automatically the system of ODEs.
- An approach that uses the MetiTarski theorem prover to verify functional properties.
- A technique that combines predicate abstraction, abstract model checking and constraint solving to perform the verification and a counterexample refutation.
- Application of the developed modelling and verification methodology on several analog circuits including a tunnel diode, Chua's circuit and Colpitts oscillators.

1.5 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 will present the bond graphing methodology for automatically extracting the state equations from an analog model. The basic concepts of bond graphs along with the background on the tools that are used in their construction is presented. This initial chapter is wrapped up with a complete illustrative example presented as further motivation behind bond graphs for the modelling of analog circuits. In Chapter 3, the methodology for using the MetiTarski theorem prover is developed and presented. We will discuss in detail the internals of MetiTarski and its decision procedure for proving inequalities over real functions. Furthermore, the results of several verification examples including the Tunnel Diode, Chua's Circuit and Colpitts Oscillator will be summarized. In Chapter 4, we will present the predicate abstraction methodology for the discretization of the state space and subsequent formal verification using a symbolic model checker. We will analyze in detail both the techniques employed by the HybridSAL abstractor, the SAL Symbolic Model Checker and the HSolver constraint solver in

the verification methodology. An analysis of the same circuit examples from the theorem proving chapter will be done for a comparison of the verification methodologies. Finally we present the conclusion and future work in Chapter 5.

Chapter 2

Bond Graph Modelling of Analog Circuits

2.1 Introduction

In this chapter we will describe the theory behind bond graphs in detail as well as the computer environments used to build, verify and simplify them. Continued emphasis will be put on the motivation behind their role in our methodology. This will be further supported with a detailed illustrative example that goes through the steps of automatically extracting the ODEs from an analog circuit.

Bond graphs were introduced in 1961 by Paynter of MIT [50] who believed that "energy and power alone are the fundamental dynamical variables, the ultimate currency of all physical interaction and transaction." Multi-port elements, which at the time were common in circuit diagrams, were combined with bi-directional connectors (a precursor to the name "bond") to model the flow of energy between abstract systems. His work was extended by Karnopp and Rosenberg [40] to develop a standard reference on bond graph notation for the professional engineer. Their motivation to use Paynter's energy notation was the ability to analyze multiple domains concurrently using a single formalism.

Bond graphs define a set of primitives for the modelling of a wide range of practical systems. They are a domain independent framework that allow for the universal treatment of different physical domains. By using the concepts of energy flow, effort and conservation models can be constructed at several levels of abstraction. Since the causal relationships of bond graphs can be algorithmically generated [51], the model's system of equations can be automatically extracted. A further causality analysis can also minimize the computational complexity of the properties to be verified [47]. Moreover, since bond graphs are object oriented, larger models can be built from simpler blocks reducing the need for a complex equation layer [13]. The ability of bond graphs to preserve the computation as well as topological organization of analog circuits makes them an attractive technique for verification.

2.2 Related Work

The main application of bond graphs for modelling has been in the area of systems where two or more different physical domains interact. A wide range of examples are available, including a hydraulic motor [12], an enclosed biosphere thermal dissipation simulation [12], a siphon pump [58], and a car wheel suspension system [47] just to name a few. The research on using bond graphs for modelling analog circuits has not yet matured.

In [51], Perelson developed an algorithm for the automatic conversion of an electrical circuit into a bond graph. It is shown that for certain complex series-parallel networks, bond graphs represent a better and more efficient model than circuit diagrams. Additionally, it is shown that when causality is assigned, the state variables and state equation can be automatically determined.

To address accuracy problems encountered when dealing with power devices, Besbes [7] successfully modelled an Insulated Gate-Bipolar Transistor (IGBT) using

bond graphs. The present day IGBT models were plagued by poor accuracy in representing switching behaviour. By isolating each doping region, the internal structures were properly modelled by combining equivalent circuit models with semiconductor equations. This was possible because of the modular and power flow based nature of bond graphs. Characteristic properties, such as the stored charge in the base of the BJT, were correctly handled by the bonds instead of equivalent circuits models that added stray capacitances to account for the behaviour. The work demonstrates the expressive power of bond graphs and the results indicate that bond graphs are a suitable candidate for modelling analog devices for verification. This is because of the precision obtained without introducing complex equations or relations.

In [47], Torsten and Vachoux propose to add a bond graphing model of computation (MOC) to the SystemC-AMS [2], system level modelling language. Their goal is to improve its modelling and simulation capabilities, using the domain independent nature of bond graphs to avoid the setup of complex systems of equations. Since the power bonds integrate easily with block diagrams a full formal specification can be created for analog and mixed signal designs.

On the verification side, there has been limited but promising progress with using bond graphs for verifying hybrid systems. In [58] Stromber *et. al.* use bond graphs as a front-end to the formal verification of an airplane landing gear system. By modelling the hydro-mechanically regulated pump that controls the wheel bay doors using bond graphs, they were able extract precise DAE models. In one bond graph they assume that the hydraulic power supply system behaves as an ideal constant pressure source. In the other, they explicitly model the hydro-mechanic regulator that keeps the pressure constant. The concepts of bond graphs allow for this multi abstraction based description of physical systems. Depending on which model was used, different parts of the system could be isolated and analyzed.

2.3 Preliminaries

2.3.1 SPICE and Schematic Capture

It has already been stated that the most common way to ensure that a designed circuit works properly is through simulation. The tool of choice for the simulation of analog circuits is the Simulation Program with Integrated Circuit Emphasis (SPICE). The benefit of this software package is that it is open source and thus there are several versions for both academic and industrial uses. The initial versions were entirely text based, where a circuit was described using a text based netlist description called a *deck*. For example, a single resistor connected in series with a single DC voltage supply could be described using the deck file shown in Figure 2.1.

```
V1 1 0 DC 0.3
R1 1 0 50
```

Figure 2.1: A Basic SPICE Deck

In this case, each node of the circuit is represented by a number. The DC voltage supply has a value of 0.3 volts and the resistor has a resistance of 50 Ω.

Since much effort is required to construct netlists by hand for complex designs, graphical schematic capture programs have been developed for the automatic generation of netlists from abstract models. Schematic capture programs work in the same way as if the circuit was being drawn with paper and pencil. Circuit symbols and external models called subcircuits can be connected with lines that represent wires. The schematic capture programs save much time for the designer since the circuit netlists can be automatically generated for input into a SPICE simulator.

2.3.2 Bond Graph Theory

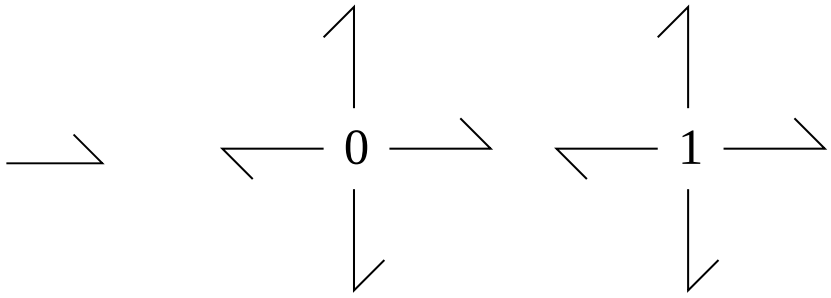
There exists nine basic bond graph elements (as shown in Table 2.1) that can be used to model any possible physical component. The storage group contains the elements for capacitive storage (C type) and inductive storage (I type). The supply group contains the sources of effort and flow. The reversible 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 Kirchoff Voltage Laws (KVL) and Kirchoff Current Laws (KCL).

Table 2.1: 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

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 2.2(a)). It is the energy link between two components. It is represented graphically by a harpoon (half arrow), which 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.



(a) Power Bond (b) Junctions

Figure 2.2: Basic Bonds

The other basic component is the junction, which represents a circuit node or mesh (Figure 2.2(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.

Elements

Using the bonds and junctions, it is possible to connect discrete elements together in a bond graph. There are different types of single and multi port interfaces that can be used to represent many possible configurations. 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 2.4(a)).

Simplifications

There exists two levels of simplification that can be performed on bond graphs. Firstly, there are equivalence rules for the junction objects. 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 (see Figure 2.3).

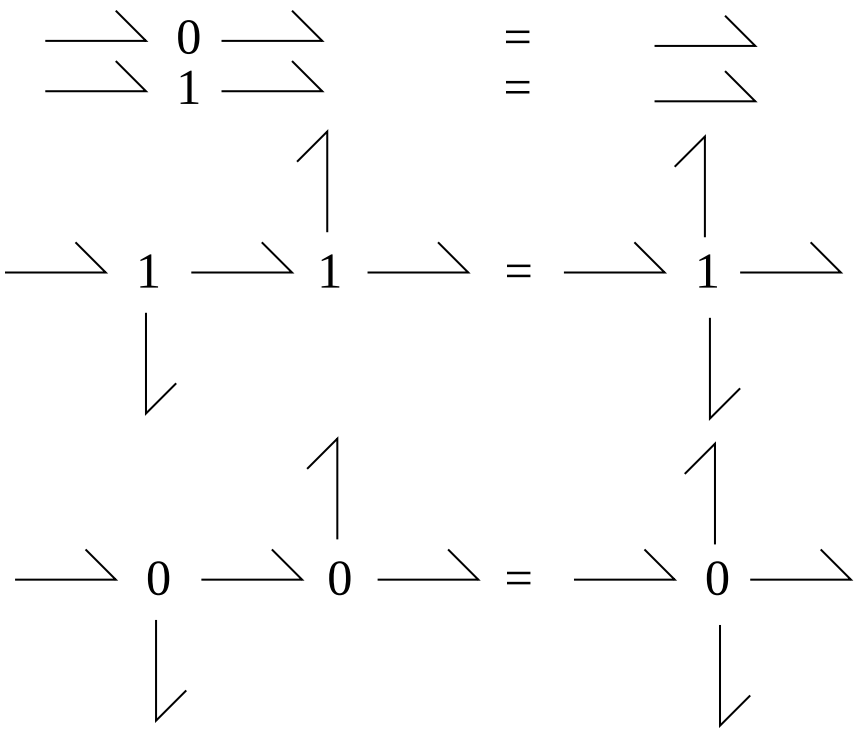


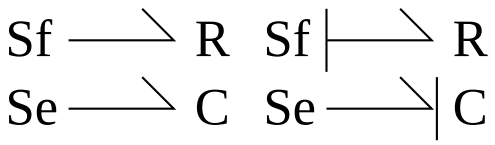
Figure 2.3: Simplification Rules for Junctions [8]

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 [31]. 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.

Causality

Causality is the determination and representation of the directional relationship between an input and an output [8]. 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 [8, 27, 40]. 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 [12] (Figure 2.4(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 [47].



(a) R and C Bonds (b) Causality Added

Figure 2.4: Bond Graph Basics

The causality 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 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.

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.

2.3.3 Dymola Modelling Laboratory

Dymola [21] is an advanced tool for the complete modelling and simulation of physical systems including electrical, thermal and mechanical domains. It comprises of a graphical user interface (GUI) that allows connections between abstract models. As well, it contains a simulator that employs advanced symbolic manipulation techniques to produce a solution to the system equations. At its core is the open source Modelica language [61] that defines the libraries and components. The automatically generated symbolic solution to the system equations is also represented in the Modelica syntax. A tunnel diode can be defined at the equation level using the syntax in Figure 2.5.

```
model TunnelDiode
  extends Modelica.Electrical.Analog.Interfaces.OnePort;

equation
  i = v^3 - 1.5*v^2 + 0.6*v;
end TunnelDiode;
```

Figure 2.5: Modelica Syntax

Since Modelica is object oriented, the TunnelDiode model must extend from a previously defined "OnePort" object that defines a generic input/output relation. The

tunnel diode in this example is modelled by a third order polynomial where i represents the current through the diode and v the voltage drop across it.

The Modelica language and consequently Dymola are object oriented. This allows for component reuse as well as the ability for external libraries to be easily built. The Dymola Bond Graphing Library (BondLib) is an example of one such library. Developed by Cellier *et al.* [12], BondLib demonstrates the benefit of object oriented modeling with bond graphs. The transistor models for BJTs and MOSFETS contained in the library can be set to different levels of complexity [12]. At each level, parasitics, current leakages and non-ideal effects can be added to the model by specifying the correct parameters in Dymola. The parameters are available to the modeler to dynamically alter the bond graph. For example, the difference between the MOSFET level 0 and 1 bond graphs is that the capacitances between the source, drain, gate and body are set to zero. With the help of bond graphs and BondLib the designer is able to maintain a deep understanding of the dynamics of the design under verification.

The symbolic manipulation performed by Dymola is quite advanced. By correctly ordering the equations and determining the constraints of the model, the final state calculation can be simplified. For example, in the physical domain Dymola will determine that two rigidly connected bodies can be represented by a single state variable [23]. Similarly, in the electric domain this is analogous to multiple capacitors that are connected in parallel or series being lumped together into a single effective capacitance. Function tearing [25] of non linear equations is also used to "break" algebraic loops and reduce the dimension of sub-models. This further decreases the number of state variables in the final solution. By combining the domain independent properties of bond graphs as well as the advanced symbolic manipulation of Dymola, the automatic extraction of ODEs from an analog circuit is possible.

2.3.4 Mathematica

The tearing function described for reducing the dimension of the differential equations comes at a computational cost. The symbolic methods employed by Dymola can potentially create quite a large number of "dummy" variables that are used to effectively "break" the algebraic loops. The only problem is that for large systems, the resulting Modelica description can be very complicated. Since Dymola does no internal simplification, the determination of the differential equations by hand can be quite taxing. Therefore, it is necessary to use an algebraic system such as Mathematica [3] to automatically simplify and remove the redundant equations. Mathematica, an advanced complete algebra system, contains functions such as "Simplify" (see Figure 2.6) that can be used to perform algebraic transformations on a set of generic expressions.

```
In[1] := Simplify[Sin[x]^2+Cos[x]^2]
Out[1]:= 1
```

Figure 2.6: Mathematica Simplify Command

MathModelica [39] is a software bridge between Modelica and Mathematica. It enables the use of the Mathematica "Simplify" function directly onto the Modelica description. By combining Dymola, Modelica, Mathematica and Mathmodelica the path to the automated extraction of the system of differential equations is complete.

2.4 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 2.7.

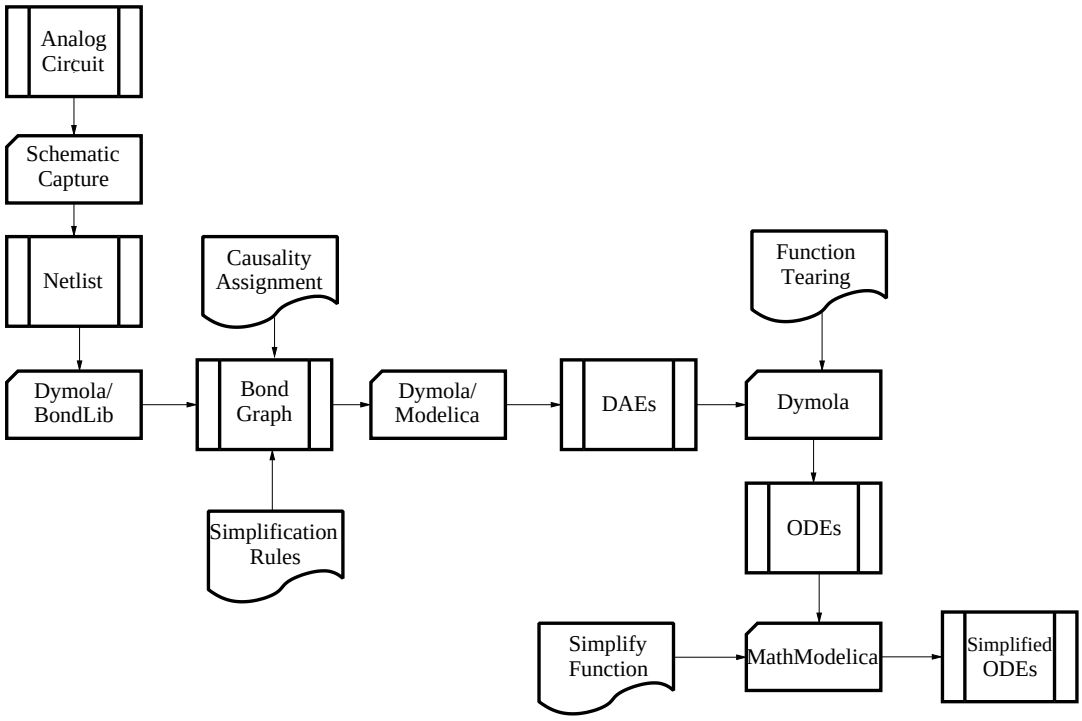


Figure 2.7: Bond Graph Modelling Methodology

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 in conjunction with the BondLib library, a bond graph is created directly from the SPICE model by using a one-to-one mapping between the nodes and components. At this point, the bond graph is not in its simplified form. Using the rules described before, 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 preferred causality is assigned to the bonds to ensure that during the extraction stage, differential equations are produced instead of integrals. 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. In this case, the simplification rules in the algebraic system Mathematica are employed to automate the ODE extraction.

2.5 Illustrative Example

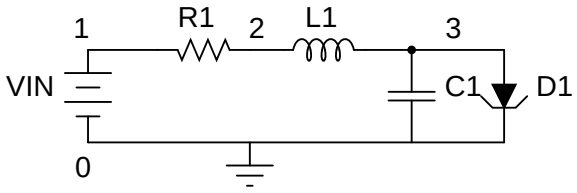
The tunnel diode oscillator circuit in Figure 2.8.(a), which has been used by many researchers (e.g.,[36, 38]) as a benchmark in formal verification research, will be used as an illustrative example to demonstrate the modelling methodology.

The tunnel diode oscillator demonstrates the effect of resonant tunneling that causes a negative resistance to appear at small forward bias voltages. Essentially, for some range of voltages the current through the tunnel diode decreases with increasing voltage. This negative resistance can be used to create a reliable oscillator that functions under many different operating conditions.

2.5.1 Spice to Bond Graph

Figure 2.8.(b) is a SPICE representation of the tunnel diode oscillator. Each node is represented by a number and each component is represent by an alphanumeric name. As expected, the conversion is a one to one mapping of the circuit diagram to the SPICE model. An external subcircuit model defines the behaviour of the tunnel diode.

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 2.9. This is preformed according to the bond graphs rules described in Section 2.3.2.



(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

Figure 2.8: Tunnel Diode Oscillator

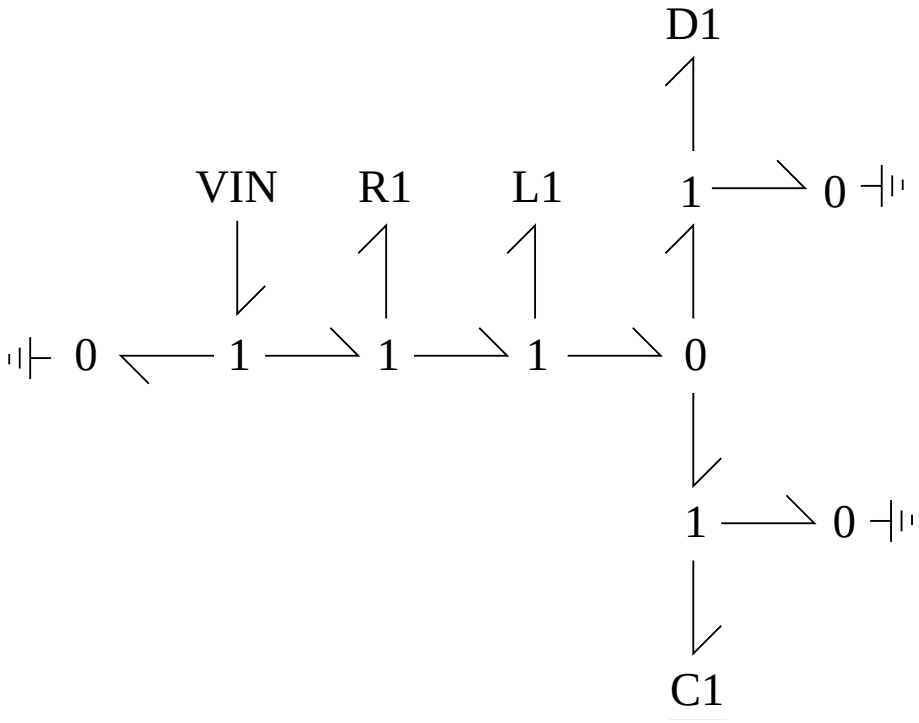


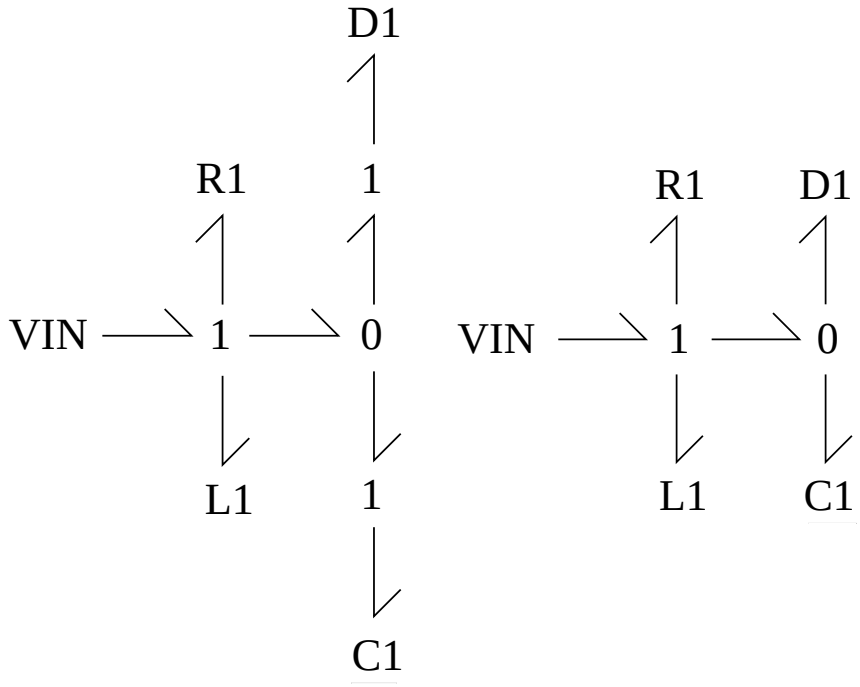
Figure 2.9: Initial Bond Graph

2.5.2 Simplifications

Simplifications of the bond graph in Figure 2.9 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 junctions in series can be merged together. The resulting simplified bond

graph is given in Figure 2.10(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 2.10(b).



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

Figure 2.10: Tunnel Diode Bong Graph Simplifications

2.5.3 Causality Assignment

The next step in the conversion process is to add a causality stroke to each bond. The C and I components have a preferred causality that is assigned first with the current being calculated at the inductor and away from the capacitor. The same idea is used to determine the placement of the causal bars at the 0 and 1 junctions. Since at the 1 junction there is a single effort equation since the flows are equal there should be only one bond without a causal bar. For the 0 junction where there is a single flow equation and the efforts are equal, there should be only one causal

bar. The causality of the resistor and the tunnel diode are arbitrary. The final bond graph is defined as shown in Figure 2.11.

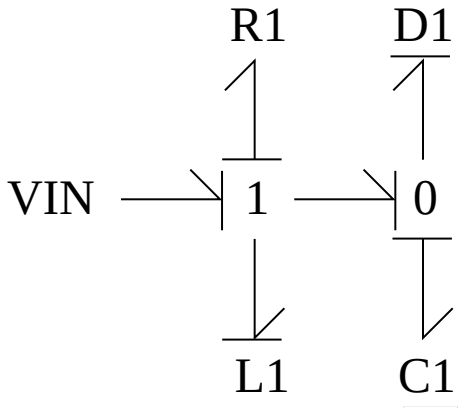


Figure 2.11: Tunnel Diode Causal Simplified Bond Graph

2.5.4 Extracting the System of Equations

Once the bond graph is built, the set of system equations can be extracted and simplified. To remove redundant equations we use the rewriting functions of Mathematica.

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

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:

$$\dot{x_k} = \frac{dx_k}{dt} = \dot{x} = F_k(\mathbf{x}(t), \mathbf{u}(t), t)$$

where $\mathbf{x}(t)$ are variables defining the voltage across the capacitances and the current through the inductances. $\mathbf{u}(t) \in \mathbb{R}^m$ are variables defining the input signals, with the vector fields F_k.

Using the Dymola environment, the bond graph for the tunnel diode is constructed. The BondLib library contains graphical modules for bonds and nodes. Dymola then converts the bond graph into a Modelica simulation description. Index reduction, function tearing and further algorithms then automatically transform

the DAEs to ODEs from the Modelica description as shown in Figure 2.12. In particular, the symbolic solution to the system of equations can be read directly from the Modelica description. There are cases where it is not as simple to determine the equations. Since Dymola uses dummy variables to aid in the conversion from DAEs to ODEs, many extra variables may be present in the final output. By using rewriting rules in Mathematica, the system of DAEs can be simplified.

```
// Dummy Variables
  resistor.v := resistor.R*inductor.i;
  resistor.n.v := constantVoltage.V-resistor.v;
  inductor.v := resistor.n.v-capacitor.v;
  capacitor.p.i := inductor.i-tunnelDiode.p.i;

  tunnelDiode.p.i = capacitor.v^3 - 1.5*capacitor.v^2 + 0.6*capacitor.v;

// 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;
```

Figure 2.12: System of ODEs generated by Dymola

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 2.8). 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.

2.6 Summary

Bond graphs have been characterized as "the most basic graphical modeling paradigm that is fully objected-oriented" [12]. 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 and thus the system of equations can be extracted from the model.

Now that a method for extracting the system of ODEs from a model has been presented, the next chapter will describe in detail a method for formally verifying properties on the extracted models. If during verification, the models need to be changed to take into account different topologies, the bond graphs can be quickly modified and the new system equations obtained.

Chapter 3

Verification by Automated Theorem Proving

3.1 Introduction

In this chapter we will present the work on using the MetiTarski theorem prover for the verification of properties on the system of ODEs that were extracted from the analog circuit bond graph model. There is a great need for research in this area since there has been limited advancement on this front. We strongly believe that automated methods should be developed to take advantage of the high confidence in results provided by theorem proving, where a complete proof with logical inference steps is generated. The main hurdle is the great deal of expertise needed to interactively guide a proof to fruition.

MetiTarski [4] is an automatic theorem prover for real-valued analytical functions, including trigonometric and exponential functions. It works by a combination of resolution inference and algebraic simplification, invoking a decision procedure (QEPCAD) [9] to prove polynomial inequalities. Since many of the circuit equations we deal with in the analog domain contain exponentials, it is a viable option

for formally verifying the properties of interest. The output of MetiTarski is a complete proof that contains algebraic simplification and decision procedure calls that can be verified using other tools.

In the last decade a new engineering field has emerged: hybrid system theory. It encompasses techniques for the automatic design and analysis of systems with real-time and continuous behavior. In [5], the authors use MetiTarski to solve various hybrid system verification problems including collision avoidance, navigation and biological mutant examples. In their methodology, they model the hybrid systems over several modes of operation. In each mode, the variables of the system vary according to a set of ODEs. By using the inverse laplace transform they solve for closed form solutions of each mode of operation. Properties are then proven in each mode using MetiTarski. By taking advantage of hybrid system theory, theorem proving is now emerging as a serious candidate for the verification of analog systems. This is because an analog circuit verification can be viewed as a hybrid system if it is defined using a proper model.

3.2 Preliminaries

3.2.1 Theory Behind MetiTarski

There exists few methods to automatically prove statements involving elementary functions such as ln, exp, sin, cos and sqrt. In their MetiTarski tool, Akbarpour and Paulson [4] use the decidability of real closed fields to automatically prove inequalities over elementary functions. MetiTarski replaces the functions with upper and lower bounds in an attempt to reduce the problem so that a decision procedure can be used to automatically prove the property of interest. Before delving into its internals, it is important to understand some basic formal method terminology (see Table 3.1).

Table 3.1: Formal Definitions [52]

Quantifiers	$\exists, \forall$ (There exists and For All)
Axioms	Logical Statements
Theory	Set of Axioms
Model	Abstract Structure that satisfies a Theory
Consistent Theory	The theory has at least one model
Complete Theory	Every model that is true can be proven
Decidable Theory	An algorithm exists for evaluating the truth of a model

An algebraic field is an abstract structure that contains the operators of addition, subtraction, multiplication and division. To be called real closed, the field must satisfy a set of axioms defined by the operators as well as be able to represent atomic polynomial formulas that contain inequalities. These formulas can also contain conjunctions, disjunctions, negations and quantifiers. A real closed field must also be ordered and thus meet following axioms [52]:

$$-1 : \text{is negative}$$

$$\forall x : x \text{ or } -x \text{ is positive}$$

$$\forall x, y \text{ positive } : x + y, x * y \text{ are positive}$$

Alfred Tarski proved in 1930 that RCF was decidable by presenting quantifier elimination procedure [11]. Given an RCF quantified formula, an equivalent formula that has no $\forall$ or $\exists$ components can be produced. Take for instance,

$$a \neq 0 \wedge (\exists x) : (ax^2 + bx + c = 0)$$

from the rules governing the solution of the quadratic equation [33], we know that for there to be a real solution the discriminant must be greater than or equal to zero or

$$b^2 - 4ac \geq 0$$

and by using this result, we can remove the quantifiers to produce the following result

$$(a \neq 0 \wedge b^2 - 4ac \geq 0) \vee (a = 0 \wedge b \neq 0) \vee (a = b = c = 0)$$

This example is trivial, but for more complex equations his method could not be implemented in practice. There has been further work on the decision procedure of RCF by McLaughlin, Harrison and Hormander [48]. Their implementations are more efficient than Tarski's but fail to work on polynomials with a degree greater than six. QEPCAD-B [9] is a recent and efficient decision procedure for the complete theory of RCF. It uses cylindrical decomposition to extract polynomials from an input formula and then uses abstraction techniques to generate an r-dimensional space, where r is the number of variables in the input formula. The abstract regions are then arranged in a cylindrical domain which allows an efficient removal of quantifiers via linear algebraic methods.

3.2.2 MetiTarski Input Syntax

MetiTarksi operates on the first-order formula in the Thousands of Problems for Theorem Provers (TPTP) format that includes the corresponding axioms. Take for instance the following code.

```
fof(
     Tunnel,conjecture, ! [X] :
     (
       (0 <= X & X <= 2.39*10^(-9)) =>
       -0.0059 - 0.000016*exp(-2.55*10^8*X) + 0.031*exp(-5.49*10^7*X)
         < 0.03
     )
   ).
```

Figure 3.1: MetiTarski Syntax

"fof" indicates to MetiTarski that the logic language used is a first-order formula. It

is then followed by a label of the proof as well as the keyword *"conjecture"* indicating that the following formula is to be proved with the included axioms. The conjecture is read as follows: For all (!) X between 0 and 2.39×10^{-9} the formula is always less than 0.03. For a complete syntax guide see Table 3.2.

Table 3.2: TPTP Syntax Guide for Figure 3.1

fof	First order logic formula
!	Universal Quantifier ($\forall$)
X	Quantified Variable
&	Logical AND
exp	e (exponential function)
$<$	Less Than

Axioms

In addition to the problem definition, axioms must be appended to tell MetiTarksi what clauses to use when performing the special function to polynomial substitution.

Fortunately, this is automated using external scripts, but it is still necessary to indicate what family of axioms is required. It is very important that only the necessary axioms files are included in the TPTP description, since each set adds computational complexity to the final proof. For example, there are two sets of axiom declarations for the exponential function. One for *regular bounds* and one for *extended bounds*. There are cases where including the extended bounds will make the inequality under test unsolvable. In reality, removing the extra axioms will enable MetiTarski to complete the proof. The converse is also true, if for instance the TPTP description contains trigonometric functions and those axioms are not included, then the proof will also be unsolvable.

Range Reduction

Since the elementary functions are being replaced by polynomials, it is necessary that the bounds on the quantified variables vary closely around 0. If the quantified variables get too high or too low, the accuracy of the polynomials is greatly diminished. Therefore, MetiTarski uses internal range reduction methods to properly scale the value of the variables to ensure that the polynomials are an accurate approximation to the real function.

There are cases though when the internal range reduction is not enough. When dealing with closed form solutions that contain trigonometric functions with bounds that are very large (greater than 6π). Then manual range reduction is required. Since the trigonometric functions are cyclical, it is possible to remove or add multiples of 2π. By doing this, MetiTarski will be able to complete proofs that fail otherwise.

3.2.3 Piecewise Linear Approximations

The first thing to note is that the input to MetiTarksi is a closed form solution that can contain any number of analytical functions. To obtain closed form solutions we have used the inverse Laplace Transform on the system of equations representing the behaviour of the system. Therefore, it is necessary that the ODEs be linear and thus we must use piecewise linear models for the analog component. PWL models are adequate to be used to model the components for the following reasons [15]:

- Piecewise-linear circuits are the simplest class of nonlinear circuits.
- The behaviour of many op-amp and diodes and switch circuits can be reasonably approximated as piecewise-linear.
- Linear methods are substantially more tractable than non-linear ones, even when they divide the problem into multiple modes.

Take for instance the behaviour of the tunnel diode. As shown in Figure 3.2, the behaviour can be correctly approximated by a piecewise linear model that operates over three modes of operation.

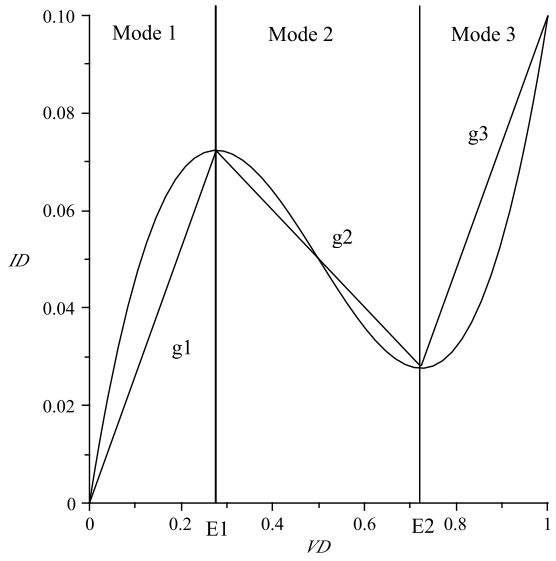


Figure 3.2: Tunnel Diode Current Linearization [15]

3.2.4 Inverse Laplace Transform for Solving Linear Analog Circuits

A common way to solve the variables of a circuit is to perform nodal analysis, by using KVL and KCL to work through the circuit solving for node voltages and currents. The problem is that many equations must be solved for required several computation steps. The inverse Laplace transform method is an another efficient way to solve for the closed form solutions of an analog circuit. Any circuit can be described by n first order differential equations [65]. By extracting the differential equation for each energy storing element (capacitors and inductors) and then assembling the "state model" matrix, an inverse Laplace transform can be used to solve for the currents and voltages. The benefit of using this method is that it is a

one step process, to go from equations to solution instead of solving many equations encountered when using the Kirchoff circuit laws. The standard circuit state model [65] is

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}$$

Where the **x** vector represents the state variables from the energy storing elements. **A** is the *circuit matrix* because it contains the values of the circuit parameters. The **B** matrix is called the *distribution matrix* because it contains values that depend on the location of the power sources.

Let X denote the Laplace transform of **x** (X = $\mathcal{L}x$); then $sX - x_0 = AX + \frac{B}{s}$, and solving for X we have $X = (sI - A)^{-1}(x_0) + \frac{B}{s}$. With the state model defined, we can take the inverse Laplace Transform of X to solve for the closed form solution of the circuit equations.

3.2.5 Maple Computer Algebra System

Maple [1] is a full computer algebra system that can be used to efficiently manipulate many types of data. As well there are internal functions that are available to perform many actions on matrices, solving equations and symbolically evaluating expressions. The functions in Table 3.3 are used to solve the closed form solutions of the ODEs that were obtained from the bond graphs.

Table 3.3: Useful Maple Functions

Function	**Description**	Example	Result
fsolve()	Uses numerical approximation techniques to find a decimal approximation to the solution to the equation	fsolve(x=0.1,t)	Value of t where x(t)=1
eval()	Evaluates an expression	eval(x^2,x=2)	Will output the value 4
invlaplace()	Takes the inverse laplace transform of vector X	invlaplace(X,s,t)	Returns x(t)

3.3 Verification Methodology

In the following, we demonstrate a methodology for the automatic verification of functional properties of analog designs using MetiTarski. An overview of the proposed, methodology is given in Figure 3.3.

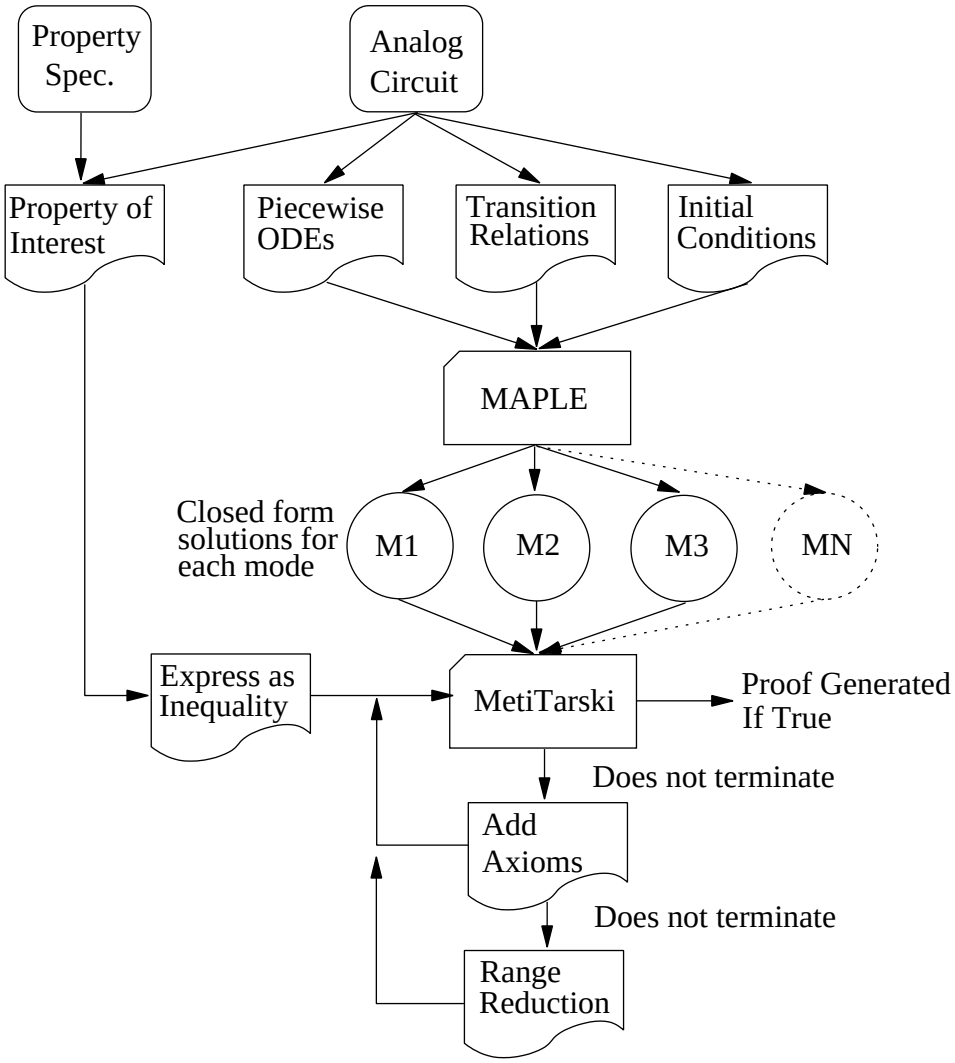


Figure 3.3: Overview of the Verification Methodology

We first obtain the system of differential equations from the circuit of interest. Any non-linear elements are transformed into their PWL models. The transition

relation between each mode of the PWL model is determined and differential equations (ODEs) are constructed over each mode of operation. Starting in any mode, the ODEs and initial conditions are supplied to the computer algebra system Maple and an inverse Laplace transform is performed to find a closed form solution for each state variable as a function of time. Using the transition relations, Maple is used to find the time instance where the system switches modes. At that time, the initial conditions for the next mode are calculated and an inverse Laplace transform is performed to again find a closed form solution. This is repeated until each mode has been visited as shown in Figure 3.4.

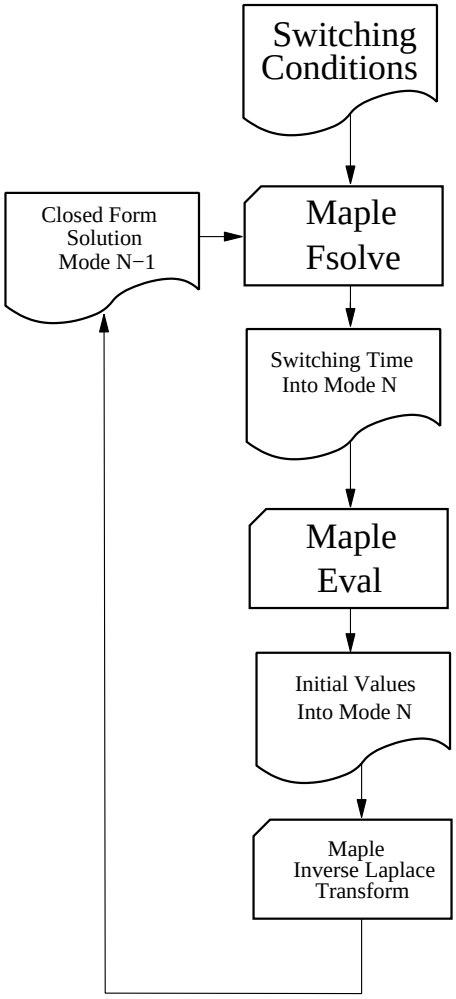


Figure 3.4: Determining the Closed Form Solutions for Each Mode

We then turn the verification property into an inequality over special functions. A first-order formula in the Thousands of Problems for Theorem Provers (TPTP) format (see Chapter 3.2.2), including the corresponding axioms, is then supplied to MetiTarski. MetiTarski uses an extension of the TPTP format, including infix notation for the arithmetic and relational symbols [59, 60].

If MetiTarski is successful, it delivers a proof and we are done. If unsuccessful, it will run until terminated by the user. Additional axioms are then added or removed to aid MetiTarski in formulating a proof. There are certain axioms that are available for special functions that take on extreme values. Including them unnecessarily in proofs will increase the computation time. If still unsuccessful, range reduction is applied to the trigonometric functions to further eliminate any extreme values that can cause problems for MetiTarski's decision procedure.

In cases when MetiTarksi still does not terminate, it could be because the functions take on values that are beyond the limits of the deduction methods. There are also cases when a closed form solution to the ODEs cannot be computed, due to trigonometric or non-linear terms. As well, it is possible the loss of precision of the PWL modelling could be affecting the results. In these cases a different approach altogether must be taken.

Since MetiTarski can prove inequalities over many analytical functions, we propose instead of using linear methods to solve for the closed form solution of a system of differential equations, and hence we will start directly with circuit model current voltage relationships. Properties concerning the DC operating points (mode of operation) could then be easily verified. Essentially, we are eliminating the laborious paper- and-pencil analysis that is necessary.

3.4 Applications

The following section will present examples of analog circuits on which we applied our methodology. At the end of each example, the results are summarized in order of the property that was verified. The experiments were all performed on a 2.8 GHz Dual Quad-Core Mac Pro.

3.4.1 Tunnel Diode Oscillator

The tunnel diode oscillator from Chapter 2 is shown again in Figure 3.5. Instead of node numbers, we now show the current and voltage variables on the circuit diagram. It demonstrates the effect of resonant tunneling that causes a negative resistance to appear at small forward bias voltages as shown in Figure 3.2. Essentially, for some range of voltages, the current through the tunnel diode decreases with increasing voltage. This negative resistance can be used to create a reliable oscillator that functions under many different operating conditions. We intend to verify that for certain initial states and component values, the tunnel diode oscillator will not oscillate. By verifying this property, we will be able to eliminate designs that do not work.

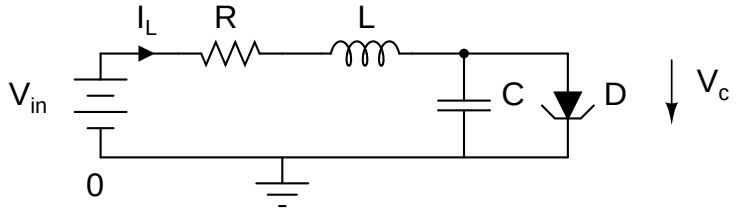


Figure 3.5: Tunnel Diode Oscillator

The differential equations of the circuit are defined as:

$$\dot{V_C} = \frac{1}{C}(-I_D(V_C) + I_L)$$
$$\dot{I_L} = \frac{1}{L}(-V_C - RI_L + V_{in})$$

$I_D(V_C)$ is a PWL model that has three modes of operation. We can define the PWL model [15] of the tunnel diode as

$$I_D(V_C) = -\frac{1}{2}(G_1E_1+G_2E_2)+(G_0+\frac{1}{2}G_1+\frac{1}{2}G_2)V_D+\frac{1}{2}G_1\,|V_C-E_1|+\frac{1}{2}G_2\,|V_C-E_2|$$

Where g_1, g_2 and g_3 represent the slope of the best fit curve in each mode. E_1 and E_2 represent the voltages where the model switches modes. Figure 3.2 shows the real continuous behaviour of the tunnel diode as well as the PWL approximation. In region 1, $G_0 = g_1$. In region 2, $G_0 + G_1 = g_2$. In region 3, $G_0 + G_1 + G_2 = g_3$. In this example, $g_1 = 0.2616$, $g_2 = -0.0992$, $g_3 = 0.2599$, $E_1 = 0.276$ and $E_2 = 0.723$ giving

$$I_D(V_C) = \begin{cases} 0.2616V_C & \text{if } V_C < 0.276 \\ -0.0992V_C + 0.0997 & \text{if } 0.276 < V_C < 0.723 \\ 0.2599V_C - 0.1599 & \text{if } V_C > 0.723. \end{cases}$$

The system is now completely specified. Each mode is defined by a set of ODEs and switching constraints. The resulting time-deterministic hybrid model can be illustrated as an FSM as shown in Figure 3.6. Each mode of operation is represented by a state circle and the switching constraints are indicated above each directional arrow.

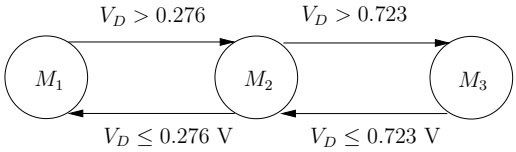


Figure 3.6: The Hybrid Model of the Tunnel Diode Current

Suppose the parameter values are $R = 50\Omega$, $L = 10^{-6}$ H, $C = 10^{-9}$ F, $V = 0.3$ V, the dynamics of mode 3 can be written as the first order linear differential system $\dot{x} = Ax + B$, where the A matrix represents the coefficients of the state variables and the B matrix represents the constants.

$$x = \begin{bmatrix} I_L \\ V_C \end{bmatrix}, A = \begin{bmatrix} -3 \times 10^5 & -10^6 \\ 10^9 & -2.621 \times 10^8 \end{bmatrix}, B = \begin{bmatrix} 3 \times 10^5 \\ 0 \end{bmatrix}$$

Let X denote the Laplace transform of x (X = $\mathcal{L}x$); then $sX - x_0 = AX + \frac{B}{s}$,

and solving for X we have $X = (sI - A)^{-1}(x_0) + \frac{B}{s}$. With the initial state as $x_0 = (0.025, 0.74)^T$, chosen to lie in Mode 1. Using Maple we have

$$X = \begin{bmatrix} \dfrac{(s + 0.262 \times 10^9)(0.55 \times 10^{-2} + \dfrac{0.300 \times 10^6}{s})}{(s^2 + 0.262 \times 10^9 s + 0.108 \times 10^{16})} - \dfrac{0.131 \times 10^6}{(s^2 + 0.262 \times 10^9 s + 0.108 \times 10^{16})} \\ \dfrac{(0.550 \times 10^7 + \dfrac{0.300 \times 10^{15}}{s})}{(s^2 + 0.262 \times 10^9 s + 0.108 \times 10^{16})} + \dfrac{(0.131s + 0.393 \times 10^5)}{(s^2 + 0.262 \times 10^9 s + 0.108 \times 10^{16}))} \end{bmatrix}$$

The closed form solutions of the state variables are obtained by taking the inverse Laplace transform $\mathcal{L}^{-1}X$ and we obtain

$$V_C(t) = 0.116\, e^{-2.58 \times 10^8 t} + 0.278 - 0.262\, e^{-4.19 \times 10^6 t}$$
$$I_L(t) = 0.448 \times 10^{-3}\, e^{-2.58 \times 10^8 t} + 0.0727 - 0.0677\, e^{-4.19 \times 10^6 t}$$

Now we have the state space representation of the system for the third mode of operation. The next step is to determine the time when the tunnel diode switches from mode 3 to mode 2. By using Maple, we determine that the condition $V_C \leq 0.723$ is true at $t = 2.38 \times 10^{-9}$ s. The values of both V_C and I_L are evaluated at this time. We then use these values for x_0 and again repeat the process of finding the matrix X, and taking its inverse Laplace transform. This is repeated as shown in Figure 3.4 until we have visited each mode and have generated the closed form solutions for the two state variables.

For mode 2, the closed form solutions are

$$V_C(t) = 0.278 + 0.0025\, e^{8.79\times10^7 t} - 0.0045\, e^{-1.10\times10^7 t}$$
$$I_L(t) = 0.0727 + 0.00039\, e^{-1.10\times10^7 t} - 0.000028\, e^{8.79\times10^7 t}$$

For mode 1, the closed form solutions are

$$V_C(t) = 0.323 - 01.64\, e^{-2.56\times10^8 t} + 0.56\, e^{-4.21\times10^6 t}$$
$$I_L(t) = -0.076 - 0.00064\, e^{-2.56\times10^8 t} + 0.144\, e^{-4.21\times10^6 t}$$

To demonstrate the power of MetiTarski, we seek to define an oscillation property that can be proved over all modes of operation. One such property is *"For a set of initial conditions, the circuit will not oscillate"*. This property can be more exactly defined as *"The current through the inductor will never pass some upper or lower bound"*. It can be described formally as :

$$[L_I < 0.03]$$

For example, to prove that in mode 1, L_I is always less than 0.03 we use the syntax shown in Figure 3.7.

```
fof(
 Tunnel,conjecture, ! [X] :
 (
   (0 <= X & X <= 2.39*10^(-9)) =>
   -0.0059 - 0.000016*exp(-2.55*10^8*X) + 0.031*exp(-5.49*10^7*X)
   < 0.03
  )
).
```

Figure 3.7: MetiTarski Input For the Verification of Mode 1

Now suppose we choose the component values $R = 0.3\ \Omega$, $L = 10^{-6}$ H, $C = 10^{-9}$ F and $V = 0.3$ V. Using the same inverse Laplace transform methodology we

get the closed form solutions of the state variables. The property of interest is now: *For a set of initial conditions the trajectory of the oscillation reaches a final set and remains bounded* [28]. This can be described formally as:

$$[V_C > 0 \wedge V_C < 0.9 \wedge L_I > 0 \wedge L_I < 0.08]$$

MetiTarski proves both properties over the three modes of operation. For property 1, it is proved that the circuit does not oscillate. For property 2, it is proved that the oscillation present in the circuit is bounded. Complete runtime results of this example can be found in Table 3.4. In the table, each row represents a single inequality proved using MetiTarski. The number indicates which mode of the PWL model the tunnel diode is in. The next field indicates which variable is being checked (IL or VC) followed by whether it is an upper (U) or lower (L) bound.

Table 3.4: Tunnel Diode Oscillator Verification Runtimes (in seconds)

Non Oscillation	
Tunnel-1-IL	0.1
Tunnel-2-IL	4.0
Tunnel-3-IL	0.3

Bounded Oscillation	
Tunnel-1-VC-U	0.2
Tunnel-1-VC-L	0.4
Tunnel-2-VC-U	2.7
Tunnel-2-VC-L	0.6
Tunnel-3-VC-U	0.3
Tunnel-3-VC-L	0.5
Tunnel-1-IL-U	0.5
Tunnel-1-IL-L	0.3
Tunnel-2-IL-U	0.6
Tunnel-2-IL-L	3.9
Tunnel-3-IL-U	0.3
Tunnel-3-IL-L	0.6

The proofs all complete quite quickly, each under five seconds. For Tunnel-2-IL, Tunnel-2-VC-U, and Tunnel-3-IL-U, the closed form solutions that were generated using Maple contained large bounds on the exponential and trigonometric functions. Initially, MetiTarski was unable to prove these cases. By adding the required extended axioms, MetiTarski was able to complete the proof.

3.4.2 BJT Colpitts Oscillator

The Bipolar Junction Transistor (BJT) Colpitts oscillator (Figure 3.8) is another example of an oscillator circuit that has a complex behaviour, which can be properly modeled with a PWL approximation consisting of two modes.

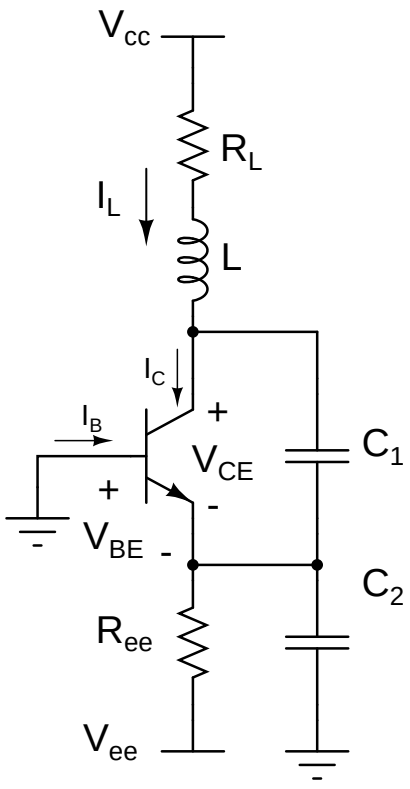


Figure 3.8: BJT Colpitts Oscillator

In order to fully understand the behaviour of the circuit it is necessary to identify the different modes of operation of the BJT. The BJT is a semiconductor device that can operate in four different regions or modes of operation. The mode is determined by the voltage across its three terminals. For instance, Kennedy [41] has shown that the BJT inside the Colpitts oscillator operates only in two distinct regions: forward active and cutoff. We use MetiTarski to prove this result formally. It is important to note that the method automates the formal analysis of the operating point of the BJT. Currently, there exists no automated, formal way

to determine the modes of operation of a semi-conductor device. 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

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

Consider the BJT Colpitts circuit with the following parameters: $L = 98.5\mu$ H, $V_{CC} = 5$ V , $R_L = 35\,\Omega$, $C_1 = C_2 = 54$ nF, $R_{EE} = 400\,\Omega$, $V_{EE} = -5$ V and $R_{ON} = 100\,\Omega$. We use the Laplace transform method described before to solve the system of ODEs over the two modes of the PWL function.

For mode 1, the closed form solutions are

$$\begin{aligned} V_{C1} &= 0.001 - 2.67\,e^{-1.93\times10^6 t} + 2.67\,e^{8.45\times10^5 t}\cos(1.71\times10^6 t) \\ &\quad + 3.20\,e^{8.45\times10^5 t}\sin(1.71\times10^6 t) \\ V_{C2} &= -0.0011 + 0.146\,e^{-1.93\times10^6 t} + 0.205\,e^{8.45\times10^5 t}\cos(1.71\times10^6) \\ &\quad - 0.039\,e^{8.45\times10^5 t}\sin(1.71\times10^6 t) \end{aligned}$$

For mode 2, the closed form solutions are

$$\begin{aligned} V_{C1} &= 0.005 + 25.6\,e^{-1.86\times10^6 t} - 23.2\,e^{9.05\times10^5 t}\cos(1.71\times10^6 t) \\ &\quad + 61.8\,e^{9.05\times10^5 t}\sin(1.71\times10^6 t) \\ V_{C2} &= -0.005 - 1.36\,e^{-1.86\times10^6 t} + 2.12\,e^{9.05\times10^5 t}\cos(1.71\times10^6 t) \\ &\quad + 2.62\,e^{9.05\times10^5 t}\sin(1.71\times10^6 t) \end{aligned}$$

We then define the following property: *"To remain in forward active and cutoff, V_{BC} must always be less than 0"*. From Figure 3.8 we can deduce that $V_{BC} = V_B - V_C = 0 - (V_{C1} + V_{C2}) = -V_{C1} - V_{C2}$. Thus $-V_{C1} - V_{C2} < 0$ or $V_{C1} + V_{C2} > 0$.

$$[V_{C1} + V_{C2} > 0]$$

This property is turned into a first-order formula as before. The MetiTarski input description for proving the property over mode 1 is shown in Figure 3.9.

```
fof(
  Colpitts,conjecture, ! [X] :
    (
      (0 <= X & X <= 1.66*10^-7) =>
        (0.001 - 2.67*exp(-1.93*10^6*X)
          + 2.67*exp(8.45*10^5*X)*cos(1.71*10^6*X)
          + 3.20*exp(8.45*10^5*X)*sin(1.71*10^6*X)) +
        (-0.0011 + 0.146*exp(-1.93*10^6*X)
          + 0.205*exp(8.45*10^5*X)*cos(1.71*10^6*X)
          - 0.039*exp(8.45*10^5*X)*sin(1.71*10^6*X))
      > 0
    )
).
```

Figure 3.9: MetiTarski Input for the Verification of Mode 1 of the Colpitts Oscillator

The two TPTP descriptions are provided to MetiTarski which subsequently proves the property over both modes of operation. The two proofs complete quite fast, in only a few seconds. See Table 3.5 for the example runtimes. In this case the two closed form solutions for the VC1 and VC2 were combined and then proved over mode 1 and mode 2 of the PWL model of the BJT.

3.4.3 Chua's Oscillator

The Chua's circuit [17], shown in Figure 3.10, presents similar complexities to the tunnel diode and Colpitts oscillators. This circuit demonstrates the behaviour of

Table 3.5: Colpitts Oscillator Verification Runtimes (in seconds)

Mode of Operation	
Colpitts-1-VC1+VC2	1.0
Colpitts-2-VC1+VC2	5.7

chaos, which is caused by the non-linear resistance N_R. If the value of the non-active circuit components are chosen properly, instead of chaos, the circuit will demonstrate a stable oscillation. Even though the non-linear resistor has three modes of operation, when the Chua's circuit exhibits stable oscillation, there is switching only between two modes. We want to prove that the oscillation is bounded for a set of parameters and initial conditions.

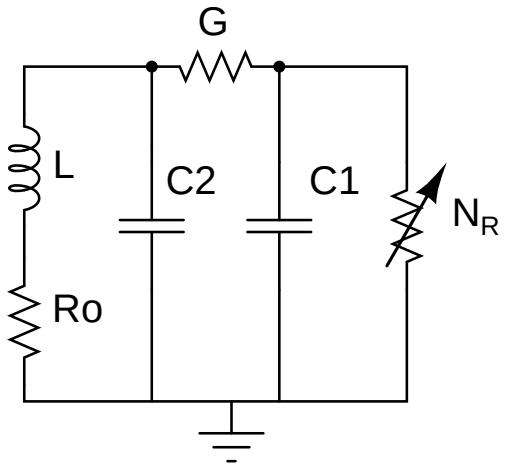


Figure 3.10: Chua's Circuit

The differential equations describing the behaviour of the circuit are

$$\dot{I}_L = \frac{V_{C2} - R_o I_L}{L}$$
$$\dot{V}_{C2} = -\frac{G(V_{C2} - V_{C1}) + I_L}{C_2}$$
$$\dot{V}_{C1} = \frac{G(V_{C2} - V_{C1}) - I_{NR}(V_{C1})}{C_1}$$

Here $I_{NR}(V_{C1})$ represents the PWL model of the non-linear resistor. This PWL model can be defined over three modes with $G_a = -0.757576$, $G_b = -0.409091$ and $V_e = 1$ that are used to represent the slope of the non-linear curve over the three modes of operation. The PWL model is further defined as,

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

As before, with the system of differential equations, the PWL model and the transition relations between each mode, the closed form solution of each state variable over each mode is solved using Maple.

For the circuit with, $R_o = 0.0125\,\Omega$, $L = 18$ H, $G = 0.5355$ S, $C_1 = 10$ F and $C_2 = 100$ F, we once again use a Laplace transform to solve the system of ODEs.

For mode 1 the closed form solutions are

$$V_{C1}(t) = 2.84 - 0.063\,e^{-0.019t} - 1.77\,e^{0.00024t}\cos(0.019t) + 0.689\,e^{0.00024t}\sin(0.0189t)$$
$$V_{C2}(t) = 0.0189 + 0.0077\,e^{-0.019t} - 0.183\,e^{0.00024t}\cos(0.019t) + 0.793\,e^{0.00024t}\sin(0.0189t)$$
$$I_L(t) = 1.51 - 0.023\,e^{-0.019t} - 2.35\,e^{0.00024t}\cos(0.019t) - 0.42\,e^{0.00024t}\sin(0.019t)$$

For mode 2 the closed form solutions are

$$V_{C1}(t) = 0.198\,e^{0.028t} + 0.8\,e^{-0.0058t}\cos(0.021t) - 0.882\,e^{-0.0058t}\sin(0.021t)$$
$$V_{C2}(t) = 0.02\,e^{0.028t} - 0.76\,e^{-0.0058t}\cos(0.0206t) + 0.15\,e^{-0.0058t}\sin(0.021t)$$
$$I_L(t) = 0.039\,e^{0.0277t} + 0.0864\,e^{-0.00575t}\cos(0.021t) - 2.02\,e^{-0.00575t}\sin(0.0206t)$$

The formal property that the oscillation remains bounded is

$$[V_C1 > 0 \wedge V_C1 < 5 \wedge V_C2 > -2 \wedge V_C2 < 2 \wedge L_I > -2 \wedge L_I < 5]$$

We create a first-order formula for each bound and provide them to MetiTarski. For half of them, a proof is returned in under 10 seconds. For the rest, it was necessary to include the axioms for the extended solutions for the trigonometric and exponential functions. Once the extra axioms were added, MetiTarski was able to complete the proof, although requiring a minute or more. Runtime results of the experiments can be found in Table 3.6.

Table 3.6: Chua's Oscillator Verification Runtimes (in seconds)

Chua's Circuit - P4	
Chua-1-VC1-L	37.0
Chua-1-VC1-U	62.2
Chua-2-VC1-L	3.0
Chua-2-VC1-U	4.5
Chua-1-VC2-L	99.7
Chua-1-VC2-U	25.5
Chua-2-VC2-L	3.3
Chua-2-VC2-U	4.8
Chua-1-IL-L	17.0
Chua-1-IL-U	27.0
Chua-2-IL-L	5.5
Chua-2-IL-U	1.4

3.4.4 MOSFET Circuit

In this example we demonstrate the power of MetiTarksi in automatically solving inequalities containing analytical functions. Take for instance the MOSFET circuit Figure 3.11 [56].

One common verification question is "What mode of operation is the MOSEFT operating?" or "For what widths (W) and lengths (L) will the transistor remain out of cutoff?".

Consider the following circuit parameters: $I_D = 0.4$ mA, $\mu_n C_{ox} = 100\mu\text{A}/V^2$, L = 1 μm, W = 32 μm and $V_{th} = 0.7V$. Where L is the legnth of the transistor, W

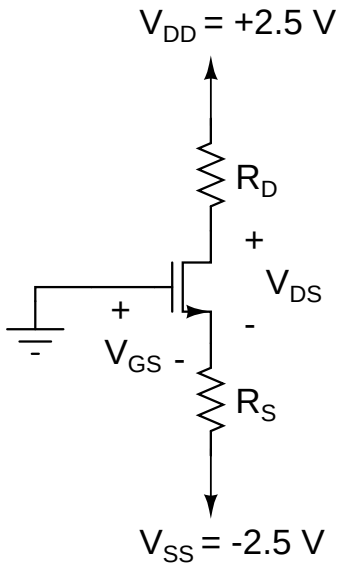


Figure 3.11: Basic MOSFET Circuit [56]

is the width, I_D is the current through the drain, $\mu_n C_{ox}$ is the process transconductance (determined by the technology used to fabricate the MOSFET) and V_{th} is the threshold voltage. If we neglect the channel-length modulation [56] we can perform the following hand DC analysis of the transistor.

First we assume that the transistor is operating in the saturation region, so we can use the square-law equation for operation in the saturation region.

$$I_D = \frac{1}{2}\mu_n C_{ox}\frac{W}{L}(V_{GS} - V_t)^2$$

Solving for V_{GS}, which is the MOSFET gate to source voltage, we have

$$\begin{aligned} V_{GS} &= \sqrt{\frac{2L \times I_D}{W\mu_n C_{ox}}} + V_{th} \\ &= \sqrt{\frac{2 \times 20^{-6} \times 0.4 \times 10^{-3}}{32 \times 10^{-}6 \times 100 \times 10^{-6}}} + 0.7 \\ &= 1.2V \end{aligned}$$

From the circuit diagram, we know that V_G is grounded. So therefore

$$V_{GS} = V_G - V_S = -V_S = 1.2 \text{ V}$$

With the circuit voltages solved for, we can now determine if the MOSFET is operating in the saturation region. The conditions are:

- $V_{GS} \geq V_{th}$ and
- $V_{DS} \geq V_{GS} - V_{th}$

The first condition is true, so we know that the MOSFET is not in cutoff. Since we know that the I_D is 0.4mA we can solve for the value of R_D to ensure that the MOSFET is in saturation.

$$\begin{aligned}
V_{DS} &\geq V_{GS} - V_{th} \\
V_D - V_S &\geq V_G - V_S - V_{th} \\
V_D &\geq V_G - V_{th} \\
R_D &= \frac{V_{DD} - V_D}{I_D} \\
V_D &= V_{DD} - R_D I_D \\
V_{DD} - R_D I_D &\geq V_g - V_{th} \\
-R_D I_D &\geq V_g - V_{th} - V_{DD} \\
-R_D &\geq \frac{V_g - V_{th} - V_{DD}}{I_D} \\
R_D &\leq -\frac{0 - 0.7 - 2.5}{0.4mA} \\
R_D &\leq 8000\,\Omega
\end{aligned}$$

From the derivations we have deduced that for the MOSFET to remain in saturation R_D must be less than or equal to a resistance of 8 $k\Omega$. Now this is for a specific choice of W, L, I_D, R_S and V_{th}. If it is necessary to change any of the parameters or even take into account a range of error in parameter values then the

analysis by hand must be redone. In the case of this trivial example, the time to redo the calculations for multiple parameters is not that great. Consider though, the re-derivations for a multi-transistor circuit.

We will now show how we can fully automate this procedure with MetiTarksi. Starting from the same equation as before solving for V_{GS} and considering the ratio of the width and length of the transistor as W/L = r then we have,

$$\begin{aligned} V_{GS} &= \sqrt{\frac{2L \times I_D}{W \mu_n C_{ox}}} + V_{th} \\ &= \sqrt{\frac{2 \times 0.4 \times 10^{-3}}{r \times 100 \times 10^{-6}}} + 0.7 \\ &= \sqrt{\frac{8}{r}} + 0.7 \\ &= 2\sqrt{\frac{2}{r}} + 0.7 \end{aligned}$$

To prove that the circuit above never enters into cutoff for a range of W/L ratios between 10 and 40, we supply MetiTarski with the description shown in Figure 3.12.

```
fof(
  Mosfet,conjecture, ! [r] :
    (
      (10 <= r & r <= 40)
        => 2*sqrt(2*r^-1) + 0.7 > 0.7
    )
   )
.
```

Figure 3.12: MetiTarski Input for Verifying the Mode of Operation

In this case we are checking that for a range of W/L ratios will V_{GS} always be greater than V_t. MetiTarski returns a proof of this inequality. This example

displays the power of the tool, but we can do even better. Take for instance the revised version (see Figure 3.13) of the MetiTarski description for the DC mode of operation

```
fof(
  Mosfet,conjecture, ! [R,VTH] :
    (
      (10 <= R & R <= 40 & 0.65 <= VTH & VTH <= 0.85)
        => 2*sqrt(2*R^-1) + 0.7 > VTH
    )
  )
.
```

Figure 3.13: Revised MOSFET Circuit MetiTarski Input

Here we are using two variables to take into account process variations of the threshold voltage between 0.65V and 0.85V. MetiTarksi returns a proof for the inequality indicating that within the tolerances specified, the transistor will not enter cutoff. The experimental results are located in Table 3.7, MOSDC1 is the experiment with one quantifier and MOSDC2 is the experiment with two quantifiers. The MOSDC2 proof takes a little bit longer to complete because of the extra quantified variable.

Table 3.7: MOSFET Circuit Verification Runtimes (in seconds)

MOSFET Circuit	
MOSDC1	0.08
MOSDC2	0.11

3.5 Summary

In this chapter we described in detail a method for using the MetiTarski theorem prover to verify functional properties of analog circuits. The input to MetiTarski is

a closed form solution of the ODEs and therefore, a PWL model is employed to be able to generate it. It is also possible to directly use a circuit level equation with the tool, to prove properties about the DC bias point of transistors. One problem with the MetiTarski tool and theorem proving in general is that in the case that the proof cannot be validated, there is no way of knowing where the problem lies. It could be in fact a correctly determined error in the design under verification, but also it could potentially be a problem within MetiTarski concerning internal algorithms. As well, we can only verify properties that always hold true. In the next chapter a verification method based on predicate abstraction will be presented where properties can be defined using Linear Temporal Logic. This allows more complex properties that could be defined as eventually happening. As well, the ODEs can be used exactly as extracted with no conversion to a closed form solution needed.

Chapter 4

Verification by Predicate Abstraction

4.1 Introduction

In this chapter, we describe a verification methodology that combines predicate abstraction, constraint solving and symbolic model checking to verify properties on the ODEs extracted from the bond graph models. In contrast to the previous chapter where we use inequalities to describe properties in MetiTarski, we now use Linear Temporal Logic (LTL) to describe properties. This is particularly useful for defining behaviour that could *eventually* happen instead of being constrained to behaviour that is *always* occurring.

The problem with traditional model checking is that there is an explicit representation of each possible state of the model under verification. This can lead to a state space explosion, which we have mentioned earlier. Symbolic model checking [10] addresses this problem by implicitly representing states, thus reducing the complexity of the verification. Further optimization of the state space division is possible by using abstractions. Predicate abstraction [34], is one of the most successful abstraction approaches for the verification of systems with an infinite state

space. By dividing the state space into a finite number of regions and then defining the transition relations between each discrete state, the verification problem can then be solved using model checking methods and tools. Then, if a counter-example is produced, further predicates can be generated to refine the abstract state space.

In [6], Alur, Dang and Ivancic extend predicate abstraction to the verification of hybrid systems. In their work they developed algorithms and tools for the reachability analysis of hybrid systems by combining predicate abstraction with polyhedra for approximating the reachable states of the system. This work is similar to that of Tiwari [62], that led to the development of the HybridSal abstractor [63] which automatically generates the discrete state space of hybrid models using abstraction techniques.

4.2 Preliminaries

4.2.1 Property Definition

The big question in formal verification is how to properly define and choose the properties to verify. Temporal logics [26], define a system for describing in a formal manner the truth of some statement that varies over time. In this thesis we will use LTL (Linear Temporal Logic) [42] that indicates that the events will all occur on the same timeline. This is in contrast to branching temporal logics that indicate that events can happen on many possible timelines, which is similar to a tree structure.

In most cases, we want to ensure that the system that has been designed is dependable [42]. This only raises another question on how to measure the level of dependability. The strongest form, where all catastrophic events are avoided is called a "safe" state. A safety property will define a condition that ensures some event (bad or otherwise) will never occur. This is formally defined by stating that any member from the set of undesired states will never be reached [28]. For example an analog oscillator safety property can be defined as "For the set of parameters L,R

and C the circuit will not oscillate" or "For a set of initial conditions, the circuit will never stop oscillating".

LTL is a logic language that defines properties by qualitatively describing their truth over time. There are four basic temporal operators available as described in Table 4.1. Safety properties can be defined using any combination including the G (always) operator.

Table 4.1: LTL temporal operators

Fp	"eventually p"
Gp	"always p"
Xp	"next time p"
pUq	"p until q"

4.2.2 HybridSal Abstractor

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 [62]. An example HybridSAL description is shown in Figure 4.1.

This is one of the examples included in the HybridSal tutorial package [63]. It represents a simple hybrid model of a thermostat. The continuous values are x and its derivative xdot, that represent the current temperature and the change in temperature. The discrete variable is called "state" and represents whether the thermostat is on or off. The initial value of x is between 70 and 80 degrees Fahrenheit. The formulas in the TRANSITION section describe the conditions for switching between

```
SimpleThermo3:CONTEXT =
BEGIN
control : MODULE =
 BEGIN
 LOCAL state : BOOLEAN
 LOCAL x : REAL
 LOCAL xdot : REAL
 INVARIANT
        TRUE
 INITFORMULA
        70 <= x AND x <= 80 AND state = TRUE
 TRANSITION
        [
        state = TRUE AND x >= 80 -->
                state' = FALSE
        []
        state = FALSE AND x <= 70 -->
                state' = TRUE
        []
        state = TRUE AND x < 80 -->
                xdot' = (100 - x)
        []
        state = FALSE AND x > 70 -->
                xdot' = 0 - x]
 END;
 G( ss:[ control.STATE -> BOOLEAN ] ) : [ control.STATE -> BOOLEAN ];
 correct: THEOREM
        control |- G( 70 <= x AND x <= 80 );
END
```

Figure 4.1: Example HybridSal Description

states as well as the differential equations defined over each mode of operation. The second to last line contains the property to be verified, defined using LTL. In this case, the condition is that the temperature will always be between 70 and 80.

4.2.3 SAL-Symbolic Model Checker

The model checking problem is defined as: given a finite state machine M and a temporal logic formula p, does M define a model of p. [42]. More explicitly, do all the transitions from some initial state to the next state of M satisfy the temporal property p. By exhaustively checking all the paths of M, the property can be verified. The problem is that if the number of states is too large, then the memory required to check all the paths becomes large as well and the model checking algorithm can fail. Symbolic model checking methods have been developed to address this problem.

Symbolic Model checking uses Reduced Ordered Binary Decision Diagrams (ROBDDs), which are computational efficient structures that encode states and transition relations. By explicitly representing the states using ROBDDs, the number of states that can be visited using model checking is increased from 10^8 to about 10^{20} [10]. The SAL-SMC uses a similar algorithm for the symbolic model checking of properties, the algorithm can be found in [30].

4.2.4 HSolver Constraint Solver

Constraint solving 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. The simplest example of a constraint solving problem can be stated as follows: Given a fixed amount of money and a number of required items to purchase, maximize the number of objects that can be acquired.

To solve the stated constraints the HSolver tool [54] first decomposes the state space into hyper-boxes. Interval arithmetic is then used to check the flow on the boundary between neighboring boxes. This is done via an abstraction refinement procedure in order to achieve precise results [53]. An example input description to Hsolver as shown in Figure 4.2.

```
VARIABLES [ x, y ]
MODES [ m1,m2 ]
STATESPACE
  m1[[0, 4], [0, 4]]
  m2[[0, 4], [0, 4]]
INITIAL
  m1{x>=2.5/\x<=3/\y=0}
FLOW
  m1{x_d=x-y}{y_d=x+y}
  m2{x_d=-2*y}{y_d=x-y}
JUMP
  m1->m2{[x>=0.03]/\[i'=i]}
  m2->m1{[x<0.03]/\[i'=i]}
UNSAFE
m1{x<=2}
```

Figure 4.2: Example HSolver Description

In this case the model is defined over two modes of operation. The variables of the system are x and y. The modes of the system are named m1 and m2. In the state-spaces of mode 1 and mode 2, the variables x and y both can vary between 0 and 4. The initial value of x is within the range between 2.5 and 3 and the initial value of y is defined as a constant value of zero. The differential equations are defined over the discrete modes m1 and m2 and they have a _d appended to their variable names. The jump conditions define when the system switches modes and in this example it depends on whether x is greater than or less than 0.03. The safety constraint on the system is that the variable x must always be greater than 2.

4.2.5 Predicate Abstraction

Predicate abstraction has been developed for the efficient reduction of an infinite state space to one that can be verified using model checking techniques [20]. The first task in predicate abstraction is the generation of predicates and then the initial generation of an abstract model. Several advanced methods exist [62], where the

derivative of the ODEs is repeatedly taken to generate new predicates. Once the abstract model is created, model checking is used to attempt to prove the property of interest. If a counter example is generated, then further predicates must be added to refine the abstract model.

4.3 Verification Methodology

We propose using a predicate abstraction approach for the verification of LTL properties on the system of ODEs that were automatically extracted from a bond graph model. HybridSal requires as an input the representation of the analog circuit as a system of ODEs, the initial conditions and the temporal property of interest. The state space is subsequently partitioned based on predicates that are extracted from the model and from the property of interest by HybridSal's abstraction algorithm. The SAL-SMC is then applied on the abstract state space to verify the property of interest. When the property cannot be validated, it is possible that the over approximation of the abstract model has led to a false negative counterexample. As the generated counterexample is an abstract one it is essential to validate it using the HSolver constraint solver. In case of this error, we iteratively remove regions violating the property and refine the model for verification again. In the case that HSolver does not prove a false negative counter example, extra predicates can be added to generate a refined abstract model that can again be passed through the steps of the methodology. The proposed verification methodology is illustrated in Figure 4.3.

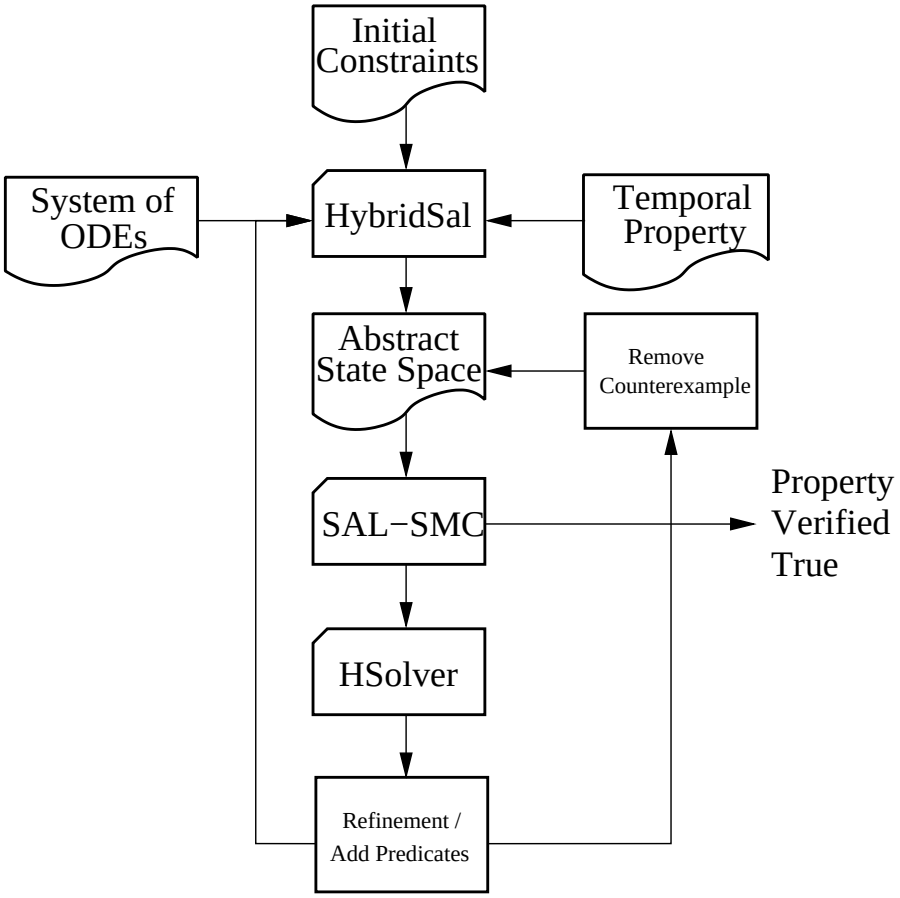


Figure 4.3: Predicate Abstraction Based Verification

4.4 Applications

4.4.1 Tunnel Diode Oscillator

Recall that the bond graph extracted simplified ODEs (Section 2.5.4) are given as $\dot{V_C} = \frac{1}{C}(-I_d(V_C) + I_L)$ and $\dot{I_L} = \frac{1}{L}(-V_C - \frac{1}{G}I_L + V_{in})$

Consider the tunnel diode circuit with the set of parameters C = 1 uF, L = 1 uH, G = 2000 mS, Vin = 0.3 V and the initial values $V_C = 0.131$ V, $I_L = 0.055$ A. We want to verify that these combinations of parameters and initial conditions do not allow the circuit to oscillate. The behaviour in question is stated as the safety property $G\ (v \leq 0.6)$. The validation of the property ensures the non-existence of

oscillation.

Once the simplified system of ODEs has been extracted 2.5.4, they can be used to form a hybrid system definition in the HybridSal modeling language, as in Figure 4.4. 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.

```
TunnelDiode:CONTEXT =
BEGIN
control : MODULE =
 BEGIN
 LOCAL v : REAL
 LOCAL vdot : REAL
 LOCAL i : REAL
 LOCAL idot : REAL
 INVARIANT
        TRUE
 INITIALIZATION
        v = 131/1000  <-- Inital Values
        i = 55/1000
TRANSITION                <-- System of ODEs
[
   v > 0  -->
   vdot' = 1000*(-1*(v*v*v-15/10*v*v+6/10*v) + i);
   idot' = (-v - 50*i + 3/10)
]
 END;
 G( ss:[ control.STATE -> BOOLEAN ] ) : [ control.STATE -> BOOLEAN ];
 correct: THEOREM
        control |- G(  v < 6/10 ); <-- Property to be Verified
END
```

Figure 4.4: HybridSal Tunnel Diode Description

We then use the HybridSal tool to generate the discrete abstract model illustrated in Figure 4.5. This abstract model is the model checked using SAL-SMC to

verify the non oscillation property.

```
Abstract variable to Polynomial Mapping:
g2 --> v
g1 --> v - 3/5
g0 --> -1*v^3 + 3/2*v^2 - 3/5*v + i

TunnelDiodeABS: CONTEXT =
 BEGIN
 SIGN: TYPE = {pos, neg, zero};
  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
```

Figure 4.5: SAL Description for the Abstract Model of the Tunnel Diode Circuit

In this case, the SAL-SMC tool returns that the property is not proved and gives a counterexample (see Figure 4.6). In the counterexample, the abstract values that the predicates take are shown. In this verification problem, 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 part of the real system or an artifact of the

abstraction process (spurious).

```
  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
------------------
```

Figure 4.6: SAL-SMC Generated Counterexample from the SAL Code in Figure 4.5

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 4.7, we can perform a more precise examination of the reachable states. If it is determined that the counterexample is never reached, then the false transitions can be

removed from the abstract model.

```
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]} ⟵ Transition Predicates
m2->m3{v*v*v+1.5*v*v-0.6*v+i>0/\[i'=i/\v'=v]}
m3->m4{v-0.6=0]/\[i'=i/\v'=v]}
UNSAFE
m4{v>=0.6}  ⟵ Possible spurious transition
```

Figure 4.7: HSolver Code for the Counterexample Validation of Figure 4.6

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 4.5 all transitions from states where predicate $g1 = neg$ holds to states where $g1 = zero$ holds. This refinement is valid because the $g1$ predicate depends only on $g0$ and not $g2$ through the abstract function ASSVP(g1, g0) in the HybridSal abstraction. This is the reason why the jump conditions implemented in the HSolver code are based solely on the $g0$ and $g1$ predicates. The verification on the refined SAL code using SAL-SMC in that case succeeds, which means that no oscillation occurs which indicates that the model parameters we have chosen are incorrect.

4.4.2 Colpitts BJT Oscillator

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.

Consider the BJT based Colpitts oscillator shown in Figure 3.8. When oscillating, the BJT will never go into its saturation region. In fact, the BJT will either be in the cut-off mode or forward active modes [41]. 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 $Vth = 0.75$. For instance, the property that no transition occurs from Forward active to Saturation, can be validated by proving that $\mathrm{G}(V_{C_1} + V_{C_2}) > 0$ is True, where V_{C_1} and V_{C_2} are voltages across the capacitors C_1 and C_2 and G is the "alway" LTL operator.

Recall that the differential equations describing the behaviour of the BJT Colpitts oscillator are

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

Also recall that 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, $I_s = 1.43 \times 10^{-14}$, $R_{ON} = 100\,\Omega$. With the ODEs and the circuit parameters we can construct the HybridSal description containing the description of the system (see Figure 4.8).

```
INITIALIZATION
       vc1 = 1;
       vc2 = 35/100;
       iL = 1
TRANSITION
       [
       vc2 >= 75/100 -->
              vc1dot' = (500000000/27*iL);
              vc2dot' = (500000000/27*iL-1250000/27*vc2);
              iLdot' = (-35*iL-10200*vc2-10200*vc1)
       []
       vc2 < 75/100 -->
              vc1dot' = (50000000/27*iL+1000000000/27);
              vc2dot' = (50000000/27*iL-6250000/27);
              iLdot' =  (-1*35*iL-10200*vc2-10200*vc1)
       ]
END;
 G( ss:[ control.STATE -> BOOLEAN ] ) : [ control.STATE -> BOOLEAN ];
 correct: THEOREM
       control |- G( 0 < (vc2+vc1)); ⟵ Property of Interest
```

Figure 4.8: Partial Colpitts Oscillator HybridSal Description

With the hybrid system described using the HybridSal syntax. We can run the abstractor algorithm. The generated abstract state description contains the predicates and abstract transition functions as shown in Figure 4.9.

Now we take the abstract description and pass it to the SAL-SMC. As expected a counter example is generated (see Figure 4.10). We then convert the predicates from Figure 4.9 into constraints. As well we express the counterexample

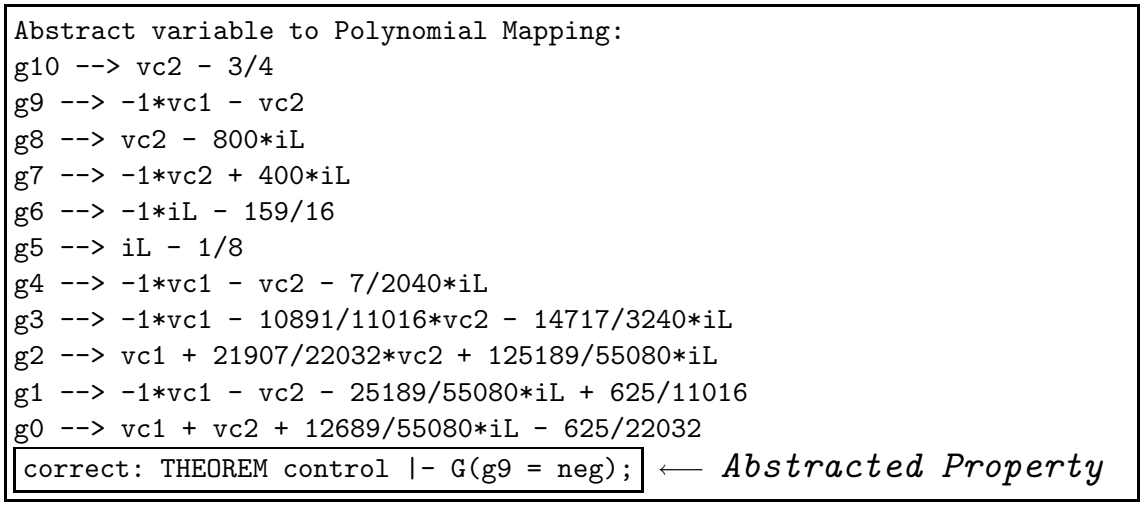


Figure 4.9: Predicates from the abstract Model of the Colpitts Oscillator

path in terms of transitions in the HSolver format (see Figure 4.11). By removing those predicates that do 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 counter example are then removed from the abstract model and then model checking is applied again. A second counter-example is produced. Following the same methodology described before, the counterexample transitions are used to construct a HSolver description. This time though, Hsolver returns "safety unknown".

At this point, we must apply the predicate abstraction step of the methodology. Using HybridSal we generate three more predicates to refine the abstract model. Then, we run SAL-SMC on the refined abstract model and the property is returned true. This indicates that for all time the transistor will remain out of cutoff and reverse-active, staying in the expected mode of operation.

4.5 Summary

In this chapter, a verification methodology based on predicate abstraction and a counter-example verification using constraint solving was presented. One of its benefits over the theorem proving method is that a full counterexample is generated. This is of primary benefit when trying to discover bugs in a design. When the theorem prover does not complete, it could either mean that the inequality is false, or on the other hand that the bounds are too large to solve. The drawback of the predicate abstraction methodology is the abstractions themselves that create a loss of precision that sometimes is not negligible. There are cases when even adding additional predicates to generate the abstract model accomplishes no gains in verification. The amount of time to go through the abstraction refinement and counterexample refinement is much greater than what is encountered in our theorem

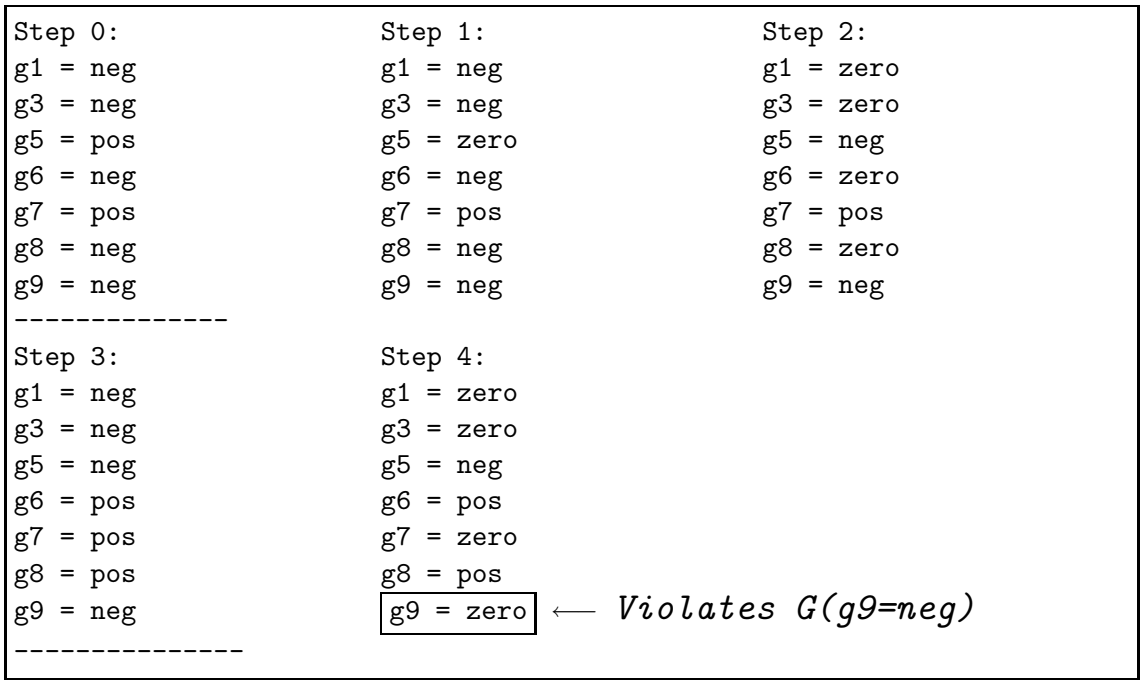


Figure 4.10: SAL-SMC Generated Counterexample for the Colpitts Oscillator

proving methodology. Overall the predicate abstraction methodology is promising since the steps are mechanical in nature and could be automated.

```
VARIABLES [vc1,vc2,iL]
MODES [m1,m2,m3,m4,m5]
STATESPACE
m1[[-1,2],[-1,2],[-1,2]]
m2[[-1,2],[-1,2],[-1,2]]
m3[[-1,2],[-1,2],[-1,2]]
m4[[-1,2],[-1,2],[-1,2]]
m5[[-1,2],[-1,2],[-1,2]]
INITIAL
  m1{[vc1=1/\vc2=0.35/\iL=1]}
FLOW
m1{vc1_d=18518518*iL+37037037}{vc2_d=18518518*iL-23148}
        {iL_d=-35*iL-10200*vc2-10200*vc1}
m2{vc1_d=18518518*iL+37037037}{vc2_d=18518518*iL-23148}
        {iL_d=-35*iL-10200*vc2-10200*vc1}
m3{vc1_d=18518518*iL+37037037}{vc2_d=18518518*iL-23148}
        {iL_d=-35*iL-10200*vc2-10200*vc1}
m4{vc1_d=18518518*iL+37037037}{vc2_d=18518518*iL-23148}
        {iL_d=-35*iL-10200*vc2-10200*vc1}
m5{vc1_d=18518518*iL+37037037}{vc2_d=18518518*iL-23148}
        {iL_d=-35*iL-10200*vc2-10200*vc1}
JUMP
m1->m2{[-vc1-vc2-0.46*iL+0.0567<0]/\[-vc1-0.988*vc2-4.54*iL<0]/\
        [iL-0.125=0]/\[-iL-9.93<0]/\[-vc2+400*iL>0]/\[vc2-800*iL<0]/\
        [-vc1-vc2<0]/\[iL'=iL/\vc1'=vc1/\vc2'=vc2]}
m2->m3{[-vc1-vc2-0.46*iL+0.0567=0]/\[-vc1-0.988*vc2-4.54*iL=0]/\
        [iL-0.125<0]/\[-iL-9.93=0]/\[-vc2+400*iL>0]/\[vc2-800*iL=0]/\
        [-vc1-vc2<0]/\[iL'=iL/\vc1'=vc1/\vc2'=vc2]}
m3->m4{[-vc1-vc2-0.46*iL+0.0567<0]/\[-vc1-0.988*vc2-4.54*iL<0]/\
        [iL-0.125<0]/\[-iL-9.93>0]/\[-vc2+400*iL>0]/\[vc2-800*iL>0]/\
        [-vc1-vc2<0]/\[iL'=iL/\vc1'=vc1/\vc2'=vc2]}
m4->m5{[-vc1-vc2-0.46*iL+0.0567=0]/\[-vc1-0.988*vc2-4.54*iL=0]/\
        [iL-0.125<0]/\[-iL-9.93>0]/\[-vc2+400*iL=0]/\[vc2-800*iL>0]/\
        [-vc1-vc2=0]/\[iL'=iL/\vc1'=vc1/\vc2'=vc2]}
UNSAFE
  m5{vc1+vc2<0}
```

Figure 4.11: HSolver Counterexample Validation of the Colpitts Oscillator

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis the primary goal was to develop a complete automated methodology for the modelling and formal verification of analog circuits. We decided to first focus at the modelling stage, which is a source of many design problems. By using bond graphs the automatic extraction of the system of equations was possible. As well, bond graphs provide an explicit description of the system, which aids in the design and specification of properties to verify. Next we presented two methods for formally verifying properties over the extracted ODEs. One method was based on predicate abstraction and symbolic model checking, the other on theorem proving. Each method has its advantages and disadvantages and the choice of which one to use lies in the specific application and the verification properties of interest. Overall, we have shown that the methodologies require user involvement, but this involvement is mechanical in nature and therefore could be automated under one complete framework. The difficulty of the future work will be on interfacing the different languages and tools together.

Bond graphs provide and efficient means for modelling analog circuits. We

have presented an example of a tunnel diode oscillator that was successfully translated into a bond graph, and had its ODEs automatically extracted. Since bond graphs are domain independent and object oriented, models can be constructed at several levels of abstraction. We found that this can reduce the complexity of the system equations that are extracted by the Dymola Modelling Environment. As more components are added to a model, as in the case of the bond graphs for BJTs or MOSFETs, reliance on algebra systems such as Mathematica increases because Dymola lacks the re-writing techniques to simplify the extracted system of equations. This remains an open issue for future work.

The theorem proving methodology can be summarized as follows. Using a system of ODEs that has been extracted from a circuit using bond graphs, they are first converted into their PWL equivalent model. Then using the algebra system Maple, the closed form solution of each mode of the PWL model is solved via Laplace transforms. Then the verification property is turned into an inequality over the closed form solution and formatted using the TPTP syntax. MetiTarski is then used to automatically generate a proof from the TPTP description. The advantage of using MetiTarski is that its verification algorithm is fully automated. As long as a property can be described using an inequality, MetiTarski can process it. Another advantage is that MetiTarski can solve inequalities that contain trigonometric functions and exponentials. One major disadvantage is that if we are trying to solve properties defined over ODEs, it is necessary that the system of equations be linear or transformed into their linear form, so that a closed form solution can be computed. This is a very constraining requirement that can have a severe effect on the accuracy of the verification. As well, if MetiTarski does not terminate, the verifier is left in the dark. It could mean that the inequality is false, but it could equally mean that the function under test takes on a value that is out of range for the decision procedure. Unfortunately, in both these cases, MetiTarski will run until manually terminated. Although when successful, MetiTarski will produce a full blown proof of

its claims. We successfully verified functional properties of a tunnel diode, Chua's Circuit and Colpitts oscillator using the MetiTarski theorem prover. As well, we showed how the hand analysis of a MOSFET could be formally verified using the tool.

The predicate abstraction methodology can be summarized as follows. Again, using the system of ODEs that has been extracted from the circuit bond graphs, the HybridSal abstractor successfully takes the continuous state space and splits it up into an abstract one. Subsequently, it determines the transitions between the abstract states. The SAL symbolic model checker is then used to verify the abstract property. If a counterexample is generated then it is verified using constraint solving. If the counterexample cannot be eliminated, then using HybridSal, we generate additional predicates to refine the abstract model and redo the symbolic model checking. The generation of the counterexample is the main advantage to this method, since the error in the model (the bug) is explicitly stated and can be easily isolated and thus removed. Secondly, the HybridSal abstractor allows properties to be defined using LTL, enabling more detailed properties to be defined. One downside, is that HybridSal can only create abstractions from ODEs that are polynomial. It does not understand any special functions such as sin, cos, ln and exp, which limits the type of problems that can be verified. We successfully verified properties of a tunnel diode and Colpitts oscillator using the predicate abstraction and symbolic model checking methodology. The summary of the two verification methods is shown in Table 5.1.

5.2 Future Work

The main direction for any future work from this thesis will be to extend the methods to the analog and mixed signal domain. In particular, we have emphasized many

Table 5.1: Comparison of the Verification Methods

Method	**Theorem Proving**	**Predicate Abstraction**
Tool	MetiTarski	HybridSal + SAL-SMC + HSolver
Property Definition	Inequalities	LTL
Input	Closed Form Solution	ODEs
Type	Analytical (sqrt, sin, cos, etc.)	Polynomials
Positive Aspects	Complete Proof Generated. Supports a large set of analytical functions.	More detailed property specification. Counter-example generated to locate bugs.
Negative Aspects	No information provided when proof fails. Closed form solutions difficult to determine for non-linear systems. Can only define basic properties.	Over-approximation leads to invalid counterexamples that cannot be validated. No trigonometric or exponential functions allowed.

times that bond graphs are domain independent, which allows the analog part described in Chapter 2 to be connected to block diagrams representing the digital part of an AMS circuit. As well, there has been recent development on a new bond graph component called a "switched bond". This component could be combined with our methodology for modelling circuits where switching occurs. This could be particularly useful for verifying AMS designs.

When defining properties in HybridSal, we are limited to using Linear Temporal Logic to define properties. It would add flexibility to the methodology if properties could be defined using CTL (Computational Tree Logic). The Symbolic Analysis Laboratory which HybridSal is build on top of, has a framework that could be used to construct a CTL extension to HybridSal. The methodology could also be improved by applying different abstraction algorithms to the state space. Currently, there are several algorithmic methods that have not been implemented in a tool such as HybridSal.

For the theorem proving methodology, it will be interesting to work on implementing an automatic PWL model generator. It will also be beneficial to investigate methods to obtain approximate closed form solutions to a system of non-linear ODEs. The Singer-Prelle [24] method is one such technique that can solve certain types of first order non-linear differential equations. This will eliminate the need for a piecewise linear function. Since the development of MetiTarski is on going, it will be necessary to continuously update the methodology to account for these changes. In particular, the efficiency of the axioms will be increased, which will lead to a quicker and more reliable verification.

One area that we did not cover in this thesis is concerning the quality of the verification between the two methods. In both the model checking and theorem proving methodologies, we make decisions in modelling to allow the verification to complete. Future work should include research on quantifying the precision of the formal methods. This would enable the designer to choose the best and most efficient method for their needs.

Lastly, it will be necessary to expand the repertoire of applications under consideration. For instance, more complicated circuits such as the feed forward ring oscillator from RAMBUS [45] pose a challenge for the current formal verification methodologies. As well, work will also need to be done to define formal properties for the frequency domain. This is a realistic goal since many frequency transfer functions can be used directly with the methodologies we have proposed.

Bibliography

[1] "Maple 12 : The essential tool for mathematics and modelling." [Online]. Available: `http://www.maplesoft.com/`

[2] "SystemC-AMS : Analog and mixed-signal exstension to SystemC." [Online]. Available: `http://www.systemc-ams.org/`

[3] "Wolfram Mathematica 7 : Core Language." [Online]. Available: `http://reference.wolfram.com/`

[4] B. Akbarpour and L. C. Paulson, "MetiTarski: An automatic prover for the elementary functions," in *Intelligent Computer Mathematics*, vol. LNCS 5114. Springer, 2008, pp. 217–231.

[5] ——, "Applications of MetiTarski in the verification of control and hybrid systems," in *Hybrid Systems: Computation and Control*, vol. LNCS 3414. Springer, 2009, in press.

[6] R. Alur, T. Dang, and F. Ivancic, "Reachability analysis via predicate abstraction," in *Hybrid Systems: Computation and Control*, vol. LNCS 2289. Springer, 2002, pp. 35–48.

[7] K. Besbes, "Modelling semiconductor devices using bond graph techniques," in *Proc. IEEE International Symposium on Industrial Electronics*, 1997, pp. 201–205.

[8] F. Broenink, "Introduction to physical systems modelling with bond graphs," University of Twente, Tech. Rep., 1999.

[9] C. W. Brown, "QEPCAD B: a program for computing with semi-algebraic sets using CADs," *SIGSAM Bulletin*, vol. 37, no. 4, pp. 97–108, 2003.

[10] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Information and Computation*, vol. 98, no. 2, 1992.

[11] B. Caviness and J. Johnson, *Quantifier Elimination and cylindrical Algebraic Decomposition*. Spinger, 1998.

[12] F. E. Cellier and A. Nebot, "The modelica bond graph library," Swiss Federal Institute of Technology, Tech. Rep., 2007.

[13] F. Cellier, C. Clauss, and A. Urquia, "Electronic circuit modelling and simulation in modelica," in *Proc. Eurosim Congress on Modelling and Simuulation*, vol. 2, 2007, pp. 1–10.

[14] H. Chang and K. Kundert, "Verification of complex analog and RF IC designs," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 622–639, 2007.

[15] W. K. Chen, *The Circuits and Filters Handbook*. CRC Press LLC, New York, 2006.

[16] E. Christen and K. Bakalar, "VHDL-AMS A hardware description language for analog and mixed-signal applications," *IEEE Transaction on Circuits and Systems II: express Briefs*, vol. 46, no. 10, pp. 1263–1272, 1999.

[17] L. O. Chua, "Chua's circuit : An overview ten years later," *Journal of Circuits, Systems and Computers*, vol. 4, pp. 117–159, 1994.

[18] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

[19] T. Dang, A. Donze, and O. Maler, "Verification of analog and mixed-signal circuits using hybrid system techniques," in *Formal Methods in Computer-Aided Design*, vol. LNCS 3312. Springer, 2004, pp. 14–17.

[20] S. Das and D. L. Dill, "Counter-example based predicate discovery in predicate abstraction," in *In Formal Methods in Computer-Aided Design*, 2002.

[21] Dassault Systemes, "The dymola modelling laboratory." [Online]. Available: `http://www.dymola.com/index.htm`

[22] T. Dastidar and P. Chakrabarti, "Verification system for transient response of analog circuits using model checking," in *Proc. IEEE International Converence on VLSI*, 2005, pp. 195–200.

[23] M. Dempsey, "Dymola for multi-engineering modelling and simulations," in *Proc. IEEE Vehicle Power and Propulsion Conference*, 2006, pp. 1–6.

[24] L. Duarte, L. Mota, and J. Skea, "Solving second order differential equations by extending the ps methods," *Journal of Physics A. Mathematical and General*, vol. 34, 2001.

[25] H. Elmqvist and M. Otter, "Methods for tearing systems of equations in object oriented modelling," in *Proc. European Simulation Multiconference*, 1994, pp. 326–332.

[26] E. A. Emerson, *Temporal Logics in Handbook of Theoretical Computer Science*. Elsevier, 1990.

[27] Y. E. Fattah, "Constraint logic programming for structure-based reasoning about dynamic physical systems," *Artificial Intelligence in Engineering*, vol. 1, pp. 253–264, 1996.

[28] G. Frehse, B. Krogh, and R. Rutenbar, "Verifying analog oscillator circuits using forward/backward abstraction refinement," in *Proc. IEEE Design, Automation and Test in Europe*, 2006, pp. 257–262.

[29] F.-I. fr Techno-und Wirtschaftsmathematik, "Analoginsydes : The intelligent symbolic design system for analog circuits." [Online]. Available: `http://www.analog-insydes.de/`

[30] P. Gastin and D. Oddoux, "Fast ltl to büchi automata translation," in *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*. Springer-Verlag, 2001.

[31] P. Gawthrop and G. Bevan, "Bond-graph modelling," *IEEE Constrol Systems Magazine*, vol. 27, no. 2, pp. 24–45, 2007.

[32] A. Ghosh, R. Vemuri, and D. R. Vemuri, "Formal verification of synthesized analog designs," in *Proc. IEEE International Conference on Computer Design*. IEEE Computer Society Press, 1999, pp. 40–45.

[33] T. Gowers, J. B. Green, and I. Leader, *Princeton Companion To Mathematics*. Princeton University Press, 2008.

[34] S. Graf and H. Saidi, "Construction of abstract state graphs with PVS," in *Computer Aided Verification*, vol. LNCS 1254. Springer, 1997, pp. 72–83.

[35] M. Greenstreet and I. Mitchell, "Reachability analysis using polygonal projections," in *Hybrid System: Computation and Control*, vol. LNCS 1569. Springer, 1999, pp. 103–116.

[36] S. Gupta, B. H. Krogh, and R. A. Rutenbar, "Towards formal verification of analog designs," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, 2004, pp. 210–217.

[37] K. Hanna, "Reasoning about analog-level implementations of digital systems," *Formal Methods in System Design*, vol. 16, no. 2, pp. 127–158, 2000.

[38] W. Hartong, K. Klausen, and L. Hedrich, "Formal verification for nonlinear analog systems: Approaches to model and equivalence checking," in *Advanced Formal Verification*. Kluwer, 2004, pp. 205–245.

[39] M. Jirstrand, J. Gunnarsson, and P. Fritzson, "A new modeling and simulation environment for mathematica," in *International Mathematica Symposium*, 1999.

[40] D. Karnopp and R. C. Rosenberg, *Analysis and Simulation of Multiport Systems: The Bond Graph Approach to Physical System Dynamics*. The MIT Press, 1968.

[41] M. Kennedy, "Chaos in the colpitts oscillator," *IEEE Transactions on Circuits and Systems*, no. 41, pp. 771–774, 1994.

[42] T. Kropf, *Introduction to Formal Hardware Verification*. Springer, 1999.

[43] R. P. Kurshan and K. McMillan, "Analysis of digital circuits through symbolic reduction," in *IEEE Transactions on Computer-Aided Design*, vol. 10, 1991, pp. 1350–1371.

[44] S. Little, D. Walter, K. Jones, and C. Myers, "Analog/mixed-signal circuit verification using models generated from simulation traces," in *Automated Technology for Verification and Analysis*, vol. LNCS 4762. Springer, 2007, pp. 114–128.

[45] S. Little and C. Myers, "Abstract modeling and simulation aided verification of analog/mixed-signal circuits," in *Proc. Workshop on the Formal Verification of Analog Circuits*, 2008.

[46] S. Little, D. Walter, N. Seegmiller, C. Myers, and T. Yoneda, "Verification of analog and mixed-signal circuits using time hybrid petri nets," in *Automated Technology for Verification and Analysis*, Springer, Ed., 2004, pp. 426–440.

[47] T. Maehne and A. Vachoux, "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.

[48] S. McLaughlin and J. Harrison, "A proof-producing decision procedure for real arithmetic," in *Automated Deduction*, vol. LNCS 859. Springer, 2005, pp. 295–314.

[49] T. Melham and M. Gordon, *Higher Order Logic and Hardware Verification*. Cambridge University Press, 1993.

[50] H. M. Paynter, *Analysis and Design of Engineering Systems*. The MIT Press, 1961.

[51] A. S. Perelson, "Description of electrical networks using bond graphs," *Circuit Theory and Applications*, vol. 4, pp. 107–123, 1976.

[52] B. Poizat, D. Brandar, and M. Klein, *Course of Models Theory*. Springer, 2006.

[53] S. Ratschan and Z. She, "Safety verification of hybrid systems by constriaint propagation based abstraction refinement." *ACM Transactions in Embedded Computing Systems*, vol. 6, no. 1, 2007.

[54] ——, "HSolver : Verification of hybrid systems based on the constraint solver RSolver." [Online]. Available: `http://hsolver.sourceforge.net/`

[55] G. Roberts and A. S. Sedra, *SPICE*. Oxford University Press, 1996.

[56] A. S. Sedra and K. C. Smith, *Microelectronic Circuits*, 5th ed. Oxford University Press, 2004.

[57] R. Sommer, E. Hennig, M. Thole, T. Halfmann, and T. Wishmann, "Proc. symbolic modeling and analysis of analog integrated circuits," in *European Conference on Circuit Theory and Design*, 1999.

[58] J.-E. Stromberg, S. Nadjm-Tehrani, and J. L. Top, "Switched bond graphs as front-end to formal verification of hybrid systems," in *Proc. of the DIMACS International Workshop on Verification and Conrol of Hybrid Systems*, 1996.

[59] G. Sutcliffe and C. Suttner, "The TPTP Problem Library: CNF Release v1.2.1," *Journal of Automated Reasoning*, vol. 21, no. 2, pp. 177–203, 1998.

[60] ——, "The TPTP problem library for automated theorem proving," `http://www.cs.miami.edu/ tptp/`, 2009.

[61] M. Tiller, *Introduction to Physical Modeling with Modelica*. Kluwer Boston, 2001.

[62] A. Tiwari, "Series of abstractions for hybrid automata," in *Hybrid Systems: Computation and Control*, vol. LNCS 2289. Springer, 2002, pp. 465–478.

[63] ——, "HybridSal: A tool for abstracting hybridsal specifications to SAL specifications," `http://sal.csl.sri.com/hybridsal/`.

[64] S. Tiwary, A. Gupta, J. Phillips, and et al., "fSpice: A boolean satisfiability based approach for fromally verifying analog circuits," in *Proc. Workshop on Fromal Verification of Analog Circuits*, 2008.

[65] V. D. Toro, *Engineering Circuits*. Prentice-Hall, 1987.

[66] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design*. Kluver, 1993.

[67] M. Zaki, G. A. Sammane, S. Tahar, and G. Bois, "Combining symbolic simulation and interval arithmetic for the verification of AMS designs," in *Proc. IEEE International Conference on Formal Methods in Computer-Aided Design*, 2007, pp. 207–215.

[68] M. Zaki, S. Tahar, and G. Bois, "Formal verification of analog and mixed signal designs: A survey," *Microelectronics Journal*, vol. 39, no. 12, pp. 1–10, 2008.