Techniques for the Formal Verification of Analog and Mixed- Signal Designs

Mohamed Hamed Zaki Hussein

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

2008

Your file Votre référence
ISBN: 978-0-494-45663-7
Our file Notre référence
ISBN: 978-0-494-45663-7

ABSTRACT

Techniques for the Formal Verification of Analog and Mixed- Signal Designs

Mohamed Hamed Zaki Hussein, Ph. D.
Concordia University, 2008

Embedded systems are becoming a core technology in a growing range of electronic devices. Cornerstones of embedded systems are analog and mixed signal (AMS) designs, which are integrated circuits required at the interfaces with the real world environment. The verification of AMS designs is concerned with the assurance of correct functionality, in addition to checking whether an AMS design is robust with respect to different types of inaccuracies like parameter tolerances, nonlinearities, etc. The verification framework described in this thesis is composed of two proposed methodologies each concerned with a class of AMS designs, i.e., continuous-time AMS designs and discrete-time AMS designs. The common idea behind both methodologies is built on top of Bounded Model Checking (BMC) algorithms. In BMC, we search for a counter-example for a property verified against the design model for bounded number of verification steps. If a concrete counter-example is found, then the verification is complete and reports a failure, otherwise, we need to increment the number of steps until property validation is achieved. In general, the verification is not complete because of limitations in time and memory needed for the verification. To alleviate this problem, we observed that under certain conditions and for some classes of specification properties, the verification can be complete if we complement the BMC with other methods such as abstraction and constraint based

verification methods. To test and validate the proposed approaches, we developed a prototype implementation in Mathematica and we targeted analog and mixed signal systems, like oscillator circuits, switched capacitor based designs, Delta-Sigma modulators for our initial tests of this approach.

To my parents and my sister

ACKNOWLEDGEMENTS

I have been very fortunate to have Dr. Sofiène Tahar and Dr. Guy Bois as my supervisors. I would like to express my deep and sincere gratitude to both of them. With the enthusiasm, inspiration, sound advice and guidance they provided throughout my Ph.D's studies, I was able to finally write this thesis. I would also like to thank them for supporting me financially which facilitated me to actively concentrate on research.

Dr. Tahar gave me the freedom to pursue this research. His continuous support and great effort were a corner stone in my research, and his great personality has shaped my research interest.

I would like to thank Dr. Bois for his patience with me delivering the research contribution he was expected and for providing the necessary feedback during the thesis.

It has been a great opportunity for me to work with Dr. Ghiath Al Sammane. I am greatly grateful to him also for the inspiring ideas and the long discussions. Without his help, I could not have completed this work.

I also wish to express my gratitude to my Ph.D committee members, Dr. Peyman Gohari and Dr. Ibrahim Hassan for their invaluable feedback throughout the Ph.D and for giving their limited time for reviewing my thesis. I am specially grateful to Dr. Glenn Cowan for accepting to be on my examination committee. I also like to thank Dr. Mark Greenstreet for taking time out of his busy schedule to serve as my external examiner. I really appreciate having an expert of high caliber like him in my committee

My colleagues from the Hardware Verification Group (HVG), at Concordia University supported me in my research work. I want to thank them for providing a stimulating and fun environment.

I would like to reserve my deepest thanks to my parents and my sister for their perpetual love and encouragement. Their life time support and encouragement have provided the basic foundation of any success I will ever achieve.

Everything I have is given by God, and my gratitude would always be due to Him.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

∀CTL	Universal CTL
AC	Alternating Current
A/D	Analog-to-Digital Converter
AMS	Analog and Mixed Signal
ASL	Analog Specification Language
ASTG	Abstract State Transition Graph
BDD	Binary Decision Diagram
BJT	Bipolar Junction Transistor
BMC	Bounded Model Checking
CAD	Computer Aided Design
CEGAR	Counter-Example Guided Abstraction Refinement
CMOS	Complementary MOSFET
CT-AMS	Continuous-Time Analog and Mixed Signal
CTL	Computational Tree logic
D/A	Digital-to-Analog Converter
DAE	Differential Algebraic Equation
DC	Direct Current
DE	Difference Equations
DT-AMS	Discrete-Time Analog and Mixed Signal
FSM	Finite State Machine
HDL	Hardware Description Language
IA	Interval Arithmetics
IP	Intellectual Property
IVP	Initial Values Problem
LHPN	Labeled Hybrid Petri Nets
LTL	Linear Temporal logic

MILP	Mixed-Integer Linear Programming
MTL	Metric Timed Linear Temporal Logic
MITL	Metric Interval Temporal Logic
MOS	Metal Oxide Semiconductor
MOSFET	MOS Field-Effect Transistor
MVT	Mean Value Theorem
nMOS	n-channel MOSFET
OBDD	Ordered Binary Decision Diagram
ODE	Ordinary Differential Equations
PLL	Phase-Locked Loop
PSL	Property Specification Language
PVS	Prototype Verification System
RF	Radio Frequency
SAT	Boolean Satisfiability Problem
SoC	System on Chip
SMT	Satisfiability Modulo Theories
SMV	Symbolic Model Verifier
SRE	System of Recurrence Equations
STL	Signal Temporal Logic
TCTL	Timed CTL
TEDHS	Threshold-Event-Driven Hybrid Systems
THPN	Timed Hybrid Petri Nets
TM	Taylor Models
TTL	Transistortransistor logic
VHDL	VHSIC HDL
VHSIC	Very-High-Speed Integrated Circuits

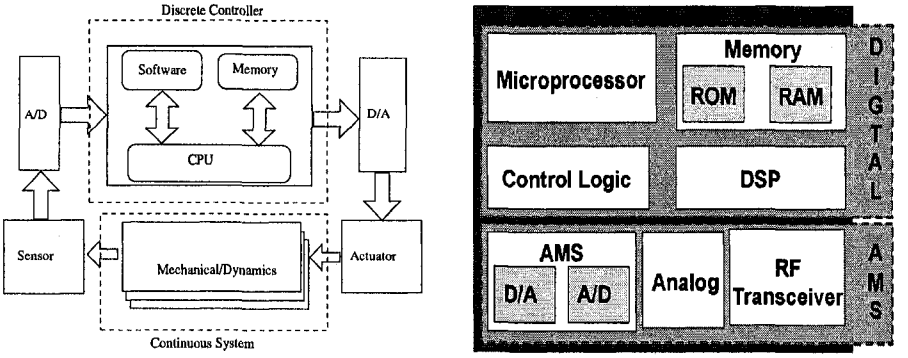
Chapter 1

Introduction

1.1 Motivation

Embedded systems are becoming a core technology in a growing range of electronic devices. Generally, embedded systems are characterized by their reactive and real-time dynamical behavior in response to their environment. Such interaction is often facilitated through sensors to capture the state of the environment and actuators to change or update the environment (Figure 1.1(a)). Cornerstones of embedded systems are the analog and mixed signal (AMS) System on Chip (SoC) building blocks [67]. Typically, SoC refers to the integration of different electronic intellectual property (IP) and custom design blocks into a single integra-ted chip as depicted in Figure 1.1(b). Among the important functions of AMS designs are the processing of analog signal on the front and back ends of the system. Other functionalities include converting between analog and digital data representation, frequency synthesis and generating timing references. In addition, analog circuits are used for biasing which is necessary for correct and stable operations of the system. In summary [42], AMS designs are needed for:

- **Analog front-end circuits:** On the front-end of the embedded system, signals from sensors or antenna (in Radio Frequency (RF) designs) must be sensed, received, amplified and filtered up to the level that allow digitization with sufficient signal to



(a) Architecture Model (b) System-on-Chip

Figure 1.1: Embedded System

noise ratio. In addition, in case of RF, a down-conversion mixer performs frequency translation by multiplying the RF signal with local oscillator generated signal.

- **Analog back-end circuits:** At the back-end of the system, signals are re-converted from digital to analog. Among the analog circuits at the back-end are filters, oscillators and buffers. For RF, the analog signal is upconverted to the desired RF band for transmission.

- **Mixed circuits:** Data processing components like analog to digital (A/D) and digital to analog (D/A) converters encode and/or transform the data between analog and digital representations. These include sample and hold circuits, which are usually used to take snapshots (samples) of the analog signal; in phase locked loops (PLL); and frequency synthesizers for generating timing references.

- **Biasing and reference circuits:** These circuits produce stable absolute current and voltage references insensitive to temperature, power supply and load variations that are necessary for correct operations and meeting the challenge arising from reduced supply voltages.

- **High Performance digital circuits:** The largest analog circuits today are high performance (high-speed, low-power) digital circuits. Typical examples are state-of-the-art microprocessors, which make extensive use of full custom design including custom sized transistors as in analog circuits, to push speed or power limits. Also, a critical part in the development of such electronic systems is high-speed inter-chip signalling. Many of the timing problems related to high-speed signalling are mitigated through the use of phase-interpolating circuits to generate precise clock phases.

- **Optoelectronics and electromechanical devices:** Optoelectronics include integrated optical circuit, photodetectors, photodiodes and phototransistors, photoresistors and photoconductor. Electromechanical devices are those that combine electrical and mechanical parts.

1.1.1 AMS Computer-Aided Design

Computer-aided design (CAD) tools have been proposed and developed to overcome challenges in the development process of AMS design circuits. For instance, the full-custom design of analog integrated circuits is very time-consuming and needs experienced designers. In addition the necessity to design and improve the quality of more complex integrated systems with the tight constraints of increasingly shorter time-to-market and productivity increase, led to the awareness of the importance of computer-aided and automated design tools for AMS designs. Such CAD tools and concepts are then needed to provide unique insights into the behavior and characteristics of the integrated circuits, to help the designer select the best design strategies. Finally, CAD tools should tackle the crucial aspects of real designs to correctly and efficiently model these circuits as well as analyzing the corresponding behaviors. In recent years, some breakthroughs have been made in different aspects of the CAD procedure, especially in the development of hardware description languages (HDL) suitable to describe the different AMS behaviors [91];

e.g., VHDL-AMS [110], Verilog-AMS [109] and SystemC-AMS [108]. Other advances have been made in the design procedure, namely analog synthesis and topology selections (in top-down methodologies), design related optimizations like design centering and device sizing and analog layout automation [96]. One important constituent of the CAD framework is the verification task which subsumes several challenging aspects that require extensive expertise and deep understanding of the AMS behavior.

Classification of AMS Designs

Unlike digital designs, the functionality of analog circuits is defined directly in terms of continuous electrical quantities and is usually sensitive to environment factors like signal noise, current leakage, temperature, etc., in addition to higher order physical effects when designing in deep submicron, such as increased parasitics and current leakage which pose a challenge in the design process.

AMS designs are usually classified based on a variety of criteria and/or the type of analysis applied on the designs [17]. For instance, we can differentiate between AMS designs based on the type of signals processed within the design components. A signal can be described as continuous-time when it can assume any analog value over a continuous-time range, whereas a discrete-time signal is an analog signal defined only for discrete values of time. In general a discrete signal can be obtained by taking samples of a continuous-time analog signal at discrete instants of time.

Therefore, for each class of AMS designs, i.e., continuous-time AMS (CT-AMS and discrete-time AMS (DT-AMS), we provide mathematical models capturing the relevant behavior at the different levels of design abstraction. For example, differential equations capture the physical characteristics of the designs, appropriately. On the other hand, certain families of AMS designs (e.g., A/D converters) are composed of digital components that can be adequately modeled at higher levels of abstraction interfaced by threshold event generators components (e.g., comparator circuits). Such systems are typically modeled using piecewise based equations.

To sum up, a key for a sound verification of the different classes of AMS designs is an adequate model that captures both the analog and digital behaviors while being amenable for algorithmic analysis. We will propose in this thesis a computational model which is general enough to represent the different behavioral aspects of CT-AMS and DT-AMS designs.

AMS Abstraction Levels

In general, the verification challenges arise throughout each of the phases of the design process. For a consistent design flow, a compliance certificate approving the correspondence between different design levels (or different designs at a specific level) is required to ensure correctness of the end product and its conformity to the specification. For instance, in the bottom-up design methodology as illustrated in Figure 1.2, the process starts with the design of the individual blocks, which are verified individually and then combined to form the system. However, such verification can be quite expensive as the entire system is represented at the transistor level. A solution to this problem lies in modeling at a higher level than the implementation level, such that an analysis for the whole design can be performed. This is achieved by the development of symbolic analysis which are simplification methods applied to obtain simplified models (e.g., macromodel, behavioral models) preserving the properties of interest. To ensure correctness of the methodology, some notion of equivalence needs to be verified between the implementation and the generated models. Moreover, we want to ensure that the extracted models when combined preserve specification properties.

A wide range of properties and requirements exist for the different classes of AMS designs. In the following, we highlight some of the design and verification challenges at the different levels of abstraction [42]:

- **Circuit Level:** Analysis at the circuit level can be conducted in the time or frequency domain. It includes DC and operating point analysis, small signal analysis;

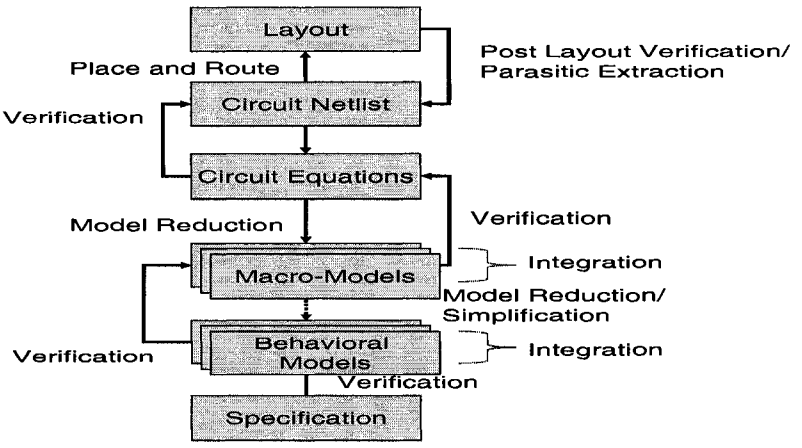


Figure 1.2: AMS Bottom-up Design Methodology

i.e. AC, noise and distortion analysis and transient analysis used to predict the nonlinear behavior of a circuit and periodic steady state analysis.

- **Macromodel Level:** Macromodels are design models with more ideal circuit elements, which approximate the behavior of the original circuit. For example, simplified but convenient approaches for discrete-time circuits such as switched-capacitor oversampling converters use difference equations to model the circuit behavior.

- **Functionality Level:** Many nonlinear blocks of interest like switches, comparators, etc., are intended to switch abruptly between two states. While such operation is obviously natural for purely digital systems, the strongly nonlinear behavior is also exploited in analog blocks such as sampling circuits, switching mixers, analog-to-digital converters, etc.

- **System Level:** Challenges arise not only in the AMS design process, but also during the integration of analog and RF IP designs in SoC platforms. Problems range from correct functionality of the integrated analog and digital parts through conformance to system specification like area and power consumption.

AMS Verification

While AMS components constitute only a small part of the whole SoC (between $5-10$ percent as noted in [10]), the AMS blocks' design and their integration account for $40-50$ percent of the overall design time [16]. Of this design time, $70-80$ percent are spent on verification [16]. Traditionally, simulation is used to verify the designs at abstraction levels from circuit level using Spice based simulators through behavioral level where design are written in programming languages like VHDL-AMS, SystemC-AMS and up to system level. However, simulation is often done manually in an informal fashion and the search of the state space is not complete. As a consequence, simulation methods lack the rigor needed to ensure correctness of the design. Besides, it does not provide the guarantees needed for correct correspondence between the implementation and the approximate models at subsequent design levels, or two models at the same level where robustness and parameter tolerances are considered. In addition, such method falls short to validate interesting properties of the design behavior such as temporal requirements.

Another problem is caused by the fact that while a design defined in advance, one cannot ensure a priori that the desired properties will exactly be met during manufacturing of the actual circuit. Component tolerances will always lead to large variations of a circuits properties, which may result in effects not expected from the results of the numerical simulation. This latter problem cannot be overcome within a single numerical simulation. Therefore more sophisticated methods are usually used as complementary to simulation to raise confidence in the end product[1]. For instance, simulation is complemented by symbolic techniques [96], where the effect of parameter variations on the system behavior is analyzed. Although successful, challenging problems like non-linear effects make these techniques only suitable for simple designs.

The last decade saw the emergence of a new engineering field known as hybrid system theory where researchers have developed formal techniques for the automatic design

[1]Monte Carlo simulation serves as a standard solution for circuits verification in the presence of parameters imprecision. However, it inherits the coverage limitation drawbacks from standard simulation methods.

and analysis of systems with real-time and continuous behavior and which are described by a composition of continuous-time systems and discrete-time systems.

Boosted by the successful application of formal methods in hybrid designs verification, formal methods became a serious candidate for the verification of AMS systems. This growing interest is due to the fact that such methods promise a complete verification, therefore, increasing the level of confidence in the verification results. In particular, one is interested in global properties connected to the dynamic behavior of the AMS systems. For example, we might be interested in properties like "will the circuit oscillate for a given set of parameters, and for all sets of constant input voltages?", "will switching occur in less than a specific amount of time?".

In this thesis, we aim at the development of methods and techniques tackling such challenges in the verification process of AMS designs using methods from hybrid system research.

1.2 AMS Designs as Hybrid Systems

The analysis of the behavior of AMS designs with mixed domains heterogeneity and at different levels of abstraction requires formal tools that cut across existing disciplinary boundaries: the analog part of which is usually modeled as continuous-time or discrete-time dynamical system while the digital part's dynamics are modeled as discrete systems. Moreover, at each level of abstraction, an appropriate model should always be set for the analysis phase. The levels of abstraction for these models include simple algebraic equations, ordinary and partial differential equations, up to block diagram level depending on the level of details needed. In this respect, AMS models have to meet two contradicting demands. On the one hand, they have to describe the physical behavior of a circuit as accurately as possible. On the other hand, the models should be simple enough to keep the computing time for verification reasonably small. For example, complex elements such as transistors can be modeled by small circuits containing basic network elements

described by algebraic and ordinary differential equations only.

1.2.1 Hybrid Systems Modeling

Hybrid systems theory [4] was developed to deal with systems with heterogeneous behavior. Specifically, to fully understand the system's behavior and meet high performance specifications, the designer needs to model all of the dynamics together with their interaction, which is very important when the different parts of the system are tightly integrating or strongly interacting. For instance, at the specification level, the embedded system architecture illustrated in Figure 1.1(a) can be modeled in an abstract way as shown in Figure 1.3. The digital controller is modeled by finite state machines (FSMs), while the dynamical environment is described using systems of ordinary differential equations (ODEs) or difference equations (DE). In addition, the sensor and A/D interface can be modeled as a threshold detector and an event generator respectively, while the actuator and D/A components can be modeled as switches that choose between different system of ODEs and set the initialization and reset conditions necessary for correct functionality.

The unified analysis of such systems results in the development of complex dynamical systems is called hybrid systems. Hybrid systems theory is a general theory dealing with the different aspects of modeling, analysis and verification of systems composed of discrete and continuous components interacting together in a specific manner. Formally, these systems are characterized by the interaction of continuous dynamics models (governed by differential or difference equations), and of logic rules and discrete event systems (described by temporal logic, finite state machines, etc.). Examples of continuous models include analog behavior of electronic components, while examples of discrete dynamics include switching behavior in circuits.

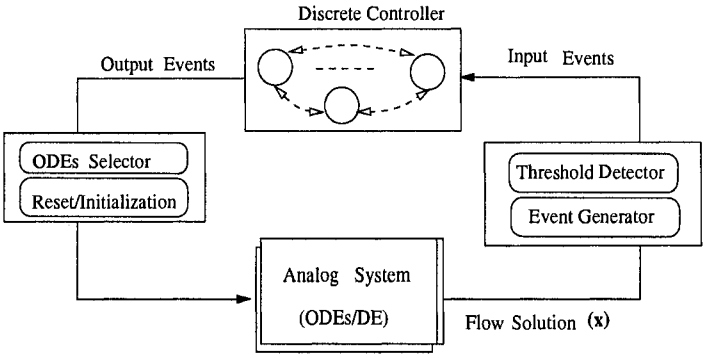


Figure 1.3: Hybrid System Modeling

1.2.2 Hybrid System Approaches

A look at the literature shows that there are many approaches to modeling, analysis and synthesis of hybrid systems. They can be characterized and described along several dimensions. In broad terms, approaches differ with respect to the emphasis on or the complexity of the continuous and discrete dynamics, and on whether they emphasize analysis and synthesis results or analysis only or simulation only. The multi-disciplinary research in hybrid system theory led to different points of view when dealing with issues related to modeling and verification:

- On one end of the spectrum there are approaches to hybrid systems that represent extensions of system theoretic ideas for systems (with continuous-valued variables and continuous time) that are described by ordinary differential equations to include discrete time and variables that exhibit jumps, or extend results to switching systems like piecewise affine and mixed logical dynamical models [95, 12]. Typically these approaches are able to deal with complex continuous dynamics and are amenable to symbolic analysis.

- On the other end of the spectrum there are approaches to hybrid systems that are embedded in computational models and methods, that represent extensions of verification methodologies from discrete systems to hybrid systems. Typically these

approaches are able to deal with complex discrete dynamics described by finite automata and emphasize analysis results (verification) and simulation methodologies. The approach pursued by computer scientists is to extend traditional finite-state automata by introducing progressively more complex continuous dynamics. Several models along these lines are hybrid automata [61] and its variants, e.g., piecewise-constant derivative systems [81, 31].

- There are additional methodologies spanning the rest of the spectrum that combine concepts from continuous control systems described by linear and nonlinear differential/difference equations, and from supervisory control of discrete event systems that are described by finite automata and Petri nets among these models is switching models [15] and threshold-event-driven hybrid systems (TEDHS) [18]. For instance, hybrid Petri Nets proposed by Bail *et al.* [71] is a combination of ordinary and continuous Petri nets. It inherits all the modeling facilities of Petri nets such as the ability to capture concurrency, synchronization and conflicts, allowing the modeling of systems with continuous flows and linear evolutions in an intuitive way. Allam and Alla [2] present a procedure for constructing the hybrid automaton associated with a hybrid Petri net, in order to benefit from the modeling power of the latter and the analysis power of the former.

In summary, the benefits of a unified hybrid system modeling for AMS designs are numerous:

- It provides a unified view of the many behavioral aspects of the AMS designs involving continuous and discrete event dynamics. Consequently, it paves the way to a reasoning mechanism on the global properties of the design.

- By taking into consideration the different dynamics and their interactions at the same time, we can capture the behavior of the system more accurately.

- From the design point of view, through a more complete study of such systems, advanced design and verification methodologies can be developed.

- Since the behavior of AMS systems are very rich and their hybrid nature makes their mathematical models quite complex, research in hybrid systems presents significant challenges; on the other hand, it offers significant promises.

Central to the AMS verification is an adequate model that captures both the analog and digital behavior meanwhile amenable for algorithmic analysis. In this thesis, we provide a modeling framework which is amenable to formal verification.

1.2.3 Hybrid Systems Verification

The goal of formal verification is to prove that a representation of the actual system satisfies the desired and anticipated behavior. More specifically, in formal methods, a decision procedure checks whether a mathematical model for the design satisfies some given properties in the specification; this can be applied using several techniques such as *model checking* [22, 66] or *theorem proving* [66]. Another verification problem is to check the correspondence between two mathematical model representing different levels of the same design; this is known as *equivalence* or *compliance checking* [66]. In addition, hybrid semi-formal techniques combining simulation and formal based methods have been developed as way to benefit of the advantages of these methods, where logical models are used to analyze the simulation results [116].

Model checking [22] is a powerful technique developed initially for the algorithmic verification of digital systems, with the dynamic properties expressed using temporal logics [22]. Model checking has several advantages when compared to other verification approaches. It can automatically provide a complete coverage of the state space, while returning sound verification results. Furthermore, the nature of model checking makes it adequate for the verification of several interesting properties that characterize the behavior of hybrid systems. In the following, we will review the major works done in adopting model checking for hybrid systems.

1.2.4 Model Checking Hybrid Systems

In model checking, the model of the design under verification is a kind of transition system describing all its possible behaviours while the specification property is a temporal logic formula that is interpreted over the model by exhaustive exploration of the state space. This exploration can be either explicit or implicit [22]. In general, extending model checking techniques for the verification of hybrid systems is not a trivial task as explained below:

- **Modeling:** Unlike the discrete models used in conventional model checking, the system under verification is modeled in some computational hybrid system formalism, which incorporates the discrete and continuous behavior.

- **Specification:** Desired properties are expressed as temporal logic formulas. However, it is very important to reason about the real-time behavior as well as continuous states behavior of the system. This requires extending the conventional temporal logic to support such constraints.

- **Analysis:** The main challenge in hybrid system model checking algorithm is to obtain information about the continuous behavior of the system. This is manifested with the solution of system of equations. More precisely, this involves the computation of flow pipes, that is, the collection of continuous-time trajectories emanating from a set of initial continuous states.

Several techniques for model checking of hybrid systems have been proposed. They can be (roughly) classified into three categories; *algebraic*, *on-the-fly* and *abstract model checking*. Literature touching the different aspects of the model checking verification is quite wide and spans through many different research domains. We will highlight in the following the most relevant work while in depth investigations can be found in references therein.

- **Algebraic Methods:** The application of algorithmic verification like model checking is based on the existence of analytic solutions to the differential equations and the representation of the state space in a decidable theory of the real numbers. This direction was initiated with the work of Pappas *et.al* [115, 70] and further extended with the work of Rodrguez-Carbonell *et.al*[94] and Mishra *et.al* [87]. Another direction was described by Henzinger *et. al* [59] where he proposed analyzing nonlinear hybrid systems by first translating the system to a linear hybrid automata counterpart, and then using automated model-checking algorithm on the simplified system.

  While the approach allows a precise and sound verification, it is not attractive in terms of practicality as the linearization method proposed in [59] is restrictive and finding a closed form solution is not possible for most classes of systems of ordinary differential equations (ODEs).

- **On-the-fly Model Checking:** This approach computes a set of reachable states that corresponds to an over-approximation of the solution of the system equations, which is obtained for a bounded period of time. In this approach only a partial state space is explored; hence, this can be referred to as bounded model checking (BMC). The basis of the methods is combining a numerical based integration of the differential equations and numerical representations of approximations of state space typically using (unions of) polyhedra. These techniques provide the algorithmic foundations for the tools that are available for computer-aided verification of hybrid systems [69, 4] like Checkmate [19], d/dt [8], PHaver [35], etc.

  For instance, in [51], Halbwachs *et.al* used convex approximation of linear equations to describe the solution flow. The work is latter implemented in HyTech [61]. HyTech supports several abstract-interpretation operators [25, 60], including the

convex-hull operator and the extrapolation operator [24, 51]. Clarke *et. al* [20], extended the Checkmate verification toolbox with an abstraction refinement methodology [20].

The on-the-fly approach is the most widely investigated model checking technique for hybrid system. Nevertheless, two main issues can be associated with the methods developed. First, the nature of the approach is bounded in time and therefore a complete verification cannot be guaranteed. Nevertheless a property like oscillation behavior can be verified by showing an inclusion fixpoint. The other issue is with the precision of the abstraction. The numerical over-approximation of the reachable states can lead to loose results that are trivial for the verification. Therefore a suitable abstract domain must be carefully chosen. Moreover, such method should always be supported with a refinement procedure to avoid spurious counter-examples.

- **Abstract Model Checking**: The whole state space is subdivided into regions and then heuristic rules define the transitions between states. Conventional model checking algorithms are applied on the new abstract model of the system, which is generally described as a finite state automaton.

  Alur *et. al* [5] used the algorithms for solving flow problems to help generate predicate for the predicate abstraction methodology. However, this work was limited to specific systems such as simple linear systems. In [59], Henzinger *et. al* considered linear hybrid automaton where the continuous environment is partitioned into a finite number of classes such that within each class, the continuous variables are governed by constant polyhedral differential inclusions. Other work in this direction is the work by Stursberg [103, 102] and the work of Ratschan, where they used the concept of predicate abstraction at the core of a constraint solver algorithm for hybrid systems [93].

In [106] a qualitative based approach was developed for abstract model generation for hybrid systems, based on higher derivative analysis. This work was later extended in [107] by using invariance to obtain more precise abstract models. A similar invariant based approach was proposed in [98], where more general invariants are constructed for the whole system. In [92], the authors proposed a similar framework using the idea of barrier certificates. Barrier certificates if they exist, are invariants that separate system behavior from a bad state. Hence, they can verify safety properties.

The *a priori* abstraction of the whole state space allows an unbounded verification of the results, hence contributing to the confidence in the verification results. On the other hand, such abstraction is only suitable for checking a small class of properties (i.e., safety properties) and therefore, it limits the capability of the model checking. Due to the over-approximation inherent in this methods, it should always be supported with a refinement procedure to avoid spurious counter-examples.

We present in this thesis, a novel on-the-fly model checking approach for AMS designs, which provides tight bounds for the reachable states by using non-convex over-approximation. In addition, the symbolic nature of the chosen representation of the reachable states using polynomials terms, has the advantage of minimizing the risk of state explosion. However, as this kind of verification is not complete in general as stated earlier, we complement the verification with an abstract model checking approach, in order to provide a complete verification framework.

1.3 Scope of the Thesis

1.3.1 AMS Formal Verification

Using formal methods, two types of properties are frequently distinguished in temporal logic: *safety properties* state that something bad does not happen, while *liveness properties* prescribe that something good eventually happens. In the context of AMS designs, examples of safety properties can be about voltages at specific nodes not exceeding certain values throughout the operation. Such properties are important when designing AMS circuits, as a voltage exceeding a certain specified value can lead to failure of functionality and ultimately to a breakdown of the circuit which can result in undesirable consequences for the whole design. On the other hand, occurrence of oscillation or switching are good examples of liveness properties. A bounded liveness property specifies that something good must happen within a given time, for example, switching must happen within n units of time, from the previous switching occurrence.

Obviously, the AMS design process must ensure, with a high degree of confidence, the proper functionality in all possible situations and that the design will meet its performance requirements. Therefore, precise constraints and properties identification along with verification from the behavioral level through functional and circuit levels is needed. This motivates the necessity of using formal verification methodologies throughout the design process. An extensive state of the art survey of the different research directions will be provided in the next chapter of the thesis.

The rich and diverse ideas that were developed in the hybrid systems community provided a fertile environment for exploring and adopting the application of formal methods to new domains. One such domain is analog and mixed signal design, which as outlined earlier poses many challenges in terms of analysis and verification. On the other hand, the diversity of the AMS modeling and representation as well as the objective properties needed to be checked make the development of a unified formal verification technique a very difficult task to achieve. Nevertheless, a formal verification framework that

subsumes the different classes of designs and addresses a variety of functional and timing specifications will alleviate the verification problem. Therefore, the research presented in this thesis is concerned with the development of a formal verification framework for AMS designs. However, before we present the proposed methodology, we will review the main research activities in the application of formal methods for the verification of AMS systems. We will emphasize techniques of interest to the work presented in this thesis. A more thorough investigation of related work will be provided in Chapter 2

1.3.2 State of the Art

Model checking and reachability analysis were proposed for validating AMS designs over a range of parameter values and a set of possible input signals. Common to the proposed methods is the necessity for the explicit computation of the reachable sets corresponding to the continuous dynamics behavior. Such computation is usually approximated due to the difficulty of obtaining exact values for the reachable state space (e.g., closed form solutions for ODEs cannot be obtained in general).

Several methods for approximating reachable sets for continuous dynamics have been proposed in the open literature. They rely on the discretization of the continuous state space by using over-approximating representation domains like polyhedra and hypercubes. In [76], the authors construct a finite-state discrete abstraction of analog circuits by providing a partitioning of the continuous state space into fixed size hypercubes. They use numerical techniques to compute the reachability relations between these cubes before applying conventional model checking on the abstract model. In contrast to the work in [76], the authors in [57] used variable sized hypercubes to model the abstract state space, while they used heuristics to identify possible transitions between adjacent regions. The *a priori* abstraction of the state space developed in [76, 57] is usually computationally expensive to apply. Moreover, such exploration techniques are not practical in general as for a given set of initial conditions, only some parts of the state space need to be explored. In this thesis, we evaluate an alternative approach where we partition the state space into

non-linear regions and use qualitative characteristics of the state space in order to define the transitions between the regions. Such qualitative based partitioning is usually more precise and also leads to a smaller abstract model.

On-the fly algorithms have been proposed with the development of the Hytech tool [61] for the verification of hybrid systems with simple dynamics using polyhedral over-approximations. To deal with the complex behavior of the circuits, the authors of [49, 117] proposed combining discretization and projection techniques of the state space, hence reducing its dimension. Variant approaches of the latter analysis were proposed. For instance, the model checking tools d/dt [28], CheckMate [50] and PHaver [37] were adapted and used in the verification of a biquad low-pass filter [28], a tunnel diode oscillator [50], and voltage controlled oscillators [37]. Petri net based models and algorithms have been developed also for the reachability analysis of AMS designs in [74, 73].

The bounded verification for continuous-time designs we present in this thesis is in the same spirit as the above mentioned works in terms of requirement for state exploration. However, we can identify two distinct features of our approach. First, we rely on functional based modeling form as a way to model the hybrid behavior design rather than a computational model like an automaton. Such modeling provides us with a more compact representation amenable to the rich application of symbolic analysis, hence leveraging the verification. Second, we apply the verification over Taylor model forms [13, 77] which provide tight bounds for the reachable states by using non-convex over-approximation. In addition, Taylor models allow the symbolic representation of the reachable states using polynomials terms, therefore minimizing the risk of state explosion and providing a way for scalability. Apart from these features, the fact that polynomial formulas reside at the heart of modeling different classes of AMS designs is an incentive to explore different verification problems within a unified framework.

Few works were concerned with the verification of discrete-time AMS designs. For instance, in [50] a discrete version of the Checkmate tool was used to verify the stability

of a $\Delta\Sigma$ modulator. In [28], the authors proposed to reformulate bounded time reachability analysis as a hybrid constrained based optimization problem that can be solved by techniques such as mixed-integer linear programming [12]. The verification idea is to compute a set of worst case trajectories whose safety implies the safety of all the other trajectories. In [38], the authors proposed a bounded model checking approach for the verification of the static behavior of AMS designs. The idea is based on validity checking of first-order formulae over a finite interval of time. The authors trade-off accuracy with efficiency by basing the analysis on rational numbers rather than real numbers, hence affecting the soundness of the verification. In addition, the method is only limited for designs with linear dynamics.

In contrast to the above discussed work, we apply bounded model checking for discrete-time AMS designs supported with an induction theorem prover engine and a counter-example refinement procedure, allowing in some cases, the complete property verification of the designs as will be demonstrated throughout the thesis. The superiority of the approach is derived from the fact that we overcome the time bounded verification of current methods by extending bounded model checking with a mathematical induction engine that allows unbounded time verification. In the following, we describe the proposed methodology preceded by a brief introduction of the basic concepts of formal verification.

1.3.3 Basic Verification Concepts

A *model checking* algorithm determines whether a mathematical model of a system meets a specification that is given as a temporal-logic formula. More formally, the model checking problem is defined as follows: Given a model M of a design and a property P expressed in temporal logic, check $M \models P$, i.e., check if P holds in the model M.

In reality, it is not always possible to generate a computational model representing all possible executions (behavior) of a design. Hence, properties in questions about the

concrete behavior of the design are most often hard or even impossible to answer. In general, the size of the state graph can be exponential in the description of the system (leading to the state explosion problem), and infinite state systems cannot be handled without further measures. Consequently, a significant amount of research in model checking has been devoted to both problems.

One possible solution is to limit the explored state space. *Bounded model checking* (BMC) was first put in practice in [14]. BMC aims at solving the same problem as traditional model checking, however, it has a unique setting for the verification problem. The user has to provide a bound on the number of cycles (time steps in case of analog models) that should be explored, which implies that the method is incomplete if the bound is not high enough. It then uses constraint satisfiability techniques [14] to verify the properties for the bounded steps.

As another approach, many researchers consider *model abstraction* as one of the most powerful tools to combat the state explosion problem. The main idea of model abstraction is to find a map between the actual set of values of state variables and a small set of abstract values such that a simulation relation (a mathematical relation) exists between the original transition system and the newly created one. The model checking problem thus becomes the following: given a model M and a temporal logic property P, compute an abstraction M^* of the model and an abstraction P^* of the property and check whether $M^* \models P^*$. Of interest in this thesis are two forms of this abstraction concept, i.e., the *abstraction refinement* framework and the *predicate abstraction* technique.

Abstraction refinement is a methodology to try to alleviate the complexity of the verification problem by starting with a coarse abstraction and subsequently refining it based on information from unsuccessful verification attempts [21]. On the other hand, predicate abstraction is a technique to obtain a finite approximation of infinite state system [45]. Given a concrete infinite state system and a set of abstraction predicates, a conservative finite state abstraction is generated. Model checking is then applied on the

generated system. If the property is verified then it holds in the concrete system. Otherwise an abstract counter-example trace is generated and analyzed according to an abstraction refinement framework. An in depth classification of abstraction concepts have been discussed in the overview paper [27].

Additionally, in some cases the verification can be achieved without the need to explore or to abstract the state space. For instance, *invariant checking* [118] is a technique in which a property is verified to always hold true over the structure of the system equations. Another method is *induction verification* [118], which is suitable to prove properties for discrete-time designs. In both approaches, the verification can be done through theorem proving or constraint solving. While incomplete in general (a negative verification answer is not conclusive), these methods are usually adequate as preprocessing steps for more complex verification tasks such as abstract model checking.

1.3.4 Proposed Verification Methodology

The verification framework described in this thesis is composed of two proposed methodologies each concerned with a class of AMS designs, i.e., continuous-time AMS designs and discrete-time AMS designs. The common idea behind both methodologies is built on top of Bounded Model Checking (BMC) algorithms. The BMC is achieved using symbolic simulation and constraint solving.

Briefly, the idea behind constraint solving is to solve problems by stating constraints about the problem area and consequently finding solutions satisfying all the constraints. On the other hand, symbolic simulation is a form of simulation where many possible executions of a system are considered simultaneously. This is typically achieved by abstracting the domain over which the simulation takes place. The symbolic simulation is generally based on algebraic rewriting rules that are applied on the design equations.

In general, the verification is not complete because of limitations in time and memory needed for the verification. To alleviate the problem, we observed that under certain

conditions and for some classes of specification properties, the verification can be complete if we complement the BMC with other methods like abstraction and constraint based verification approaches.

Continuous-time AMS Verification

The proposed verification methodology for continuous-time AMS designs is shown in Figure 1.4. For continuous-time AMS designs, bounded model checking is applied on an over-approximation of the system model based on the concept of Taylor model arithmetics. Taylor model arithmetics were developed by Berz *et. al* [13, 77] as an interval arithmetics extension to Taylor approximations allowing the non-linear approximation of system reachable states using non-convex enclosure sets. In the proposed approach, state space exploration algorithms are handled symbolically with Taylor model arithmetics to verify timed temporal logic properties. Such modeling allows the computation over continuous quantities while avoiding the unsoundness inherent in the conventional numerical Taylor approximation. If there exits a path for which the property evaluates to false, then we provide a counter-example that is subject to a validation procedure to check whether it is spurious or not. If it is not spurious, then the counter-example is a concrete one and the design is proved faulty, otherwise a refinement procedure is used to remove the spurious counter-example and the verification is repeated. If all paths give true, then we say that the design satisfies the property for a bounded time.

In some cases, an unbounded verification of continuous-time can be achieved using the concept of lazy abstraction. We propose a qualitative abstraction approach for Continuous-Time AMS designs represented such that the satisfaction of the property in the abstract model guarantees its satisfaction in the circuit-level model. This is done in two stages. In invariant checking, the state space is initially partitioned based on the qualitative properties of the AMS model and symbolic constrained based methods are applied to check for invariant property validation. In case of failure, an iterative verification/refinement process is applied where the regions violating the property are refined

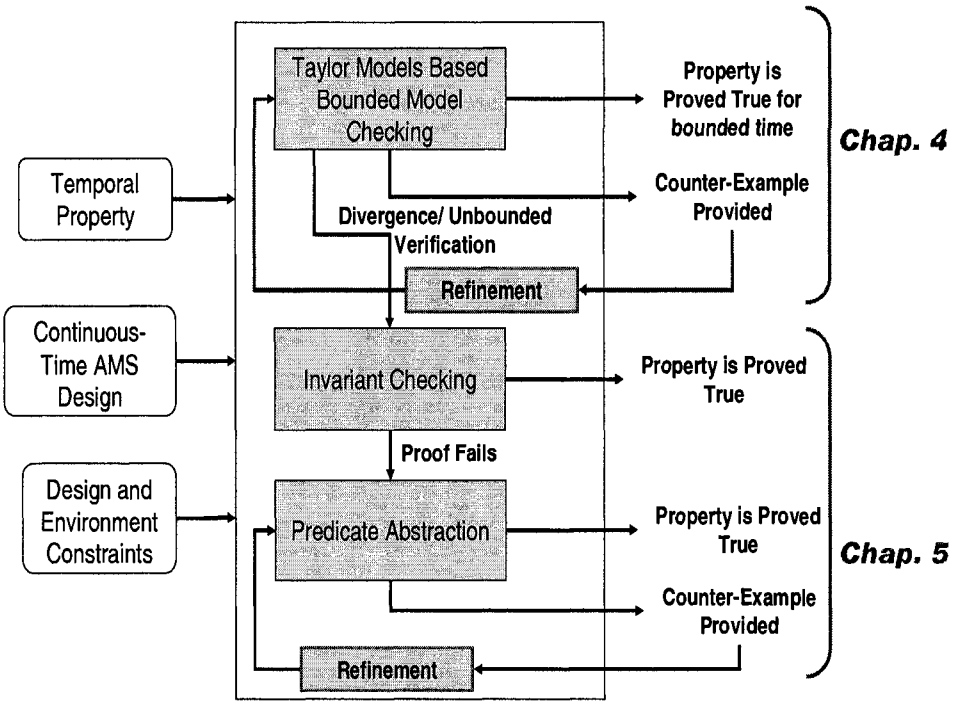


Figure 1.4: Verification Methodology for Continuous-Time AMS Designs

using the concept of predicate abstraction and symbolic model checking is applied for the property validation. The extraction of the predicates is incremental in the sense that more precision can be achieved by adding more information to the original construction of the system. When the property is marked violated, one possible reason is because of the false negative problem due to the over-approximation of the abstraction. In this case, refinement techniques are introduced.

Discrete-time AMS Verification

For the discrete-time AMS designs, the proposed verification algorithm is based on combining induction and bounded model checking to generate a correctness proof for the system as shown in Figure 1.5. Given an AMS described using standard recurrence equations and a set of properties, the bounded model checking is applied using interval analysis [85]

over the normal structure of the recurrence equations. Interval analysis is used to simulate the set of all input conditions with a given length that drives the discrete-time system from given initial states to a given set of final states satisfying the property of interest. If for all time steps, the property is satisfied, then verification is ensured otherwise we provide counter-examples for the non-proved property. Due to the over-approximation associated with interval analysis, divergence can occur leading to false negative. To overcome this drawback, unbounded verification can be achieved using the principle of induction over the structure of the recurrence equations. A positive proof by induction ensures that the property of interest is always satisfied, otherwise a witness can be generated that identifies a counter-example. One drawback of this method is the requirement of predefined constraints to achieve the verification. In order to find a suitable set of constraints, we resort to the d-induction verification method. The method is an algebraic version of the induction based bounded model checking developed recently for the verification of digital designs [6]. We start with an initial set of states encoded as intervals. Then iteratively the possible reachable successors states from the previous states are evaluated using interval analysis based computation rules over the system equations. If there exists a path for which the property evaluates to false, then we search for a concrete counter-example. Otherwise, if all paths give true, then we transform the set of current states to constraints and we try to prove by induction that the property holds for all future states. If a proof is obtained, then the property is verified. Otherwise, if the proof fails then, the BMC step is incremented; we compute the next set of interval states and the operations are re-executed.

1.4 Thesis Contribution

The main contribution of the thesis is the development of a formal verification framework that brings together a set of mathematical and computational tools for reasoning about properties of AMS designs. The contribution can be summarized with the following points:

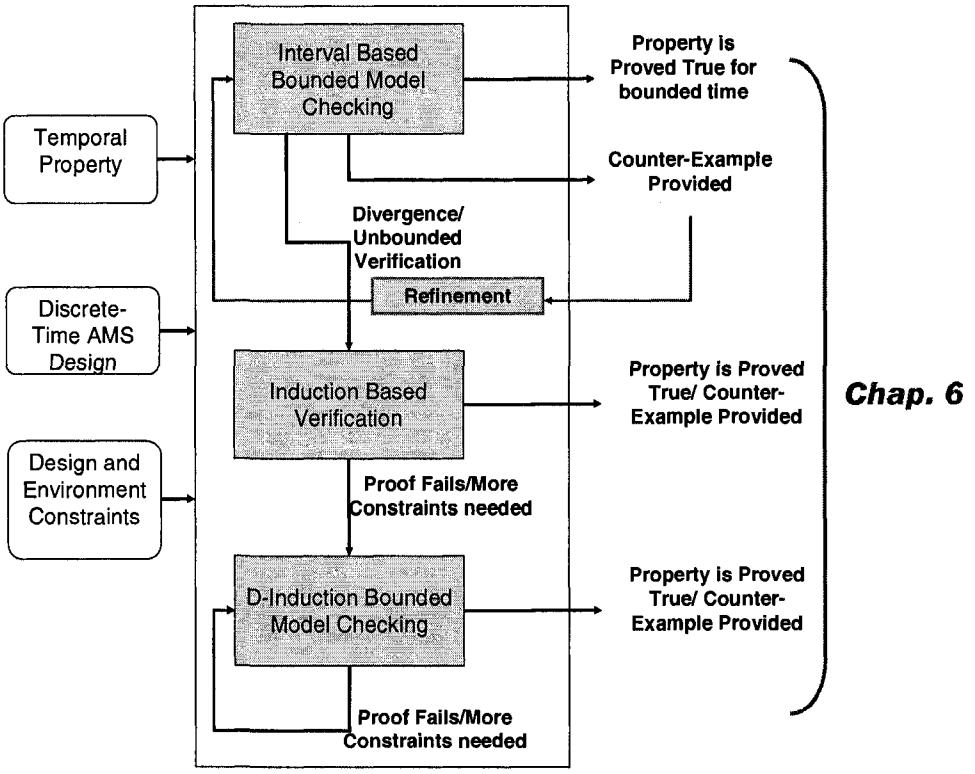


Figure 1.5: Verification Methodology for Discrete-Time AMS Designs

- We provide an extensive survey of the research activities in the AMS formal verification [Bio:Jr-02, Bio:Cf-12].
- We introduce a functional modeling method for AMS designs, which allows the hybrid representation of the digital and continuous part of the designs [Bio:Jr-03, Bio:Jr-05, Bio:Cf-10].
- For CT-AMS systems, we propose a bounded model checking algorithm extended with counter-example analysis and refinement procedure. The algorithm is based on Taylor model arithmetics and symbolic simulation [Bio:Jr-05, Bio:Cf-05].
- We propose a bounded model checking algorithm for DT-AMS. The underlying idea of the BMC is based on combining symbolic simulation, and interval analysis

[Bio:Jr-03, Bio:Cf-06].

- We develop an induction based verification engine for unbounded properties of DT-AMS, which extends the BMC to form the d-induction bounded model checking algorithm [Bio:Jr-03, Bio:Cf-10, Bio:Cf-09].

- We develop a qualitative based predicate abstraction for checking unbounded properties of CT-AMS designs. The idea is based on using constraint solving to check for invariants. Additionally, qualitative predicates are extracted from the system equations to construct an abstract state space in a lazy abstraction fashion [Bio:Jr-01, Bio:Jr-04, Bio:Cf-11, Bio:Cf-08].

- We implemented the proposed algorithms and techniques in the computer algebra system Mathematica [Bio:Jr-01, Bio:Jr-03, Bio:Jr-04, Bio:Jr-05]. The advantage of using Mathematica over other systems is the availability of numerous built-in functions and proof capabilities that allows the implementation of the verification algorithms proposed in the thesis.

- We applied the verification on a variety of AMS designs at several levels of design abstraction. We checked different types of functional and timing properties. Among the examples are oscillator circuits [Bio:Jr-01, Bio:Jr-04, Bio:Jr-05], switched capacitor based designs [Bio:Jr-03] and Sigma-Delta modulators [Bio:Jr-03, Bio:Jr-05].

1.5 Thesis Organization

In this thesis, we propose a formal verification methodology for AMS designs. The dissertation is divided into seven chapters with each chapter beginning with an introductory paragraph and a section in which the subject of the chapter is informally introduced. A chapter is devoted to each central contribution. We conclude each chapter with a summary. In addition, experimental studies are provided whenever is needed to support the

corresponding theoretical development.

A sketch of the content of the next chapters is given in the following:

Chapter 2 provides a literature overview on the relevant work on formal verification of AMS designs, along with a critical review of the various schemes used in the modeling and analyzing. We provide summary tables comparing the different techniques based on several criteria relevant to the thesis. We also highlight the pros and cons of the surveyed approaches [2].

After having surveyed through the prior research in Chapter 3, we recall some basic definitions, fundamental analysis concept and results used throughout the thesis. The remainder of Chapter 3 is devoted to the modeling portion of the verification flow. We introduce the modeling and specification approaches used to represent the behavior and the properties of AMS designs. The modeling framework is built upon a discrete-time representation. We also present for the case of continuous-time AMS, an approximation criteria and establish a formal relation ensuring that the devised model preserves the main behavioral aspects of the AMS design under verification.

In the next two chapters, we address the verification problem for continuous-time designs using two complementary approaches. In Chapter 4, we present the bounded model checking approach developed for continuous-time AMS designs. After providing background material related to the verification, a detailed description of a new symbolic verification algorithm is provided. A counter-example refinement procedure is also introduced to enhance the verification results. We end the chapter with an application section, where we experimented with the verification of basic circuits. Invariant checking and predicate abstraction are described in Chapter 5. In this chapter we explain the method for representing the verification as constraint based problem in a way that allows unbounded verification. After introducing the technical background we describe in detail the verification steps before we provide illustrative results for the proposed approach. We

[2] An expert of the field may pass directly from Chapter 1 to Chapter 3.

also show how such a verification approach can complement the bounded model checking to provide a complete verification framework. This is illustrated with the tunnel diode oscillator circuit.

In Chapter 6, we focus on the verification problem of discrete-time designs. We present a bounded verification algorithm based on interval analysis. To enhance the verification, we extend the verification with an induction engine in order to prove safety properties of the system. We apply the technique on several classes of discrete-time AMS designs.

Chapter 7 summarizes the results of this thesis, where a critical analysis of the contributions of the thesis is presented. The successes and limitations of this approach to verifying AMS circuits are discussed. Finally, we propose perspectives for future work, with several ideas for extending this research.

Chapter 2

Literature Overview

2.1 Introduction

During the last two decades, formal verification has been applied to digital hardware and software systems. Recently, however, formal verification techniques have been adapted and applied to the verification of AMS systems as a way to tackle the limitations of conventional simulation techniques [57]. In addition, hybrid semi-formal techniques combining simulation and formal based methods have been developed as a way to benefit from the advantages of these methods, where logical models are used to analyze the simulation results.

In this chapter, we provide a survey and comparison of the research activities in the field of formal verification of AMS design with the proposed approaches in this thesis. We point out the different strengths and weaknesses of the methods and compare to our proposed model checking approaches. In the remaining of this chapter we overview of equivalence checking methods applied to AMS designs, followed by deductive methods and run-time verification. We devot the last part of the chapter for a survey of the different research directions in model checking and reachability techniques for AMS designs.

2.2 Equivalence Checking

Equivalence checking is a problem where we are given two system models and are asked whether these systems are equivalent with respect to some notion of conformance, or functionally similar with respect to their input-output behavior [66]. Verification can be based on specific properties like transient or steady state response properties, in time domain or frequency domain. Such correspondence relation between designs is classically done through exhaustive testing by proving that two expressions are equivalent, which can be a difficult task for any reasonably large circuit. Instead, symbolic reasoning methods can prove or disprove equivalence using decision procedures over the whole range of inputs described symbolically.

An important requirement in behavior equivalence is the specification of tolerance or bounds on parameters and signals which may be needed. A failure occurs if the comparison finds that the results of both design levels are different or different beyond a certain tolerance. In the rest of this section, we survey the relevant work dealing with the equivalence checking problem. A comparison between these work is outlined in the end of the section.

2.2.1 Relevant Work

In [9], the authors proposed a method for applying equivalence checking between two designs (e.g., specification and implementation) of analog systems described by their linear transfer function. The verification idea is based on the discretization of the transfer functions to the Z-domain using bilinear transformation, thereby, the design can be represented in terms of discrete-time components and encoded into FSM representation like Binary Decision Diagrams (BDDs). The verification problem can be stated as follows: the transient behavior of the implementation mimics that of the specification iff for any initial state of the specification, there exists a state in the implementation such that the FSMs representing the two circuits produce identical output sequences for all input sequences.

The discretization of the behavior raises issues like the error analysis which must account for tolerance between the output sequences for both models must be specified. Another issue is state space explosion when the inherited discretization of the design is encoded. This is largely due to the large word size used to encode real signals. Finally, the methodology is only practical for linearized systems as transfer function generation for non-linear circuits is very difficult in general.

Realizing the coefficient of a transfer function exactly using actual components and devices is not always possible as the tolerance region around nominal characteristic must be taken into account. The ideas in [9] have been extended in [99] in the following way. *Given the transfer function description of both the specification and implementation, verify the conformance of the magnitude and phase response of the implementation against the specification over a desired frequency range.* The equivalence verification problem is modeled in [99] as an optimization problem by ensuring that the implementation response is bounded within an envelope around the specification under the influence of parameter variation.

The conformance in [99] is defined using the notion of different frequency bands product response functions of both design models and which serve as objective functions for the global optimization routine. Such definition allows s-domain verification, hence avoiding loss of precision due to the bilinear transformation used in [9].

Conformance checking with parameters variation was also investigated in [63], where the authors present an equivalence checking for linear analog circuits to prove that an actual circuit fulfills a specification in a given frequency interval for all parameter variations. Linear analog circuits can be described by transfer functions, extracted from the netlist by symbolic analysis methods (in case of implementation), resulting in a parameterized description of the circuit behavior. The main idea of the procedure is to compare by inclusion the value sets of the transfer functions of specification and implementation. To ensure soundness, the authors chose an over-approximation for the implementation transfer function while an under-approximation is chosen for the specification transfer

function.

Comparing [9] with [63], we see that in the first work, the authors trade-off accuracy for practicality. They adapt the developed technology based on BDD equivalence checking for verification of analog systems. This comes at the cost of precision which is affected by the discretization process. In contrast, the authors in the second work insist on soundness by checking that the implementation of the behavior is included in the specification behavior.

While the above-mentioned work are concerned with frequency domain verification, others tend to focus on verification in time domain. For instance, in [62], the authors proposed an equivalence checking approach based on qualitative comparison between two representations of the non-linear analog system. However, direct comparison of vector fields for non-linear systems is usually not possible. Therefore, the authors propose to apply non-linear transformations on the sample state spaces to make the comparison possible. The difference between the evaluations of the sampled equations is then calculated allowing the identification of behavior similarity between the two designs under verification by giving an explicit error measure. Unfortunately, finding the correct transformations is a non trivial task and automation is not possible, leading to the introduction of some heuristics to analyze and approximate qualitative behaviors of the circuits, but affecting the soundness of the methodology. The authors applied their methodology for comparison verification of two CMOS inverters with different parameters as well as the verification of an Opamp against its specification.

Another equivalence checking verification approach was proposed in [97] for verifying VHDL-AMS designs. The idea is based on combining equivalence checking, rewriting systems and simulation into one verification environment. The verification methodology consists of partitioning the specification and implementation codes into digital, analog and data converter components. Digital components are verified using classical equivalence checking, while analog specification and implementation are simplified using rewriting rules and pattern matching. Furthermore, the outputs are fed to comparators

to be verified using simulation. This syntactic method can only be performed on simple designs where rewriting techniques can be easily applied. While the presented methodology is practical, it ignores the coupling between the analog and digital parts.

Such syntactic verification for analog circuits can only be applied when the designs are treated at higher level (architectural or behavioral and functional levels) as at low level, non-linear behavior makes such approaches impractical for verification. Instead of direct simulation, advanced verification techniques mentioned earlier can be used to compare analog model behaviors.

2.2.2 Discussion

In general, the nature of analog circuits, most notably the presence of tolerance margins, makes equivalence verification a difficult problem. However, with careful definition of bounds on the parameters as well as the signals, certain compliance relations can be checked. In addition, in contrast to equivalence checking for digital systems where a canonical representation allows easy comparison of two functions representation, no such form exists for analog systems and all the methods presented are design driven in the sense that a priori knowledge of the qualitative and quantitative properties of the design under verification is a requirement for the methodology application. Table 6.2 draws a brief comparison among the above mentioned projects. The table describes the class of system verified, the models used, the analysis regions and domains, the adopted analysis techniques, the tool used, and the case studies verified.

In summary, equivalence checking as it currently stands is premature and is computationally expensive. The extensive use of over-simplification of the designs cast doubts on the soundness of the proposed approaches. A trade-off between automation and soundness was explored using deductive methods as shown next.

Table 2.1: Equivalence Checking Techniques

	[9]	[99]	[63]	[62]	[97]
Type of Systems	Linear Analog	Linear Analog	Linear Analog	Non-linear Analog	Non linear AMS
Models	Transfer function	Transfer function	Transfer function	ODE - DAE	ODE - DAE FSM
Analysis Regions	Transient response	Transient response	Transient response	Near operating point	N/A
Analysis Domain	Z-domain	S-domain	S domain	Time	Time
Techniques and Analysis	OBDDs comparisons	Global optimization	Interval analysis	Qualitative analysis	Rewriting, SAT simulation
Tools	N/A	Matlab	MAPLE	MAPLE	M-CHECK
Case Studies	Low Pass filter	Filters Opamp	Band pass filter, opamp	CMOS inverter opamp	D/A converter

2.3 Proof Based and Symbolic Methods

Theorem provers are formal systems that were developed to prove design properties using formal deduction based on a set of inference rules [66]. Even though these deductive methods are not constrained by any decidability frontiers, their application requires expertise and significant human intervention which makes their application to complex systems very difficult. A lot of research has been focusing on extending theorem provers with decision procedures for verification assistance and automation, as well as formalizing important theories like the real analysis theory. Some primary efforts on verifying AMS systems using theorem provers started recently. In addition to deductive based methods, induction and symbolic based methods were also proposed to prove properties of some classes of AMS designs.

2.3.1 Relevant Work

In [41], the authors used the PVS theorem prover to formally prove the functional equivalence between behavioral specification of VHDL-AMS designs and approximated linearized models of their synthesized netlist. The verification was applied for DC and small signal analysis. The ideas presented can be considered as a starting point for a methodology to verify analog designs, yet important extensions should be studied more, like

avoiding informal linearization, in addition to tackling more complex verification issues especially related to AC analysis.

Similar but more elaborate research was done in [54]. The author proposed an approach for specifying and reasoning about implementations of digital systems that are described at the analog level of abstraction. The approach relies upon specifying the behaviors of analog components (such as transistors) by conservative approximation techniques based on piecewise-linear predicates on voltages and currents. Theorem proving was initially used to check for the implication relation between the implementation and the specification [52]. In order to automate the verification process, the author proposed afterwards the usage of constraint based techniques instead [53].

2.3.2 Discussion

In Table 2.2, we highlight the main points of the work surveyed. The table describes the class of system verified, the models used, the analysis domains, the adopted analysis techniques, the tools used, and the case studies verified.

Comparing with the equivalence checking methods proposed earlier, theorem proving provides a sound answer to the verification problem. However, verifying complex behavior of the designs is a laborious and challenging task and only primitive properties of the designs can be checked. In order to verify more complex properties, and to make the verification process more efficient, run-time verification approaches were proposed as discussed in the next section.

2.4 Run-Time Verification

Run-time verification (logic based monitoring) methods were developed where no computational model is needed prior to the verification, avoiding state space explosion [116]. By employing logical monitors, an efficient analysis of the results is achieved, avoiding exhaustive inspection, by testing whether a given behavior satisfies a property [104].

Table 2.2: Theorem Proving

	[41]	[52, 53, 54]
Type of systems	Piecewise linear	Piecewise linear
Modelling	set of predicates over real	set of predicates over real
Domain Analysis	Time	Time
Verification Method	Deduction	Deduction and constraint solving
Tool	PVS	N / A
Case Studies	Analog Receiver Transmitter	TTL

Monitors for hybrid systems have been developed in [104], where the authors developed tools for monitoring real-time and hybrid systems. Timed and linear hybrid automata can be used to monitor real-time and hybrid behavior, respectively.

Property monitoring of AMS designs is performed in general using assertions and tests. The monitoring can be described in general as follows: the AMS design under verification is simulated by attaching it to a testbench which provides the inputs necessary to drive while monitoring its output. Assertions have the property that they are always checked, regardless of what tests are running. An assertion is a piece of code that continually observes one or more signals and raises a fault when it detects an error condition. They can be placed in the models or in the circuit where they check that the design is being used correctly. The monitor could be as simple as observing a current or voltage, or could be more complicated, taking several signals, processing them and then compare against the expected results.

The main challenges in this technique is the development of adequate monitors. This process can be performed in two different fashions: namely, *Offline* and *Online* monitoring [79]. *Offline monitoring* starts after the whole sequence is given. *Online monitoring* is interleaved with the process of reading the sequence and is similar to the way the sequence is read by an automaton. The two types of monitoring have their strengthes and weaknesses. Offline monitoring allows the verification of more complex properties

like those described backward in time (e.g., using past operators). However, offline monitoring requires the gathering of simulation or execution data in advance which can cost lots of time and memory resources. In addition, violations are not detected as soon as they happen but only after simulation is finished. On the other hand online monitoring is more practical when simpler properties are needed to be verified and violations are identified as soon as they occur. In the following we survey the main projects concerned with monitoring AMS designs

2.4.1 Relevant Work

In [78], the authors proposed an offline methodology for monitoring the simulation of continuous signals described by differential equations. This work is based on extending the PSL (Property Specification Language) [1] logic to support monitoring analog signals, by defining the syntax and semantics of metric timed linear temporal logic (MTL) [105] and extending it with predicates over reals to define the signal temporal logic (STL) [78]. STL is then synthesized into timed automata [80, 79] which monitor simulation traces to check for property violation in an online fashion. The approach was implemented in [90]. No techniques for test case generation is proposed.

A different effort for using PSL properties to monitor AMS designs was proposed in [Bio:Cf-04], where the authors generated observers from PSL properties to monitor the simulation behavior of discrete-time designs using symbolic methods. While the approach is applicable only to discrete-time circuits, it has the advantage of using the standard PSL language making it attractive to be incorporated in the design flow.

In [29, 30], the authors use an extended temporal logic, AnaCTL (CTL for analog circuit verification), for monitoring the transient behavior of non-linear analog circuits. The transient response of a circuit under all possible input waveforms is represented as an FSM created by means of repeated SPICE simulations, bounding and discretizing the continuous state space of an analog circuit. Exhaustive simulation is again a drawback as the created FSM is not guaranteed to cover the total transient behavior leading to soundness

problem.

An online monitoring technique was proposed in [36], where the authors used linear hybrid automata as template monitors for time domain features of oscillatory behaviors, such as bounds on signal amplitude and jitter. For the automata with an error state, the reachability computation can be stopped as soon as this state is reachable. The monitors are used within the PHAver tool where nonlinear circuit equations are modeled with piecewise affine differential inclusions.

In [Bio:Cf-13], the authors propose an online monitoring methodology for analog systems. They present a run-time verification methodology based on monitoring the behavior (solution flow) of analog circuits validated using interval analysis. Given the system description and its specification described by non-linear differential equations and timed computational temporal logic (T-CTL) formulas, respectively, the authors build a timed automata monitor which can detect bad behavior within a specified time period of the interval arithmetics simulation.

2.4.2 Discussion

Run-time verification, although considered only a partial verification technique, combines desirable properties from simulation and formal verification while avoiding the undesirable ones. No computational model needs to be generated prior to the verification, avoiding state space explosion. By employing logical monitors, an efficient analysis of the results is achieved, avoiding exhaustive inspection by engineers.

Table 2.3 summarizes the main characteristics of the described projects. The table describes the class of systems verified, the models used, the monitors language, the monitoring methods, analysis regions and domains, the adopted analysis techniques, the tools used, and the case studies verified.

Run-time verification is considered an enhancement of simulation methods. It allows the detection of faulty properties that are usually hard to detect by simple observation

Table 2.3: Run-time Verification Techniques

	[78, 79, 80, 90]	[29]	[36]	[Bio:Cf-13]	[Bio:Cf-04]
Type of Systems	Non-linear	Non-linear	Piecewise affine	Non-linear	Non-linear
Models	ODE	ODE	ODE	ODE	SRE
Monitors	STL	Ana CTL	Linear HA	TCTL, Timed Automata	PSL observers
Monitoring Type	Offline/Online	Offline	Online	Online	Offline
Analysis Regions	No restriction	Transient response	No restriction	No restriction	No restriction
Analysis Domain	Time	Time	Time	Time	Time
Techniques	Numerical simulation	Numerical simulation	Numerical approx.	Numerical approx.	Numerical simulation
Tools	AMT & Matlab	Spice simulator	PHAver	AWA	Matlab
Case Studies	Sine wave signals, memory	VCO Opamp	Tunnel diode circuit	Tunnel diode circuit	PLL $\Delta\Sigma$ Mod

of simulation results. Yet, run-time verification suffers from the major problems of simulation which lacks the exhaustive machinery needed to gain confidence in the verification results. We believe that model checking techniques stand at a middle ground between the above mentioned approaches. Model checking offers the rigors needed in verification while allowing the automatic verification of complex properties.

2.5 Model Checking and Reachability Analysis

Model checking was initially developed for discrete finite state systems and has been successful in validating communication protocols and hardware circuits. In recent years [61], model-checking algorithms have also been developed for real-time systems that are described by discrete programs with real-valued clocks as well as for hybrid systems. Model checking and reachability analysis of AMS designs have the potential of validating designs over a range of parameters and for all possible input signals all at once such that none of them drives the system into a bad state. An important issue is the solution of the system of differential equations; that is, the collection of continuous time trajectories starting from a set of initial states where in practice the initial conditions are usually not known exactly but only known to lie within some range. However, the effectiveness of model checking is severely constrained by the state space explosion problem and even

undecidability limitations when systems are described by differential equations [65]. It is not always possible to generate a computational model representing all possible executions (behaviors) of a program as well as all its possible execution environments. In such cases, abstraction techniques are usually required in order to achieve the verification task [68].

2.5.1 Relevant Work

The first effort in applying model checking for electronic designs is the work in [76], where the authors proposed verification of digital designs at the transistor level. Given a circuit, they construct a finite-state discrete abstraction by partitioning the continuous state space representing the characteristics of transistors into fixed size multidimensional cubes. Heuristics methods are then used to predict possible transitions between these cubes. The final constructed model is then encoded into an automata that is verified subsequently against some properties using conventional model checking techniques.

In a series of papers [48, 47, 117], the authors proposed overcoming the expensive computational method in [76], by using discretization and projection techniques of the state space into category of geometric polygons called projectahedra (projected polyhedra) [49]. Such models have the property of reducing the dimension of the state space, while maintaining an over-approximation of the dynamic behavior of the design. While this method results in less precise analysis due to projection, it still allows sound verification. Such approach proved useful for the verification of designs with high dimension state space as reported in [117]. Variant approaches of polyhedral based analysis were adapted in [28, 50].

In [28], the authors used techniques developed for hybrid system verification to verify AMS designs. For systems described using differential equations, they use the tool d/dt [8] to overapproximate the reachability analysis. In [50], the authors use the Checkmate tool for the verification of AMS designs. The tool is based on constructing

abstractions of the continuous dynamics, using flow pipes approximations, which are sequences of polyhedra that follow the natural contour of the vector field. Therefore, the state space is partitioned along the waveforms that the system can generate for the given set of initial conditions and there is no need to discretize the entire state space. Checkmate specifications to be verified can be provided as ACTL formulas. For the verification of systems like Δ-Σ modulator, which is described by discrete time components, a modification of the tool to support discrete time analysis was proposed [50].

The work in [50] has been extended further in [37] for the PHAver tool. In this work, the authors proposed a refinement process for the state space, which is carried out using iterations between forward and backward reachability. Such technique as claimed in [37] allows generating more precise bounds for the reachable states.

In [74], the authors proposed modeling analog designs using timed hybrid petri nets (THPN), which is an extension of petri nets for real-time and hybrid systems. They proposed two methods for the generation of the THPNs verification model. In the first method, they translate the circuits differential equation into THPNs. This is done by first discretizing the state space as in [55, 56] and then encoding the state space into THPNs. Additionally, they developed an algorithm in [75], to generate THPNs from simulation data. Over-approximation based analysis is applied on the generated model. In [86], the authors compared verification using their methodology in [74] against simulation results, by examining the effect of variable delays caused by parasitic capacitances and interconnect capacitances on the performance and functionality of the circuits. In [73], they enhanced their methodology in [74] by using a variant of petri nets named labeled hybrid petri nets (LHPNs), that offer a more efficient representation. BDD based symbolic algorithms and satisfiability modulo theories (SMT) [82] techniques are then applied in [112, 113] to check for properties of the design.

The bounded verification for continuous-time designs we present in this thesis is in the same spirit as the above mentioned works in terms of requirement for state exploration. However, we identify two distinct points. First, we rely on a functional based

modeling form as a way to model the hybrid behavior design rather than a computational model like an automata. Such modeling provides us with more compact representation amenable to the rich application of symbolic analysis, hence leveraging the verification. Second, we apply the verification over Taylor model forms which provide tight bounds for the reachable states by using non-convex over approximation. In addition, Taylor models allow the symbolic representation of the reachable states using polynomials terms, therefore minimizing the risk of state explosion.

In contrast to the on-the-fly techniques mentioned above, a priori state space division have been explored as a way to obtain abstractions of the analog behavior of the systems. In [55, 56], the authors proposed to use an automatic state space subdivision method, by discretizing the whole continuous state space into variable sized regions where each of these regions represents a homogeneous part of the state space and is treated as a discrete state of the simplified system. Some kind of estimation techniques are then proposed to describe possible transitions between partitions under the condition of retaining the essential nonlinear behavior of the analog system. Different criteria take care of the resulting error during discretization and try to automatically minimize the error by choosing a suitable subdivision of the state space. The discretized state space is then encoded and CTL based model checking is applied. The proposed approach was implemented in a tool called *Amcheck* [57].

In [44], the authors proposed extending their previous work for the verification of time constraints of analog signals like rise and fall time. The presented extensions are based on developing the analog specification language ASL [100] tailored to represent properties of interest in analog circuit design, such as offset, gain, rise time, and slew rate.

The *a priori* abstraction of the state space developed in [76, 57] is computationally expensive to apply. Moreover, such exploration technique is not practical in general as for a given set of initial condition, only some parts of the state space needs to be explored. In this thesis, we try an alternative approach where we propose to partition the state space

into non-linear regions and use qualitative characteristics of the state space in order to define the transition between the regions. Such qualitative based partitioning is usually more precise and also leads to smaller abstract models.

In order to tackle the state explosion problem for the class of discrete time AMS designs, they proposed to use techniques from optimal control (i.e., hybrid constrained optimization) in order to find bounds of the reachability. The idea is to reformulate bounded time reachability analysis as a hybrid constrained based optimization problem that can be solved by techniques such as mixed-integer linear programming (MILP)[12]. The basic idea is to compute a set of worst case trajectories which implies the safety of all other trajectories.

In [38], the authors developed a bounded model checking tool (*Property-Checker*) for the verification of the quasi-static behavior of AMS designs. The basic idea is based on validity checking of first order formulas over a finite interval of time steps using SMT. In contrast to other approaches, the work presented in [38] trades-off accuracy with efficiency by basing the analysis on rational numbers rather than real numbers.

The approach used in [38], while it avoids the overapproximation issue, is limited to simplified models of AMS design. In fact, the approach does not support systems described using differential equations, however, it is more suitable for systems described using difference equations.

2.5.2 Discussion

Tables 2.4(a) and 2.4(b) give a comparison between the work presented in this section. They describe the class of system verified, the models used, the analysis regions and domains, the adopted analysis and state space partitioning techniques, the tools used, and the case studies verified.

Unlike the presented works, in this thesis we provide a methodology that combines several model checking techniques in an effort to enhance the verification results. We provide a novel on-the-fly model checking approach for AMS designs, which provides tight

Table 2.4: Model Checking Techniques

(a) Comparisons Table

Project	[76]	[49, 117]	[50]	[28]
Type of Systems	Non-linear	Non-linear	Non-Linear	Non-linear
Models	ODE	ODE	HA/ ODE - DAE	HA/ODE -DAE
Analysis Regions	No restriction	No restriction	No restriction	No restriction
Analysis Domain	Time	Time	Time	Time
Techniques and Analysis	Simulation lang. containment	Projection numerical appro.	Numerical approx.	Numerical approx., MILP
State Space partitions	Fixed size hyperCubes	Projectaherda	Convex polyhedra	Orthogonal polyhedra
Temporal Logic	N/A	-	ACTL	-
Verification Method	Abstract model checking	On-the-fly reachability	On-the-fly model checking	On-the-fly reachability
Tools	COSPAN	Matlab/ Coho	Checkmate	d/dt
Case Studies	Interlock circuits	Van der Pool oscillator, toggle circuit	Tunnel diode $\Delta - \Sigma$ mod	Low pass filter $\Delta - \Sigma$ mod.

(b) Comparisons Table (Cont')

Project	[57, 100]	[74, 112, 113]	[38]	[37]
Type of Systems	Non-linear	Non-linear	AMS	Non-linear
Models	ODE, DAE	THPN/ODE	piecewise linear automaton	Piecewise ODE
Analysis Regions	No restriction	No restriction	Steady state	Steady state
Analysis Domain	Time	Time	Time	Time
Techniques and Analysis	Numerical analysis	Numerical approx.	Bounded MC	Numerical approx.
State Space partitions	HyperCubes	Convex polygons	-	Convex polygons
Temporal Logic	ASL/CTL-AT	ACTL	FOL	-
Verification Method	Abstract model checking	On-the-fly/Symbolic model checking	Symbolic model checking	On-the-fly reachability
Tools	Amcheck	ATACS/LEMA	SVC, Property checker	PHAver
Case Studies	Schmidt trigger, Opamp , VCO	Tunnel diode PLL	Sequential AMS circuit	Tunnel diode VCO

bounds for the reachable states by using non-convex over-approximation. In addition, the symbolic nature of the chosen representation of the reachable states using polynomials terms, have the advantage of minimizing the risk of state explosion. However, as this kind of verification is not complete in general as stated earlier, we complement the verification with abstract model checking approach, in order to provide a complete verification framework.

2.6 Summary

In this chapter, we provided a summary of the research activities in the application of formal methods for the verification of AMS systems. We tried to be as exhaustive as possible in collecting the different related work as well as giving comparisons among the research proposed.

As the field of research did not reach the maturity phase yet, standard aspects for comparisons of the various projects are not well defined and there is a lack of a coherent framework and criteria that allows a theoretical analysis and comparison of the methods. We made some efforts in this direction by categorizing and comparing the available state-of-art projects in several aspects which we believe are important to identify the qualitative strengths and weaknesses of each project.

One drawback of our comparison is the lack of testing of the several approaches. This is due to different reasons. First the public unavailability of the prototypes developed in the various projects. Second the lack of benchmarks required for comparison. We hope that in the future, these two obstacles could be overcome so that more insights can be gained about the available methodologies for AMS formal verification.

In the next chapter, we will provide the necessary theoretical concepts required for the development of the verification methodologies proposed in this thesis. We will also tackle one of the main challenges of the verification, which is the development of an adequate model that preserves the required behavior. In this respect, we will provide a

modeling framework for the different classes of AMS designs.

Chapter 3

Preliminaries

During the AMS analysis and verification phase, we usually provide mathematical models that capture the relevant behavior of the designs at different levels of abstraction. For instance, continuous-time models can express a designs' behavior in great details and can thus be seen as residing at the lower end of the abstraction scale. Such models are generally based on differential equations that capture the corresponding functional behavior of the given design as well as its physical characteristics.

Typically, an AMS design can be seen as a composition of two main components, i.e., a continuous-time or a discrete-time analog component and a discrete event controller (digital component) connected through signal interfaces. The analog component is usually composed of circuits built from basic passive and active components (resistors, capacitance, inductance, transistors, etc), connected to various current and voltage sources in a certain topology, achieving a specific desirable behavior (e.g., filtering, amplification, etc.). The digital component is generally modeled at higher level of abstraction (i.e., register level or behavioral model). An interface converting between the components signals (analog and digital signals) can be of the form of a threshold event generator based on comparator circuits. An interface can be also a set of electronic switches that choose between different dynamics based on applied signals at their input. We can therefore view AMS designs as a class of hybrid systems described generally using piecewise modeling,

with piecewise constraints (threshold detection and/or switching conditions) to determine the choice of the appropriate analog dynamics. In case of continuous-time AMS designs, the dynamics of the analog circuits are usually described using differential algebraic equations (DAEs) or system of ordinary differential equations (ODE), while for discrete-time AMS designs, the dynamics of the analog circuits are usually described using system of difference (recurrence) equations (DE).

In this chapter, we provide a unified modeling framework for both continuous-time and discrete-time AMS designs. Such modeling can be seen as a generalization of piecewise modeling which is suitable for symbolic analysis and formal verification. However, due to the difficulty of obtaining a closed form solution for the system of ODEs of continues-time AMS [111], for practical analysis, we also provide necessary condition for obtaining precise approximation of the design models, hence, ensuring the soundness of the verification.

The first part of this chapter reviews some basic definitions and concepts that will be used through the thesis. We will define the concept of generalized If-formula, overview the basics of symbolic simulation and interval arithmetics and Taylor approximation theory. Next, we provide a modeling scheme for AMS designs based on generalized If-formulas, followed by an abstraction approach preserving the behavior of the continues-time designs. After that, we introduce the specification languages necessary for representing the properties of interest. Following these introductory materials, we show how symbolic simulation can be used to obtain a simplified form of the design equations.

3.1 Basic Concepts

3.1.1 Generalized If-Formula

Conditional constructs like $(if - then - else)$ statements are features of many programming languages which perform selected actions depending on whether a specified condition evaluates to $true$ or $false$. In the context of functional programming, these constructs

are referred to as conditional expressions (*if expressions*) as the outcome of the selection is usually evaluated expressions [3]. Moreover, a conditional expression can be seen as an algorithmic generalization of piecewise modeling, where nested expressions can be allowed.

In the context of hardware modeling and verification, the concept of *generalized* $If-formula$ expression was defined by Moore [84] and subsequently used by Al-Sammane in order to model VHDL designs [3]. In this thesis, *generalized* $If-formula$ expressions extend piecewise expressions to describe hybrid behavior of AMS designs. A *generalized* $If-formula$ is formally defined as follows:

Definition 3.1.1. ***Generalized If-formula.***
Let $\mathbb{K}$ be a numerical domain ($\mathbb{N}, \mathbb{Z}, \mathbb{Q}$, $\mathbb{R}$ or $\mathbb{B}$), a *generalized If-formula* is one of the following:

- A variable $x_i(n) \in \mathbf{x}(n)$, with $i \in \{1,\ldots,d\}$, $n \in \mathbb{N}$ *or* $n \in \mathbb{R}$ and $\mathbf{x}(n) = \{x_1(n),\ldots, x_d(n)\}$.
- A constant $C \in \mathbb{K}$
- Any arithmetic operation $\diamond \in \{+,-,\div,\times\}$ between $x_i(n) \in \mathbb{K}$
- A comparison formula: any expression constructed using a set of $x_i(n) \in \mathbb{K}$ and comparison operator $\alpha \in \{=,\neq,<,\leq,>,\geq\}$.
- A logical formula: any expression constructed using a set of $x_i(n) \in \mathbb{B}$ and logical operators: $not, and, or, xor, nor, \ldots$, etc.
- An expression $IF(X,Y,Z)$, where X is a logical formula or a comparison formula and Y,Z are any generalized `If-formula`. Here, $IF(x,y,z) : \mathbb{B}\times\mathbb{K}\times\mathbb{K} \longrightarrow \mathbb{K}$ satisfies the axioms:
  (1) $IF(True,X,Y) = X$
  (2) $IF(False,X,Y) = Y$

Note: When modeling continuous-time AMS designs, *continuous-time If-formula* denotes *generalized If-formula* where n is interpreted as the continuous time variable and we will refer to the index n by $t \in \mathbb{R}$. Otherwise for a discrete-time description we understand that the index $n \in \mathbb{N}$ refers to the discrete-time variable.

3.1.2 Taylor Approximation

Classical numerical approaches for solving an initial value problem consider a sequence of discrete points $t_0, t_1, \ldots, t_m$ for which the solution is approximated. At each new point t_{i+1}, the solution $\mathbf{x}(t_{i+1})$ is approximated by a value $\vec{x}_{i+1}$ computed from the approximated values at the previous points. Taylor series methods [39] are single-step methods that use the Taylor series expansion of the solution function around a point, to obtain an approximation of its value at the next point. This series is computed up to a given order, requiring the evaluation of higher order derivatives of the function. The basic idea is to use the approximation $\mathbf{x}[t_{k+1}] = f(\mathbf{x}[t_k]) + \mathcal{R}m$ of the ODE $\dot{\mathbf{x}} = f(\mathbf{x})$ as a truncated Taylor series for $\mathbf{x}(t)$, expanded about time instant t_k, with a remainder term $\mathcal{R}m$.

Theorem 3.1.1. ***Taylor Approximation [39].***
Suppose a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ over state vector $\mathbf{x} \in \mathbb{R}^d$ is $m+1$ time partially differentiable on the interval $[a,b]$. Assume $\mathbf{x}_0 \in [a,b]$, such that $a, b \in \mathbb{R}^d$, then for each $\mathbf{x} \in [a,b]$, $\exists \lambda \in \mathbb{R}$, $0 \leq \lambda \leq 1$, such that:

$$f(\mathbf{x}) = \sum_{k=0}^{m} \frac{[(\mathbf{x}-\mathbf{x}_0).\nabla]^k f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0}}{k!} + \frac{[(\mathbf{x}-\mathbf{x}_0).\nabla]^{m+1} f(\mathbf{x})|_{\mathbf{x}=\Lambda}}{(m+1)!}$$

where $\nabla = \mathbf{i}_1 \frac{\partial}{\partial x_i} + \ldots + \mathbf{i}_d \frac{\partial}{\partial x_d}$ and $\Lambda = \mathbf{x}_0 + \lambda(\mathbf{x}-\mathbf{x}_0)$

One way of defining solutions is to specify how to generate a future behavior $x(t)$ of the system from any initial state. This approach is closely related to providing a simulation algorithm, in a specific discrete location, integration of the equation gives the unique

solutions inside this location. In general, to obtain an approximate solution of the ODE system, we consider a sequence of discrete time points $t_0, t_1, \ldots, t_m$ for which the solution is approximated, with $h_i = t_{i+1} - t_i$. If the solution $\mathbf{x}(t)$ of an ODE system $\dot{\mathbf{x}} = f(\mathbf{x})$ is a function which is $p+1$ times continuously differentiable on the open interval $(t_i..t_{i+1})$, then, from the Taylor approximation theorem, we have:

$$\mathbf{x}(t_{i+1}) = \mathbf{x}(t_i) + \sum_{k=1}^{p} (\frac{h^k}{k!}\mathbf{x}^{(k)}(t_i)) + (\frac{h^{p+1}}{(p+1)!}\mathbf{x}^{(p+1)}(\xi))$$

with $h = t_{i+1} - t_i$ and $\xi = [t_i, t_{i+1}]$ and $\forall k \in [1, p+1].\mathbf{x}^{(k)} = f^{(k-1)}(\mathbf{x}(t), t)$, where the vector function f is composed by d elementary functions $f_q(x_1, \ldots, x_d)$, $q \in \{1, \ldots, d\}$, such that:

$$f_q^{(k)}(x_1, \ldots, x_d) = \sum_{m=1}^{d} (\frac{\partial f_q^{(k-1)}(x_1, \ldots, x_d)}{\partial x_m} f_m(x_1, \ldots, x_d))$$

3.1.3 Interval Arithmetics

Interval domains make it possible to extend the notion of real numbers by introducing a sound computation framework [85]. In fact, the computer representation of real numbers suffers from the problem of a precision approximation due to limited digits. However, in interval arithmetics, we deal with domains, represented by their endpoints. Thus, computation is carried over intervals that include the real number with full precision. The basic interval arithmetics is defined as follows:

Let I_1 and I_2 be two real intervals (bounded and closed), the basic arithmetic operations on intervals are defined by:

$$I_1 \Phi I_2 = \{r_1 \Phi r_2 | r_1 \in I_1 \wedge r_2 \in I_2\}$$

with $\Phi \in \{+, -, \times, /\}$ except that I_1/I_2 is not defined if $0 \in I_2$ as shown below [85]:

$$\begin{cases} [a,b]^{\iota} & \triangleq & [a,b] \\ [a,b] +^{\iota} [a',b'] & \triangleq & [a+a', b+b'] \\ [a,b] -^{\iota} [a',b'] & \triangleq & [a-b', b-a'] \\ [a,b] \times^{\iota} [a',b'] & \triangleq & [min(aa', ab', ba', bb'), \\ & & \quad max(aa', ab', ba', bb')] \\ 1 \div^{\iota} [a,b] & \triangleq & [1 \div b, 1 \div a] if 0 \notin [a,b] \\ [a,b] \div^{\iota} [a',b'] & \triangleq & [a,b] \times [1 \div [a',b']] \end{cases}$$

In addition, other elementary functions can be included as basic interval arithmetic operators. For example, *exp* may be defined as $exp([a,b]) = [exp(a), exp(b)]$. The fundamental property of interval analysis that ensures soundness of the analysis is described using the following definition:

Definition 3.1.2. ***Inclusion Function [85].***
Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a continuous function, then $F : \mathbb{I}^d \rightarrow \mathbb{I}$ is an interval extension (inclusion function) of f if

$$\{f(x_1, \ldots, x_d) | x_1 \in X_1, \ldots, x_d \in X_d\} \subseteq F(X_1, \ldots, X_d)$$

where $\mathbb{I}$ is the interval domain and $X_i \in \mathbb{I}$, $i \in \{1, \ldots, d\}$.

In order to deal with the discrete part of the AMS design, as a generalization of the inclusion function, interval analysis provides efficient and safe methods for checking truth values of Boolean propositions over intervals by using the notion of an *inclusion test.*

Definition 3.1.3. ***Inclusion Test.***
Given a constraint $c : \mathbb{R}^d \rightarrow \mathbb{B}$, we define $C_{\mathbb{I}} : \mathbb{I}^d \rightarrow \mathbb{B}_{\mathbb{I}}$ to be an inclusion test of c, with a boolean interval domain defined with three values set; $\mathbb{B}_{\mathbb{I}} = \{0, 1, [0,1]\}$, where 0 stands

for $false$, 1 for $true$ and $[0,1]$ for $indeterminate$, iff:

$$\{c(x_1,\ldots,x_d)|x_1 \in X_1,\ldots,x_d \in X_d\} \subseteq C_{\mathbb{I}}(X_1,\ldots,X_d)$$

where $X_i \in \mathbb{I}$, $i \in \{1,\ldots,d\}$.

Inclusion test can be used during the verification algorithm to prove whether the reachable interval states satisfy a given property, or not. We define the inclusion test as follows: $C_{\mathbb{I}}(X) = 1 \Rightarrow \forall x \in X, c(x) = 1$ and $C_{\mathbb{I}}(X) = 0 \Rightarrow \forall x \in X, c(x) = 0$.

Let $x_{\mathbb{I}} = [a,b]$ and $y_{\mathbb{I}} = [a',b']$ be two real intervals. Boolean intervals will be used to extend predicates over reals to intervals. For instance:

$$\begin{cases} x_{\mathbb{I}} \leq^{\mathbb{I}} y_{\mathbb{I}} = 1 & \Leftrightarrow \quad b \leq a' \\ x_{\mathbb{I}} \in^{\mathbb{I}} y_{\mathbb{I}} = 1 & \Leftrightarrow \quad x_{\mathbb{I}} \in y_{\mathbb{I}} \\ & \Leftrightarrow \quad a \geq a' \text{ and } b \leq b' \end{cases}$$

A set of the main logical rules that define the inclusion test is given as follows:

$$\begin{cases} x_{\mathbb{I}} \cap^{\mathbb{I}} y_{\mathbb{I}} & \triangleq & \{max(a,a'),min(b,b')\} \\ x_{\mathbb{I}} \cup^{\mathbb{I}} y_{\mathbb{I}} & \triangleq & \{min(a,a'),max(b,b')\} \\ x_{\mathbb{I}} \vee^{\mathbb{I}} y_{\mathbb{I}} & \triangleq & \{x \vee y | x \in x_{\mathbb{I}} \text{ or } y \in y_{\mathbb{I}}\} \\ x_{\mathbb{I}} \wedge^{\mathbb{I}} y_{\mathbb{I}} & \triangleq & \{x \wedge y | x \in x_{\mathbb{I}} \text{ and } y \in y_{\mathbb{I}}\} \\ \neg^{\mathbb{I}} x_{\mathbb{I}} & \triangleq & \{\neg x | x \in x_{\mathbb{I}}\} \end{cases}$$

3.1.4 Taylor Models

Taylor model arithmetics were developed by Berz *et. al* [13, 77] as an interval extension to Taylor approximations allowing the non-linear approximation of system reachable states using non-convex enclosure sets. Formally, a Taylor model $T_f := p_n(x) + I$ for a given function f consists of a multivariate polynomial $p_r(\mathbf{x})$ of order r in d variables,

and a remainder interval I, which encloses the Lagrange remainder of the Taylor approximation. Hence, the Taylor model arithmetics use interval computation to obtain reliable enclosures not only for the error term but also for every term of the series, allowing the computation of an over-approximation of the solution function at each time point. In addition, symbolic simplifications are applied at each step, hence reducing the interval calculations and consequently delaying divergence problems, usually, associated with interval based techniques.

Definition 3.1.4. ***Taylor Model.***
$T_f := (P_{r,f}, I_{r,f})$ is called a Taylor model of order r of a function $f \Leftrightarrow \forall x \in X : f(x) \in P_{r,f}(x - x_0) + I_{r,f}$, where X is an interval, $P_{r,f}(x - x_0)$ is a Taylor approximation polynomial of order r around the point x_0. An interval $I_{r,f}$ is called a remainder bound of order r of f on $X \Leftrightarrow \forall x \in X : R_{r,f}(x - x_0) \in I_{r,f}$.

The basic arithmetic rules on Taylor models are defined as follows [13, 77]:

- *Addition:* $T_{r,f+g} \triangleq T_{r,f} + T_{r,g} = (P_{r,f} + P_{r,g}, I_{r,f} + I_{r,g})$
- *Scalar multiplication:* $T_{r,\alpha f} \triangleq \alpha T_{r,f} = (\alpha P_{r,f}, \alpha I_{r,f})$, $(\alpha \in R)$
- *Multiplication:* $T_{r,fg} \triangleq T_{r,f} T_{r,g} = (P_{r,fg}, I_{r,fg})$

  with:

  - $P_{r,f} P_{r,g} = P_{r,fg} + P_e$
  - $P_e \in I_{P_e}$
  - $P_{r,f} \in I_{P_{r,f}}$
  - $P_{r,g} \in I_{P_{r,g}}$
  - $I_{r,fg} \triangleq I_{P_e} + I_{P_{r,f}} I_{r,g} + I_{r,f}(I_{P_{r,g}} + I_{r,g})$

where $I_{P_{r,f}}$ and $I_{P_{r,g}}$ are the interval evaluations of $P_{r,f}$ and $P_{r,g}$ respectively. I_{P_e} is the interval evaluation of P_e, which is a polynomial composed of terms with order greater than r.

Similar to interval arithmetics, algorithms supporting such Taylor models are used to produce bounded envelopes for the reachable states not only at some discrete time points but also for all continuous ranges of intermediate states between any two consecutive time discrete points. The fact that the generated bounds provide a sound abstraction for the reachable states, makes it attractive for use with formal verification techniques. Based on the above rules, the Taylor model method extends mathematical operations and functions to Taylor models such that the inclusion relationships are preserved. This is demonstrated by the following theorem:

Theorem 3.1.2. [77] Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a continuous function, F be an inclusion function of f as in Definition 3.1.3 and $f \in T$, where T is the Taylor model of f, then $T \subseteq F$. Moreover, for two functions $f_1 \in T_1$ and $f_2 \in T_2$, we have $(f_1 + f_2) \in T_S$ and $(f_1.f_2) \in T_P$, where T_S and T_P are Taylor models for the sum and product of T_1 and T_2, respectively.

In practice, the evaluation of a function is transformed to symbolically computing the Taylor polynomial $p_r(x)$ of the function, which will be propagated throughout the evaluation steps. Only the interval remainder term and polynomial terms of orders higher than r, which are usually small, are bounded using intervals as described by the rules mentioned above and are processed according to the rules of interval arithmetic. This will be demonstrated by the following example:

Example 3.1.1. In non-linear analog circuits, voltages and currents can be described using analytic functions. For example, in the differential stage shown in Figure 3.1 [46], the BJT transistor collector current is described as $i_C = I_S e^{\frac{V_{BE}}{V_T}}(1 + \frac{V_{CE}}{V_A})$, where I_S is the saturation current, V_T is the thermal voltage, V_{CE} is the output voltage of a differential stage and V_A is the Early voltage and V_{BE} is the base emitter voltage. In such case, for transistor

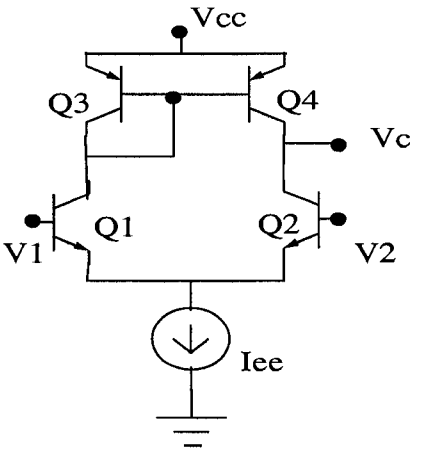


Figure 3.1: Emitter Collector Differential Stage

Q_4, $V_{CE} = tanh(y) + K$, where K is an arbitrary voltage, $y = \frac{V_i}{2V_T}$, with $V_1 = V_2 = \frac{V_i}{2}$. Consider the Taylor models T_1 and T_2 of the functions e^x, and $tanh(y)$, respectively, where $x = \frac{V_{BE}}{V_T}$, the multiplication $e^x tanh(y)$ can be done using Taylor model arithmetic of two Taylor models of order 3.

Let $x, y \in W = [-0.693, 0.693]$ and $T_1(x) := 1 + x + \frac{x^2}{2} + [-0.11, 0.11]$ and $T_2(y) := y - \frac{y^3}{3} + [-0.108, 0.108]$. It holds that:

$$\begin{aligned} T_1(x)T_2(y) \in\ & (1 + x + \tfrac{x^2}{2})(y - \tfrac{y^3}{3}) + (1 + x + \tfrac{x^2}{2}) \\ & [-0.108, 0.108] + (y - \tfrac{y^3}{3})[-0.11, 0.11] + \\ & [-0.11, 0.11][-0.108, 0.108] \\ \subseteq\ & -\tfrac{1}{6}x^2y^3 - \tfrac{xy^3}{3} - \tfrac{y^3}{3} + \tfrac{x^2y}{2} + xy + y + \\ & (1 + W + \tfrac{W^2}{2})[-0.108, 0.108] + \\ & (W - \tfrac{W^3}{3})[-0.11, 0.11] + [-0.218, 0.218] \\ \simeq\ & -\tfrac{y^3}{3} + \tfrac{x^2y}{2} + xy + y + [-0.62, 0.54] \end{aligned}$$

3.1.5 Symbolic Simulation

Symbolic simulation is a form of simulation where many possible executions of a system are considered simultaneously. This is typically achieved by abstracting the domain

over which the simulation takes place. A symbolic variable can be used in the simulation state representation in order to refer to multiple executions of the system. For each possible valuation of these variables, there is a concrete system state that is being indirectly simulated. The symbolic simulation described in this section rely on rewriting rules based on the algorithms developed in [3] for digital systems. In the context of functional programming and symbolic expressions, we define the following functions.

Definition 3.1.5. ***Substitution.***
Let u and t be two distinct terms, and x a variable. We call $x \rightarrow t$ a substitution rule. We use $Replace(u, x \rightarrow t)$, read "replace in u any occurrence of x by t", to apply the rule $x \rightarrow t$ on the expression u.

The function *Replace* can be generalized to include a list of rules. *ReplaceList* takes as arguments an expression *expr* and a list of substitution rules $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$. It applies each rule sequentially on the expression. The symbolic simulation function $ReplaceRepeated(Expr, \mathcal{R})$ shown in Definition 3.1.6 below is based on rewriting by repetitive substitution, which applies recursively a set of rewriting of rules $\mathcal{R}$ on an expression *Expr* until a fixpoint is reached.

Definition 3.1.6. ***Repetitive Substitution.***
Repetitive Substitution is defined using the following procedure:
$ReplaceRepeated(expr, \mathcal{R})$

Begin
Do
$expr_t = ReplaceList(expr, \mathcal{R})$
$expr = expr_t$
Until $FP(expr_t, \mathcal{R})$
End

$ReplaceRepeated(expr, \mathcal{R})$ applies a set of rules $\mathcal{R}$ on an expression *expr* until a fixpoint is reached, as shown in Definition 3.1.7.

Definition 3.1.7. ***Substitution Fixpoint.***

A substitution fixpoint $FP(expr, \mathcal{R})$ is obtained, if:

$$Replace(expr, R) \equiv Replace(Replace(expr, \mathcal{R}), \mathcal{R})$$

Depending on the type of expressions, we distinguish the following kinds of rewriting rules:

Polynomial Symbolic Expressions R_{Math}: are rules intended for the simplification of polynomial expressions ($\mathbb{R}^n[x]$).

Logical Symbolic Expressions R_{Logic}: are rules intended for the simplification of Boolean expressions and to eliminate obvious ones like $(and(a,a) \rightarrow a)$ and $(not(not(a)) \rightarrow a)$.

If-formula Expressions R_{IF}: are rules intended for the simplification of computations over *If-formulae*. The definition and properties of the IF function, like reduction and distribution, are defined as follows (see [84] for more details):

- IF Reduction: $IF(x,y,y) \rightarrow y$
- IF Distribution: $f(A_1,\ldots,IF(x,y,z),\ldots,A_n) \rightarrow$ $IF(x, f(A_1,\ldots,y,\ldots,A_n), f(A_1,\ldots,z,\ldots,A_n))$

Interval Expressions R_{Int}: are rules intended for the simplification of interval expressions.

Interval-Logical Symbolic Expressions $R_{Int-Logic}$: are rules intended for the simplification of Boolean expressions over intervals.

Taylor expressions: R_{Tlr} are rules intended for the simplification of Taylor model expressions ($T_{r,f}$)

Example 3.1.2. ***Horner Form Rules.*** One interval expressions R_{Int} simplification rule we use is *the Horner form transformation* [85] of a polynomial. For instance, for the univariate $p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_kx^k$, the horner form is a polynomial $q(x) = a_0 + x(a_1 + \ldots + x(a_{k-1} + a_kx))$. The interval evaluation of $q(x)$ is often more precise than the one of $p(x)$. This property is a direct consequence of the subdistributivity property of interval arithmetics. For example, let $x \in [-1,1]$, we have $x^4 \in [0,1] \subseteq [-1,1] \ni x \times x^3$

The symbolic computation uses the repetitive substitution $ReplaceRepeated(Expr, \mathcal{R})$ (Definition 3.1.6) over the set of rules defined above as follows:

Definition 3.1.8. ***Symbolic Computation.***
A symbolic computation over an expression $X_i(n)$ is defined as:

$$Symbolic\_Comp(X_i(n)) = ReplaceRepeated(X_i(n), R_{simp})$$

where $R_{simp} = R_{Math} \cup R_{Logic} \cup R_{IF} \cup R_{Tlr} \cup R_{Int} \cup R_{Int-Logic}$

The correctness of this algorithm and the proof of termination and confluence of the rewriting system formed by all above rules are discussed in [3].

Example 3.1.3. The objective of the symbolic computation is to obtain a normal form (as defined in [84]) for cases like $a + IF(x > 0, b, a)$. This expression will be normalized using two rules:

- IF Distribution : $a + IF(x > 0, b, a) \rightarrow IF(x > 0, b + a, a + a)$
- Polynomial Addition: $IF(x > 0, b + a, a + a) \rightarrow IF(x > 0, b + a, 2a)$

3.2 Modeling AMS Designs

The dynamical behavior of AMS designs is usually represented through equations describing the progressive change of the state variables. These state variables can be regarded as memory elements that are able to preserve previous states for a certain time

interval. For instance at the circuit level capacitance can be seen as a voltage storage element while inductance as a current storage element[1]. At higher level of design abstraction, a delay element can be used to affect the notion of state. In digital design, sequential logic circuits are clocked designs that have memory characteristic. An AMS model can be defined formally as follows:

Definition 3.2.1. *AMS Model.*

An *AMS* Model is a tuple $\mathcal{AMS} = (\mathcal{X}, \mathcal{X}_0, \mathcal{D}, \mathcal{D}_0, \mathcal{U}, \mathcal{F})$, with $\mathcal{X} \subseteq \mathbb{R}^d$ is the analog state space with d-dimensions, where d is the total number of state variables in the design. $\mathcal{X}_0 \subseteq \mathcal{X}$ is the set of initial states (e.g., initial voltages on the capacitances and initial currents through the inductance). $\mathcal{D} \subseteq \mathbb{K}^{d_2}$ are discrete variables (i.e., $\mathbb{K}$ is a numerical domain ($\mathbb{B}$ or $\mathbb{N}$))[2], with initialization $\mathcal{D}_0 \subseteq \mathcal{D}$. $\mathcal{U} \in \mathbb{R}^j$ is the set of possible input signal to the AMS design and $\mathcal{F} : \mathcal{X} \times \mathcal{D} \times \mathcal{U} \rightarrow \mathbb{R}^d$ is the vector field.

3.2.1 Discrete-Time AMS Designs

The notion of recurrence equation was extended in [3] to describe digital circuits using what is called generalized `If-formula`.

Definition 3.2.2. ***A System of Recurrence Equations (SRE).***

Consider a set of variables $x_i(n) \in \mathbb{K}$, $i \in \{1, \ldots, d\}$, $n \in \mathbb{N}$, an SRE is a system consisting of a set of equations of the form:

$$x_i(n) = f_i(x_j(n-\gamma)), (j, \gamma) \in \varepsilon_i, \forall n \in \mathbb{Z}$$

where $f_i(x_j(n-\gamma))$ is a generalized `If-formula`. The set ε_i is a finite non-empty subset of $1, \ldots, d \times \mathbb{N}$, with $j \in \{1, \ldots, d\}$. The integer γ is called the delay.

[1] It is worth noting that a resistance is a memoryless element.

[2] We refer to variables with discrete amplitudes as discrete variables. This should not be confused with discrete-time variables which are variables that are assigned values at discrete time points. For example, if the discrete domain is (0,1), then the variable is called boolean variable. In addition, in here, discrete variables are not states, rather they can be thought of as discrete locations such that we assign to each location a set of continuous states based on a predefined (switching) conditions.

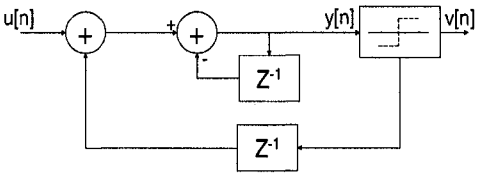


Figure 3.2: First-order ΔΣ Modulator

Example 3.2.1. Figure 3.2 shows a first-order ΔΣ of one-bit with two quantization levels, $+1$V and -1V. The quantizer (input signal $y(n)$) should be between -2V and $+2$V in order to not be overload. The SRE of the ΔΣ is :

$$y(n) = y(n-1) + u(n) - v(n-1)$$
$$v(n-1) = IF(y(n-1) > 0, 1, -1)$$

3.2.2 Continuous-time AMS Designs

Continuous-time AMS (CT-AMS) designs can be simplified to the composition of basic analog components, connected to some digital components, i.e., sequential logic and combinational logic. In this thesis, we will restrict our focus to the class of AMS, whose memory constituents are only capacitance (voltage storage) and inductance (current storage). In other words, we will assume that the digital parts can be only composed of combinational logic. The reason for such restriction is the requirement to restrict the notion of time over which the states evolve to only continuous time.

The behavior of a CT-AMS design, is governed by a system of generalized differential equations. A generalized differential equation is a non-linear equation of the form $\dot{x} = \mathcal{F}(\mathbf{x}, \mathbf{u}, t)$, whose right hand side is a $generalizedIf - formula$. More formally, the behavior of a CT-AMS design is described as follows:

Definition 3.2.3. ***Generalized System of ODEs.***

Consider a set of variables $x_k(t) \in \mathbb{R}$, $i \in \{1, \ldots, d\}$, $t \in \mathbb{R}$, a Generalized System of ODEs is a system consisting of a set of equations of the form:

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

where $\mathbf{x}(t)$ is a vector of analog state variables defining the voltage across the capacitance and the current through the inductance. $\mathbf{u}(t) \in \mathbb{R}^j$ are variables defining the input signal. The vector field $\mathcal{F}_k$ is defined as *continuous-time If-formula.*

For example, the discrete behavior of the CT-AMS can be due to a change in the input signal amplitude u, or abrupt changes in design parameters or even changes in the function $\mathcal{F}$ based on some control logic or switching conditions. The most common situation, however, is when the system equations are piecewise in the system states $\mathbf{x}$. Such a model arises for example in the linearization of the nonlinear system around different operating points.

The semantics of the AMS model [3]. $\mathcal{AMS} = (\mathcal{X}, \mathcal{X}_0, \mathcal{D}, \mathcal{D}_0, \mathcal{U}, \mathcal{F})$ over a continuous time period $T_c = [\tau_0, \tau_1] \subseteq \mathbb{R}^+$ ($t_1 = \infty$ in case of complete behavior) can be described as a trajectory $\Phi_x : T_c \rightarrow \mathcal{X}$ for $x \in \mathcal{X}_0$ such that $\Phi_x(t)$ is the solution of $\dot{x_k} = \mathcal{F}_k(x_1, \ldots, x_d)$, with initial condition $\Phi_x(0) = x$ and $t \in T_c$, is a time point.

Example 3.2.2. One of the interesting circuits used in RF designs is the Colpitts oscillator. The circuit diagram for the Colpitts circuit is shown in Figure 3.3 [33]. The circuit is composed of a MOS transistor with a constant $V_g = 0.6$, $Vcc = 1.2$, two capacitors C_1 and C_2, an inductor L, a resistance R_L and a current source I_{ee} connected to the source of the transistor.

The simplified equations are described as follows:

$$\begin{aligned} \dot{Vc_1} &:= \frac{1.2-(Vc_1+Vc_2)}{R*C_1} + \frac{I_l}{C_1} - \frac{I_{ds}}{C_1} \\ \dot{Vc_2} &:= \frac{-Iee}{C_2} + \frac{1.2-(Vc_1+Vc_2)}{R*C_2} + \frac{I_l}{C_2} \\ \dot{I_l} &:= \frac{1.2-(Vc_1+Vc_2)}{L} \end{aligned}$$

[3]Throughout the thesis, we refer to the AMS model in Definition 3.2.1 as CT-AMS model and DT-AMS model if the vector field $\mathcal{F}$ is defined using ODEs and SREs respectively.

with

$$
\begin{aligned}
I_{ds} \;:=\; & If[(Vc_1 + Vc_2 \geq 0.3 \wedge V_{c_2} < 0.3), \tfrac{k_p}{2} * \tfrac{w}{l} * (0.3 - Vc_2)^2), \\
& If[(Vc_1 + Vc_2 < 0.3 \wedge V_{c_2} < 0.3), \\
& k_p * \tfrac{w}{l} * ((0.3 - Vc_2) * (Vc_1) - 0.5 * (Vc_1)^2), 0]]
\end{aligned}
$$

where w is the gate width, l is the gate length, $|V_t| = 0.3$ is the threshold voltage of the device and K_p is a constant depending on the physics of the device.

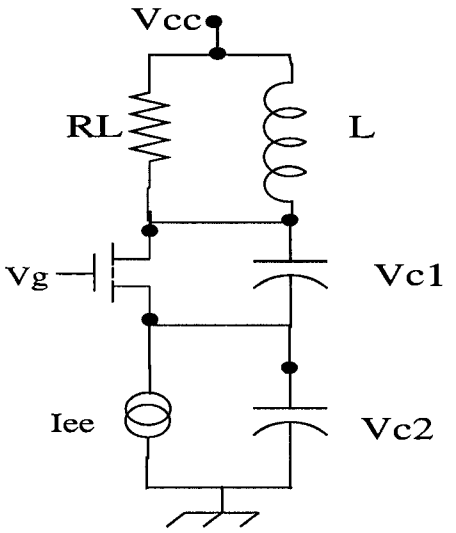


Figure 3.3: Colpitts Circuit Diagram

Note: We assume that we have correct initial conditions that are consistent with the laws of voltages and currents in the circuit [111]. We also assume that the generalized differential equation has a unique solution for each initial value (see [7] for more information about existence and uniqueness of solutions for piecewise systems).

We can model explicitly the possible trajectories of the AMS model using the notion of timed state sequence, which we refer to as *CT-AMS Trace*.

Definition 3.2.4. ***CT-AMS Trace.***

Given a sequence of time stamps τ, a trace of an AMS model is an extended timed state

sequence (σ, τ, λ), where:

- $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_n$ is a sequence of states, for every $n \in \mathbb{N}$, $\sigma_i \in \mathbb{R}^d$
- $\tau = t_0, t_1, \ldots, t_n$ is an increasing sequence of time intervals with the following condition:

  $\forall i \in \mathbb{N}$, $\exists T_i \in \mathbb{R}^+$ such that there exists a trajectory $\Phi_x(T_i) = \sigma_i$ and $T_i = t_i$ and $x \in X_0$[4]
- λ is a mapping function described as $\lambda : \mathbb{R}^d \rightarrow \mathbb{B}^j$, which is a function associating each analog state with a set of predicates $\mathbf{B}$ such that $\lambda(\sigma_i) = \mathbf{B}$ iff $\mathbf{B}(\Phi_x(T_i)) = True$.

Note: It is clear from the above definition that the behavior of a CT-AMS design can be described using analog states. In here, the discrete/digital part of the design is reduced to some predicates that control the switching between the different analog behaviors of the design. We can think of a CT-AMS trace as a concatenation of simple analog traces for which the initial state of an analog trace is in fact the final state of the previous analog trace in the concatenation. We assume that there is no ambiguity in switching conditions, meaning that each switching condition leads to only one new analog dynamic, thus avoiding non-determinism.

The complete behavior of the CT-AMS design can be specified as the set of all possible CT-AMS traces which can be used to construct the corresponding transition system:

Definition 3.2.5. ***CT-AMS Transition System.***

The transition system for CT-AMS model $\mathcal{AMS}$ is described as a tuple $\mathcal{T}_{\mathcal{AMS}} = (Q, Q_0, \sigma, L)$ where $q \in Q$ is a configuration (x, z, Γ), $x \in \mathcal{X}$, $z \in \mathbb{B}^j$ and set of time intervals Γ where $\cup_{i \geq 0} t_i \subseteq \mathbb{R}^+$, $t_i \in \Gamma$. We have $t_1, t_2 \in \Gamma$ for $\Phi_{x'}(t_1) = \Phi_{x''}(t_2) = x$ and $x', x'' \in X_0$. $q \in Q_0$, when $t_0 \in \Gamma$ and t_0 is the singular interval $(t_0 = 0)$, L is an interpretation function such that

[4]Note that we slightly abused the definition of a trajectory, where we assume that the domain is a set of time intervals rather than a set of time points, i.e., $\Phi_x(T_i) = \{\Phi_x(T_i) | T_i \in T_i, l \in \mathbb{N}, \tau_i \in \mathbb{I}\}$.

$L : Q \rightarrow \mathbb{R}^n \times 2^{\mathbb{B}^j} \times 2^{\mathbb{R}^+}$. Finally, $\sigma \subseteq Q \times Q$ is a transition relation such that $(q_n, q_m) \in \sigma$ iff $\exists t_n \in \Gamma_n$, $\exists t_m \in \Gamma_m$. $t_n < t_m$ and $\lim_{t_n \rightarrow t_m} \Phi_x^{q_n}(t_n) = \Phi_x^{q_m}(t_m)$, $x \in \mathcal{X}_0$, where trajectory $\Phi_x : T_c \rightarrow \mathcal{X}$ for $x \in \mathcal{X}_0$ over a continuous time period $T_c = [\tau_0, \tau_1] \subseteq \mathbb{R}^+$ ($t_1 = \infty$ in case of complete behavior), such that $\Phi_x(t)$ is the solution of $\dot{x_k} = \mathcal{F}_k(x_1, \ldots, x_d)$, with initial condition $\Phi_x(0) = x$ and $t \in T_c$, is a time point.

3.2.3 Approximating the Behavior of CT-AMS Designs

Obtaining the complete behavior of CT-AMS designs is often a hard problem as it requires finding a closed form solution of the system equations. Such a solution is hard to get in practice for the general equations. Therefore, an approximation that guarantees preserving the behavior of the system must be used instead. One possible methods to approximate the continuous behavior is by using Taylor approximation described in Section 3.1.2.

Example 3.2.3. Consider the analog circuit in Figure 3.4, composed of a network of passive components (capacitors and conductances), along with non-linear current sources and two switches. The switches can be designed using CMOS transistors working in saturation mode as shown in the figure. This circuit exhibits an oscillatory behavior when the initial capacitor voltages are within a specified range, based on the switches positions. The voltages across the capacitors can be described using ODEs as follows:

$$\begin{cases} \dot{v_{c1}} = v_{c2} \text{ or } \dot{v_{c1}} = v_{c2} + v_{c2}^3 \\ \dot{v_{c2}} = -v_{c1} + v_{c1}^3 \text{ or } \dot{v_{c2}} = -v_{c1} + (1/2)v_{c1}^3 \end{cases}$$

Suppose that we specify the switching conditions as

$$Cond_1 = Cond_2 := v_{c1}(n-1) \leq v_{c2}(n-1)$$

For illustration purposes and for clarity, we use Taylor approximation limited to order 2 to obtain the corresponding SREs:

$$v_{c1}(n) := IF(Cond_1, X_1, X_2) \quad and \quad v_{c2}(n) := IF(Cond_2, Y1, Y2)$$

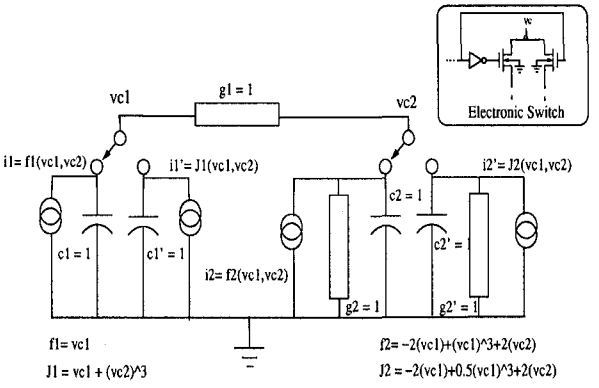


Figure 3.4: Switched Analog Circuit

with:

- $X_1 := \frac{h^2 v_{c1}(n-1)^3}{2} - \frac{h^2 v_{c1}(n-1)}{2} + v_{c1}(n-1) + h v_{c2}(n-1) + \mathcal{R}m_1[\widetilde{v_{c1}}, \widetilde{v_{c2}}]$

- $X_2 := \frac{h^2 v_{c1}(n-1)^3}{4} + \frac{3}{4}h^2 v_{c2}(n-1)^2 v_{c1}(n-1)^3 - \frac{h^2 v_{c1}(n-1)}{2} - \frac{3}{2}h^2 v_{c2}(n-1)^2 v_{c1}(n-1) + v_{c1}(n-1) + h v_{c2}(n-1)^3 + h v_{c2}(n-1) + \mathcal{R}m_2[\widetilde{v_{c1}}, \widetilde{v_{c2}}]$

- $Y_1 := h v_{c1}(n-1)^3 + \frac{3}{2}h^2 v_{c2}(n-1) v_{c1}(n-1)^2 - h v_{c1}(n-1) - \frac{h^2 v_{c2}(n-1)}{2} + v_{c2}(n-1) + \mathcal{R}m_3[\widetilde{v_{c1}}, \widetilde{v_{c2}}]$

- $Y_2 := \frac{h v_{c1}(n-1)^3}{2} + \frac{3}{4}h^2 v_{c2}(n-1)^3 v_{c1}(n-1)^2 + \frac{3}{4}h^2 v_{c2}(n-1) v_{c1}(n-1)^2 - h v_{c1}(n-1) - \frac{h^2 v_{c2}(n-1)^3}{2} - \frac{h^2 v_{c2}(n-1)}{2} + v_{c2}(n-1) + \mathcal{R}m_4[\widetilde{v_{c1}}, \widetilde{v_{c2}}])$

where $\mathcal{R}m_i[\widetilde{v_{c1}}, \widetilde{v_{c2}}]$ are the Taylor approximation remainders, $i = \{1, \ldots, 4\}$ and h is the time step.

However, in order to ensure the correctness of the analysis, we must define a sufficient condition for an adequate approximation. In order to define a notion of abstraction precisely, we establish a correspondence between a discrete $\theta : \mathbb{N} \mapsto \mathcal{X}$ and a continuous trajectories $\Phi_x : [0, \infty) \mapsto \mathcal{X}$. This is done using discrete sampling.

Definition 3.2.6. ***Sufficient Trajectory Discretization.***

A discrete evolution $\theta : \mathbb{N} \mapsto \mathcal{X}$ is a sufficiently complete discretization of a continuous evolution $\Phi_x : [0,\infty) \mapsto \mathcal{X}$ if there exists a strictly increasing sequence of reals in the interval $[0,\infty)$ such that $t_0 = 0$, Φ_x does not change in either the domain $(t_i,t_{i+1}]$ or the domain $[t_i,t_{i+1})$, that is either $\|\Phi_x(t) - \Phi_x(t')\| \leq \varepsilon$ for all $t,t' \in (t_i,t_{i+1}]$ or $t,t' \in [t_i,t_{i+1})$, where ε is the sampling error and $\theta(i) = \Phi_x(t_i)$ for all i.

Intuitively, a sufficiently complete discretization captures all the different continuous states in the continuous evolution. In general, we have $\|\theta(i) - \Phi_x(t_i)\| \leq \varepsilon$ for all i, where ε is the discretization error and exact valuation cannot be achieved. We can explicitly model the possible trajectories of the sampled AMS model as a *Sampled CT-AMS Trace.*

Definition 3.2.7. ***Sampled CT-AMS Trace.***

A timed state sequence (σ',τ',λ') is a sampled CT-AMS trace of a CT-AMS model such that:

- If (σ,τ) is a CT-AMS trace of a continuous evolution Φ_x and $\theta : \mathbb{N} \mapsto \mathcal{X}$ is a sufficiently complete discretization of $\Phi_x : [0,\infty) \mapsto \mathcal{X}$, then there exists a trajectory such that:$\forall i \in \mathbb{N}$, $\theta(i) = \sigma'_i$ with $\|\sigma_i - \sigma'_i\| \leq \varepsilon$ and $t'_i \in (t_i,t_{i+1}]$ or $t'_i \in [t_i,t_{i+1})$.
- λ' is a mapping function described as $\lambda' : \mathbb{R}^{d_1} \rightarrow \mathbb{B}^j$, which is a function associating to each analog state a set of predicates $\mathbf{B}$ such that $\lambda'(\sigma_i) = \mathbf{B}$ iff $\mathbf{B}(\Phi_x(T_i)) = True$.

We can then view the sampled behavior of an CT-AMS model as a transition system, which can be constructed from the set of all possible sampled traces (trajectories). We define a sampled CT-AMS transition system as follows:

Definition 3.2.8. ***Sampled CT-AMS Transition System.***

A Sampled CT-AMS Transition System $\mathcal{T}_S$ is a tuple (Q',Q'_0,δ',L'), $q \in Q'$ is a configuration (x,z,Δ_x), $x \in \mathcal{X}$, $z \in \mathbb{B}^j$ and set Δ_x where $t_{i_1},t_{i_2} \in \Delta_x$ if $\theta(i_1) = \theta(i_2) = x$. $Q'_0 \subseteq Q'$ is the set of all initial configurations. L' is an interpretation function such that

$L' : Q' \rightarrow \mathbb{R}^n \times 2^{\mathbb{B}^j} \times 2^{\mathbb{N}}$. Finally, $\delta' \subseteq Q' \times Q'$ is a transition relation such that such that $\theta : \mathbb{N} \mapsto \mathcal{X}$ satisfying initial condition: $\theta(0) \in Q'_0$ and discrete evolution $\forall i \in \mathbb{N}$, $(\theta(i), \theta(i+1)) \in \delta'$.

Statement 1. We say that a Sampled AMS Transition System $\mathcal{T}_{\mathcal{S}}$ is an approximation of a CT-AMS Transition System $\mathcal{T}_{\mathcal{AMS}}$, denoted $\mathcal{T}_{\mathcal{S}} \cong \mathcal{T}_{\mathcal{AMS}}$, if the discrete evolution in the former and the continuous evolution of the latter are related according to Definition 3.2.6.

It is thus natural to look for a model that gives a sufficiently accurate answer to the analysis. In practice, it is hard to fulfill such condition; however, some approximation techniques under certain conditions can lead to a model that preserve the original behavior of the system but with the cost of introducing more (undesirable) behaviors. Such approximations are referred to in formal methods literature as over-approximation techniques [25].

In practice, to ensure the sufficient approximation criteria, the goal of a numerical approach (like Taylor approximation) for solving an initial value problem (IVP) over an interval range of t is to approximate as accurately as possible its solution at some discrete points placed along that interval. Usually, by starting at point t_0 (whose solution value is known: $\mathbf{x}(t_0) = \mathbf{x}_0$) an increasing (decreasing) sequence of discrete points is considered by adjusting the step size (the gap between two consecutive discrete points) as the calculation proceeds. The purpose of this adaptive step size policy is to keep some control over the accuracy of the approximation. However, a common source of errors is the discretization error (also known as truncation error), which is partially due to propagation of errors made at previous steps (from t_0 to t_i) along with the current step. To preserve the inherited behavior of the actual solution, the remainder term should not be discarded and instead bounds must be specified. Interval approaches attempt to produce bounds for the solution flow not only at some discrete points of t but also for all the continuous range of intermediate values between any two consecutive discrete points. In this case, we can allow for over-approximation of behavior, but guaranteeing the sufficient approximation required

to ensure sound construction of approximate model of the CT-AMS designs. Having attained this goal, we can claim that achieved recurrence equations can be suitable under certain conditions for modeling continuous-time AMS systems, hence allowing a unified modeling framework for discrete-time and continuous-time AMS designs. In the remainder of this section, we will provide a procedure to obtain such approximation based on Taylor theorem and interval arithmetics.

3.2.4 Interval Abstraction

As outlined earlier, to preserve the inherited behavior of the actual solution, the remainder term of the Taylor approximation should not be discarded and instead bounds must be specified. Interval approaches [85] attempt to produce bounds for the solution flow not only at some discrete points of t but also for all the continuous range of intermediate values between any two consecutive discrete points. In this case, we can allow for over-approximation of behavior, but guaranteeing a sufficient approximation requires a sound construction of the approximate model of the AMS design.

Interval domains are numerical domains that enclose the original states of a system of equations at each discrete step [85]. Interval methods produce boundeding envelopes for the reachable states not only at some discrete time points but also for all continuous ranges of intermediate states between any two consecutive time discrete points. Solution methods for ODEs based on Interval arithmetics, also known as *validated methods*[85], are an attractive tool to use in the verification of the behavior of systems with uncertainty on the design parameters or initial conditions as they allow sound discretization.

Interval Abstraction for the Traces. Given a Taylor based approximation of a system of ODEs, we can describe its trajectories starting from a set of initial conditions by the notion of interval analog traces.

Definition 3.2.9. ***Interval AMS Trace.***

An interval AMS trace of a CT-AMS design is a timed state sequence $(\tilde{\sigma}, \tilde{\tau}, \tilde{\lambda})$, such that:

- $\tilde{\sigma} = \tilde{\sigma_0}, \tilde{\sigma_1}, \ldots, \tilde{\sigma_n}$ is a sequence of states for every $n \in \mathbb{N}$, $\tilde{\sigma_i} \in \mathbb{I}^d$.
- $\tilde{\tau} = t_0, t_1, \ldots, t_n$ is a sequence of time intervals stamps with the following condition: $\forall i \in \mathbb{N}$, there exists an interval evaluation of a Taylor approximation trajectory $\mathbf{x}(T_i) = \tilde{\sigma_i}$ with $t_i = (T_{i-1}, T_i]$.
- $\tilde{\lambda}$ is a mapping function described as $\tilde{\lambda} : \mathbb{R}^{d_1} \rightarrow \mathbb{B}^j$, which is a function associating to each analog state a set of predicates $\mathbf{B}$ such that $\tilde{\lambda}(\tilde{\sigma_i}) = \mathbf{B}$ iff $\mathbf{B}(\mathbf{x}(T_i)) \neq \mathit{False}$.

The concepts of inclusion function and inclusion test can be used to define an abstraction from the concrete traces to corresponding interval traces as follows:

Definition 3.2.10. ***Trace Abstraction.***
Let $\mathcal{T}r_a = (\sigma, \tau, \lambda)$ be a CT-AMS trace and $\mathcal{T}r_i = (\tilde{\sigma}, \tilde{\tau}, \tilde{\lambda})$ be an Interval AMS trace. We say $\mathcal{T}r_i$ is an abstraction of $\mathcal{T}r_a$ if there exists a map $abs : \mathcal{X} \rightarrow \mathbb{I}^d$ such that $abs(\sigma_0) \subseteq \sigma'_0$ and for every $\sigma_i \in \sigma$, if σ'_i is a sufficiently complete discretization of σ_i, then $abs(\sigma_i) = abs(\sigma'_i) \in \sigma'$

We can argue that for each concrete trace, we can find an associated interval trace that over-approximates it, in a way that preserves its properties and that for a given abstraction, the set of all possible concrete traces is a subset of the set of interval based traces that can be generated by the system.

Lemma 3.2.1. ***Existence of Trace Abstraction.***
Given a bounded time CT-AMS trace, we can always find an interval AMS trace which is an abstraction of that trace.

Proof. By Weierstrass Approximation [39] and existence of solution for validated methods [85].
Weierstrass Approximation ensures that any continuous function on a closed and bounded interval can be uniformly approximated on that interval by polynomials to any degree of

accuracy. Validated methods provide techniques to construct such approximation.

We can represent the AMS design behavior over intervals using a state transition system as follows:

Definition 3.2.11. ***Interval based State Transition System.***
An *Interval based State Transition System* is a tuple $\mathcal{T}_I = (S_I, S_{I,0}, \rightarrow_{\delta_I})$, where S_I is the interval state space, $S_{I,0} \subseteq S_I$ is the set of initial interval states, $\rightarrow_{\delta_I} \subseteq S_I \times S_I$ is a relation defined using SRE forms δ_I and capturing the abstract transition between interval states such that:

$$\{s \rightarrow_{\delta_I} s' | \exists a \in s, \exists b \in s' : b = \delta_I(a) \ \textit{and} \ \delta \in \delta_I\}$$

where $a, b \in \mathbb{R}^d$, $s, s' \in S_I$, $\delta = \{f_1, \ldots, f_d\}$ with $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ is an *if-formula*, $i \in \{1, \ldots, d\}$, $\delta_I = \{f_1^I, \ldots, f_d^I\}$ and $f_i \in f_i^I$, where f_i^I is the interval extension of the *if-formula* f_i.

Statement 2. We say that an *Interval based State Transition System* $\mathcal{T}_I$ is an abstraction of a CT-AMS State Transition System $\mathcal{T}_{\mathcal{A}}$ if $Abs(\mathcal{T}_{\mathcal{A}}) \subseteq \mathcal{T}_I$, and we denote it as $\mathcal{T}_{\mathcal{A}} \preceq \mathcal{T}_I$

Unfortunately, due to the over-approximation nature of interval analysis, a quick divergence in the reachability calculation generally happens. This is mainly due to the following issues [85]:

- *The dependency problem* which is the inability of interval arithmetic to identify different occurrences of the same variable. For example, $x - x = 0$ holds for each $x \in [1,2]$, but $X - X$ for $X = [1,2]$ yields $[-1,1]$.
- *The wrapping effect* which appears when the results of a computation are overestimated when enclosed into intervals, hence leading to error accumulation at each time step.

The undesirable properties associated with interval analysis can be partially avoided if instead of relying on interval traces with loose accuracy (large overapproximation),

we search for tighter enclosures that still preserve the original traces. This goal can be guaranteed with the following lemma:

Lemma 3.2.2. Let $Tr_{set}(\mathcal{T}r_a)$ be the set of all AMS traces and $Tr_{set}(\mathcal{T}r_i)$ be the set of all Interval AMS traces of a given analog systems, then $Abs(Tr_{set}(\mathcal{T}r_a)) \subseteq Tr_{set}(\mathcal{T}r_i)$

Proof. This lemma is a direct consequence of Definition 3.2.10.
In more concrete sense, Taylor models described in Section 3.1.4 satisfies these properties; moreover, they have been proved to be the best available interval based approximation [88].

3.3 Specification Languages

In order to reason about the functional properties of the designs under verification, we need a language that describes the temporal relations between the different signals of the system, including input, output and internal signals. Temporal logics are a special kind of modal logics that include operators (modalities) to reason about the truth values of assertions at different times during the execution of a program. There are two basic types of temporal logic: Linear time (e.g., Linear Temporal Logic (LTL)) and branching time (e.g., Computational Tree Logic (CTL)). Temporal logics distinguishing a linear and a branching view on time respectively. In the linear view, each point in time has exactly one future. A specification is interpreted over a linear structure, i.e., a computation is a sequence of events. In the branching view, there is a (non-deterministic) choice between several potential futures at each point in time. This results in a tree of potential computations. Neither view can, on its own, express all properties that the other can, however, there are subset of properties that can be supported by both kind of logics. In general temporal logic formulas are interpreted over state sequences of labeled transition systems called Kripke structures. The semantics of formulas is formally defined for a model (state sequence) and a formula ϕ by means of the satisfaction relation $\models$. $\sigma \models \phi$

denotes that the formula ϕ holds for the state sequence σ. A survey on temporal logic is available in [32].

For the verification purposes in this thesis, we provide the basics of two types of temporal logic; namely $MITL$ which is timed linear temporal logic and $\forall CTL$ which is a subset of the standard CTL. The motivation for choosing two different logics in the proposed verification methodology is based on the following. For BMC verification, we are interested in checking properties over a set of traces for a given amount of time. The verification idea is based on encoding each property as a set of constraints to be satisfied. In particular, LTL has been shown to be practical for such verification technique [14]. As we are extending BMC for AMS designs, which are characterized by their real-time behavior, choosing MITL as specification logic provides us with an intuitive formalism to express the required properties as will be demonstrated below. On the other hand, the predicate abstraction proposed in the thesis is based on the qualitative analysis of the AMS design state space rather than particular traces. Therefore, an untimed logic like $\forall CTL$ suffices for describing the desired properties.

3.3.1 MITL

We use a variant of Metric Interval Temporal Logic (MITL) which is an extension of LTL tailored for specifying desired timed properties of real-time designs. In MITL, temporal modalities are restricted to intervals of the form $I = [a, b]$ with $a, b \in \mathbb{Q}_{\geq 0}$. The benefit of bounding the temporal properties is to restrict the verification for a specific amount of time avoiding the non-termination. To specify analog behavior of the AMS designs, the logic is augmented with a mapping from continuous domains into propositions. We extended the MITL language with predicates over real constants and real variables. We can define atomic properties as follows:

Definition 3.3.1. ***Atomic Property.***

An atomic property $\lambda(x_1, \ldots, x_n)$ is a logical formula defined as follows: $\lambda(x_1, \ldots, x_n) =$

$\chi \diamond c$, where $\diamond \in \{<, \leq, >, \geq, =, \neq\}$, χ is an arithmetic formula over the design state variables $\mathbf{x}$ and c is an arbitrary value ($c \in \mathbb{R}$)

The main temporal operators describing properties of a trace:

- **F** ("eventually or in the future") asserts that a property will hold at some states on the path.
- **G** ("always or globally") specifies that a property holds at every state on the path.

The syntax of MITL is defined by the following grammar:

Syntax of MITL. The basic formulae of the MITL are defined by the following grammar:

$$\varphi := \lambda(x_1, \ldots, x_n) | \neg\varphi | \varphi_1 \vee \varphi_2 | \mathbf{F}_I\varphi | \mathbf{G}_I\varphi | true$$

where λ belongs to a set of atomic properties over the design state variables and x_i is a term (that is a constant or a variable). [5] **G** and **F** are temporal operators and I is an interval $I = [a, b]$ with $0 < a < b$ and $a, b \in \mathbb{Q}_{\geq 0}$ and $a \neq b$.

Semantics of MITL. We define the Kripke structure which is a transition system as in Definition 3.2.5 $\mathcal{T}_{\mathcal{AMS}} = (Q, Q_0, \sigma, L)$, extended with an interpretation function $[\![.]\!]$, written as $K = (\mathcal{T}_{\mathcal{AMS}}, [\![.]\!])$. The semantics of the language is provided by the interpretation $[\![.]\!]$ as follows:

- For a constant C, $[\![C]\!]$ is an element of $\mathbb{R}$
- For a state variable $x \in \mathbf{x}$ (where $\mathbf{x}$ is the set of state variables), $[\![x]\!]$ is a function $\mathbb{R}^+ \rightarrow \mathbb{R}$
- For an n-ary predicate λ, $n \geq 1$, the meaning $[\![\lambda]\!]$ is a function $\mathbb{R}^n \rightarrow \mathbb{B}$.

The interpretation $[\![.]\!]$ extends to arbitrary terms, inductively:

$$[\![\lambda(x_1, \ldots x_n)]\!] = [\![\lambda]\!]([\![x_1]\!], \ldots [\![x_n]\!])$$

[5] To describe properties on analog signals like current and voltages, atomic propositions, $\lambda(x_1, \ldots, x_n)(n)$, are predicates (inequalities) over reals, with time index n. The provided propositions are algebraic relations between signals (variables) of the system.

In addition, we have the concretisation function $\Upsilon_\lambda : \mathbb{B} \rightarrow 2^{\mathbb{R}^n}$ such that $\Upsilon([\![\lambda(x)]\!]) = \Upsilon_\lambda(b) = \{x \in \mathbb{R}^n | \lambda(x) = b\}$. Intuitively, Υ_λ is a set of states, where λ holds with the condition $\Upsilon_\lambda \cap \Upsilon_{\neg\lambda} = \emptyset$

In general in real-time temporal logic, observations have to be extended with information about their timing. This is done by representing a the timed state sequence as a timed word over state observations. Thus, it is a pair $\Sigma = (\sigma, \Gamma)$, consisting of a state sequence σ and an interval sequence I. We use the notations $\hat{s}(\Sigma)$ and $\hat{\tau}(\Sigma)$ for the states and respectively of timed part of the timed state sequence.

Let $\Sigma = (\sigma, I)$ be a state sequence associated with the Kripke structure, with $I = [a, b]$, the satisfaction relation $\Sigma \models \varphi$, indicating that a state sequence satisfies a property φ starting from position τ_0 and $\tau_0 \in \Gamma$ is defined inductively as follows:

- $\sigma \models true$
- $\sigma \models \lambda(y_1, \ldots y_n)$ iff $L_X(\sigma_0) \in \Upsilon([\![\lambda(y_1, \ldots y_n)]\!])$
- $\sigma \models \neg\varphi$ iff $\sigma \not\models \varphi$
- $\sigma \models \varphi_1 \vee \varphi_2$ iff $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$
- $\sigma \models \mathbf{F}_I\varphi$ iff starting from position $\underline{t}$, where $t = [\underline{t}, \overline{t}]$ and $t \in \Gamma_0$, $\exists t' \in [\underline{t} + a, \underline{t} + b]$. $\sigma \models \varphi$
- $\sigma \models \mathbf{G}_I\varphi$ iff starting from position $\underline{t}$, where $t = [\underline{t}, \overline{t}]$ and $t \in \Gamma_0$, $\forall t' \in [\underline{t} + a, \underline{t} + b]$. $\sigma \models \varphi$

Note: The verification algorithms in this thesis consider abstract models overapproximating the original behaviors. Therefore, correctness must be proved for all possible abstract behaviors. In fact, MITL has implicit universal quantifiers in front of its formulas. For example, $M \models \forall f$ means that M satisfies f over all initialized paths. Such property makes MITL an adequate for writing specifications.

3.3.2 $\forall CTL$

In Chapter 5, we will be using temporal logic to verify properties on discrete abstractions of AMS designs. For the purpose of verification, we need a temporal logic for reasoning over the possible behaviors of the design. We use a subset of CTL which only allows the use of the universal path quantifier $\forall$. We refer to this subset as $\forall$CTL [72]. $\forall$CTL formulas are specified and evaluated over the semantic model of the system; usually modelled as a Kripke structure. Beside boolean connectives, $\forall$CTL provides linear time operators and path quantifier. The linear time operators allow expressing properties of a particular behaviour of the system given by a series of events in time. Path quantifiers used with time operators account for the possible existence of multiple future scenarios starting at a given state at a point in time.

The main temporal operators describing properties of a path through the tree are :

- **F** ("eventually or in the future") asserts that a property will hold at some states on the path.
- **G** ("always or globally") specifies that a property holds at every state on the path.

Based on the path quantifiers and temporal operators, we can define state formulas and path formulas as follows.

Syntax of $\forall$CTL. Let AP be the set of atomic propositions. The $\forall$CTL is the set of state formulas on AP inductively defined as follow:

- Any boolean formula over atoms from AP using the connectives $\vee, \wedge and \neg$ is a pure state formula.
- If ϕ and φ are *state formulas*, then $\phi \wedge \varphi$and $\phi \vee \varphi$ are state formulas.
- If ϕ and φ are *state formulas*, then $\mathbf{F}\phi$, $\mathbf{G}\varphi$ are path formulas.
- If ϕ is a path formula, then $\mathbf{A}(\phi)$ is a state formula.

The semantic of a discrete model [6] under verification is usually represented by a *Kripke structure.*

Semantics of ∀CTL. The Kripke structure of a discrete model is a tuple $M = (C, C_0, R, L)$, where C is the set of all possible states for the model, $C_0 \subseteq C$ is the set of initial states, R is a transition relation between two states such that $R \subseteq C \times C$. $L : C_i \rightarrow 2^{AP}$ is a labeling function associating each state with a non-empty set of atomic propositions (AP).

Definition 3.3.2. A path π of a Kripke structure M is a finite sequence of states $\pi = [c_0, c_1, ..., c_i]$ such that $i \geq 0$. Given an integer $i \geq 0$ and a path π,we denote by π_i the i-th state of π.

Definition 3.3.3. Let c and π be a generic state and path respectively in the Kripke structure of discrete model M. Then the satisfaction relation $\models$ for state and path formulas is defined as follow :

- $c \models p$ iff $p \in L(c)$ where L(c) is the labelling function of state c
- $c \models \neg p$ iff $\neg p \in L(c)$
- $c \models \varphi \wedge \psi$ iff $c \models \varphi$ and $c \models \psi$.
- $c \models \varphi \vee \psi$ iff $c \models \varphi$ or $c \models \psi$.
- $c \models \mathbf{A}(\mathbf{G}\varphi)$ iff for every path π starting at the state c, for all states π_i along the path such that $\pi_i \models \varphi$
- $c \models \mathbf{A}(\mathbf{F}\varphi)$ iff for every path π starting at the state c, there is some states π_i along the path such that $\pi_i \models \varphi$

[6] In here, a discrete model is model representing the approximation of an AMS design using predicate abstraction as described in Chapter 5.

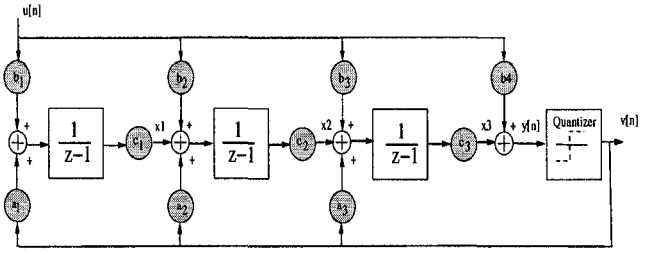


Figure 3.5: Third-order ΔΣ Modulator

3.4 Symbolic Simplification

The AMS description is composed in general of a digital part and an analog part. The analog part can be approximated using recurrence equations. The digital part can be described using event driven models. The properties that we verify are temporal relations between signals of the system. Starting with an AMS description and a set of properties, the symbolic simulator performs a set of transformations by rewriting rules in order to obtain a normal mathematical representation called a generalized system of recurrence equations (SRE) [3]. These are combined recurrence relations that describe each property blended directly with the behavior of the system.

Given a model representing the behavior of the design and a property of interest expressed in LTL, the symbolic simulation defined in Section 3.1.5 is used to obtain a unified representation adequate for applying the verification methods developed in the subsequent chapters (mainly in Chapter 4 and Chapter 6). This is illustrated with the following example.

Example 3.4.1. Data converters are needed at the interface of analog and digital processing units. The ΔΣ architecture uses several stages to make rough evaluations of the signal, measure the error, integrate it and then compensate for that error. Higher-order single stage modulators have been proposed to increase the converter's resolution by adding more integral and feedback paths. The number of integrators, and consequently, the numbers of feedback loops, indicates the order of a ΔΣ modulator. Consider the third-order

discrete-time $\Delta\Sigma$ modulator illustrated in Figure 3.5. Such class of $\Delta\Sigma$ design can be described using the vector recurrence equations:

$$X(k+1) = C\,X(k) + B\,u(k) + A\,v(k)$$

where A, B and C are matrices providing the parameters of the circuit and $u(k)$ is the input signal, $v(k)$ is the digital part of the system and $b_4 = 1$. In more detail, the recurrence equations for the analog part of the system are:

$$\begin{aligned} x_1(k+1) &= x_1(k) + b_1 u(k) + a_1 v(k) \\ x_2(k+1) &= c_1 x_1(k) + x_2(k) + b_2 u(k) + a_2 v(k) \\ x_3(k+1) &= c_2 x_2(k) + x_3(k) + b_3 u(k) + a_3 v(k) \end{aligned}$$

The condition of the threshold of the quantizer is computed to be equal to $c3x3(k) + u(k)$. The digital description of the quantizer is transformed into a recurrence equation using the approach defined in [3]. Thus, the equivalent recurrence equation that describes $v(k)$ is

$$v(k) = IF(c_3 x_3(k) + b_4 u(k) \geq 0, -a, a)$$

Applying symbolic simulation (Definition 3.1.6) for the $\Delta\Sigma$ modulator, we obtain the following unified modeling for both the analog and discrete parts.

$$\begin{aligned} x_1(k+1) &= if(c_3 x_3(k) + u >= 0, x_1(k) + b_1 u - a_1 a, \\ &\quad\; x_1(k) + b_1 u + a_1 a) \\ x_2(k+1) &= if(c_3 x_3(k) + u >= 0, c_1 x_1(k) + x_2(k) + b_2 u(k) \\ &\quad\; -a_2 a, c_1 x_1(k) + x_2(k) + b_2 u(k) + a_2 a) \\ x_3(k+1) &= if(c_3 x_3(k) + u >= 0, c_2 x_2(k) + x_3(k) + b_3 u(k) \\ &\quad\; -a_3 a, c_2 x_2(k) + x_3(k) + b_3 u(k) + a_3 a) \end{aligned}$$

The modulator is said to be stable if the integrator output remains bounded under a bounded input signal, thus avoiding overloading of the quantizer. This property is of

a great importance since the integrator saturation can deteriorate circuit performance. If the signal level at the quantizer input exceeds the maximum output level by more than the maximum error value, a quantizer overload occurs. The quantizer in the modulator shown in Figure 3.5 is a one-bit quantizer with two quantization levels, $+1$V and -1V. Hence, the quantizer input should be always between -2V and $+2$V in order to avoid overloading [50].

The stability property of the $\Delta\Sigma$ modulator is written as $\mathbf{G}P(k+1)$, where

$$P(k+1) = (x_3(k+1) > -2 \wedge x_3(k+1) < 2)$$

Applying Symbolic simulation (Definition 3.1.6), the state variable $x_3(k+1)$ is replaced by its corresponding expression and the expression of the property is defined as:

$$\begin{aligned} P(k+1) \;=\; & if(c_3x_3(k)+u >= 0, \\ & -2 < c_2x_2(k)+x_3(k)+b_3u(k)-a_3a, \\ & c_2x_2(k)+x_3(k)+b_3u(k)+a_3a < 2) \end{aligned}$$

The techniques for verifying the $\Delta\Sigma$ modulator will be presented in Chapter 4.

In this chapter, we presented the necessary concepts required for the verification approaches described in the thesis. In the next chapter, we will present a bounded model checking algorithm for continuous-time AMS designs. The basic idea will be to combine symbolic simulation and Taylor model arithmetics to verify properties on the SRE model.

Chapter 4

Bounded Model Checking for CT-AMS Designs

Model checking was initially developed as a method of complete verification through the exploration of the whole state space of the given design. But with the limited space (memory) and time resources, such complete exploration was severely limited with the state space explosion problem. The bounded model checking (BMC) [14] approach has been advocated recently as means to combat this problem, by limiting the explored state space. This is done by providing bounds on the number of cycles that should be explored.

In BMC, the transition relation and the property are unwound up to a given depth (number of cycles) to obtain a formula, which is then checked using constraints satisfiability techniques. If a counter-example is found or a fixpoint is reached, the verification task is achieved, else the number of steps can be increased for further verification. This implies that the method is incomplete in general as *a priori* calculation of the maximum cycles (depth) needed to ensure the verification is not always possible. Hence, BMC is typically used for refutation of a property rather for ensuring safety and reachability properties. Nevertheless, BMC can be an attractive tool for verification rather than refutation if some limitations are to be imposed on the type of properties to verify (e.g., bounds on the temporal operator as in the MITL language described in Chapter 3, Section 3.3).

As a matter of fact, AMS designs are usually characterized by a bounded state space (i.e., voltages and currents across a circuit are always confined within a specific ranges defined through the connection settings of the circuit components as well as the voltages applied across it.). Furthermore, many properties related to the characteristics of the designs are associated with its time bounded functionality. For instance, one interesting property is to check whether a switching will occur within a specific amount of time. In this perspective, we propose in this chapter, an approach for CT-AMS designs based on bounded model checking [14].

The proposed methodology as shown in Figure 4.1 is composed of two distinct phases: a modeling phase and a verification one. In the modeling phase, continuous-time based analog components are described using ordinary differential equations, while the digital parts of the AMS design are described using event based models. In order to obtain the verification model, which is a formed of a set of recurrence equation (Chapter 3, Section 3.2.3), the differential equations are approximated using the Taylor Approximation Theorem (Chapter 3, Section 3.1.2). Therefore the recurrence model gives the possibility of handling continuous behaviors like that of current and voltages, but in discrete time intervals, which cover a non-trivial class of mixed behaviors. In the next step, the AMS description and the MITL property of interest are input to a symbolic simulator that performs a set of transformations by rewriting rules in order to obtain the system of generalized recurrence equations (SREs).

The next phase is to prove the desired property using a verification engine that performs the state space exploration and BMC over Taylor model forms. The Taylor model form is a combined symbolic-numerical representation of the system equations using polynomials and interval terms that ensure enclosure of the reachable states. Such arithmetics allows the computation over continuous quantities while avoiding the unsoundness inherent in the numerical Taylor approximation by providing an overapproximation of the possible reachable states of the system. The BMC is composed of two sequential steps. In the first step, rules are applied on the SREs to set up the Taylor model forms (See

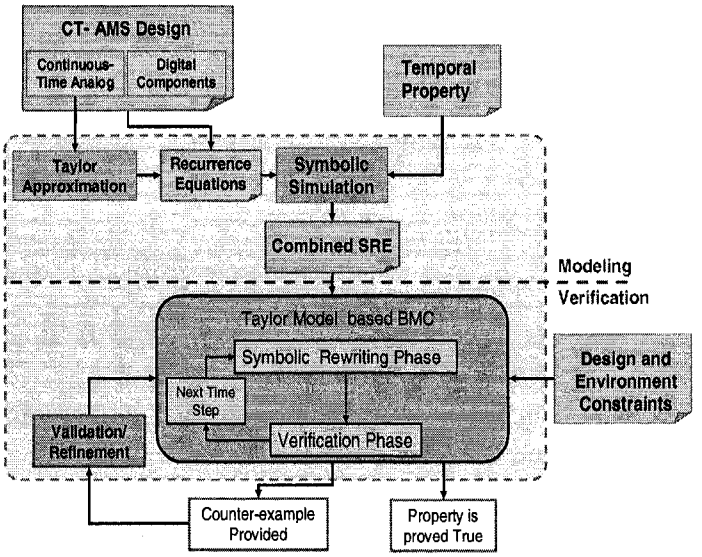


Figure 4.1: CT-AMS BMC Verification Methodology

Chapter 3, Section 3.1.4) for the current cycle, in the verification step, constraint solving approaches are applied to check for property satisfaction. In case the property could not be verified a counter-example is generated. A validation and refinement procedure is then applied to identify spurious counter-examples and discard them, while returning concrete ones.

The verification procedure terminates into one of the following cases:

- Complete verification:
  - Fixed point is reached and the timed property is proved True.
  - The property is false and a concrete counter-example is found.
- Bounded Verification:
  - The resource limits have been attained (memory or CPU) as the verification is growing exponentially with increasing number of reachability analysis steps.
  - The constraints extracted from the interval states are divergent with respect to some pre-specified criteria (e.g., width of computed interval states).

In the remaining of this chapter, we will also describe the main verification algorithms based on Taylor models reachability analysis. We will also provide a counter-example analysis and refinement used in order to enhance the bounded verification. We will end the chapter by applying the verification to different AMS examples, including oscillator circuits and a continuous-time $\Delta\Sigma$ modulator.

4.1 Reachability Analysis

In Chapter 3, we defined the reachable behavior of the AMS design as a set of traces representing the possible solution of a system of ODEs. We also proposed interval traces as an overapproximating abstraction of the reachable behavior. However, no specific way has been proposed to build such trace. In this chapter, we will explicitly tackle the issue of obtaining such traces. Several techniques have been proposed in literature to obtain abstract traces (See Chapter 2 for an overview of the methods used), mainly based on techniques inspired from computational geometry and optimization. In this chapter, we are taking a different approach based on symbolic simulation and rewriting techniques. Obtaining the set of traces and applying bounded reachability analysis is based on the concept of the semi-symbolic Taylor models. In the remaining, we will be giving an overview to the problem of reachability in general, followed by an exposition to Taylor models and interval arithmetics, before presenting our reachability analysis algorithm based on Taylor model symbolic simulation. We will also show how to enforce the sufficient approximation condition necessary to ensure the correctness of the results.

The set of reachable states from given states X_0 at time t can be defined as the set of all states visited by the trajectories starting from states X_0.

Definition 4.1.1. ***CT-AMS Model Reachable States.***
The set of reachable states *Reach* can then be defined as:

$$Reach \triangleq \{x' \in \mathcal{X} | \exists x \in Reach^0 \; such \; that \; \Phi_x(t) = x'\}$$

where $Reach^0 = X_0$. The set of reachable states in less than k steps $(0 < l < k)$, from a given set of X_0 of states, is denoted by $\mathcal{R}^{<k}(X_0)$, and is defined as:

$$\mathcal{R}^{<k}(X_0) \triangleq \bigcup_{l<k} \mathcal{R}^l(X_{l-1})$$

with $\mathcal{R}^l$ is the set of states reached during one step.

Obtaining the exact set of reachable states is not possible unless a closed form solution of the design equations is known. The goal is to construct an over-approximation that includes the original behavior. We propose a novel approach for reachability analysis using Taylor model arithmetics. As explained in Chapter 3, Taylor model arithmetics use interval methods allowing the computation of an over-approximation of the solution function at each time point. Furthermore, symbolic simplifications are applied at each step, thereby reducing the interval calculations and consequently delaying divergence problems that are typically associated with interval based techniques.

4.1.1 Taylor Model Based Reachability

We describe now the reachability analysis algorithm based on Taylor model arithmetics. The image computation is the set of states reachable during one execution step.

Definition 4.1.2. ***Taylor Model State Machine.***
A *Taylor Model State Machine* is a tuple $\mathcal{T}_I = (S_I, S_{I,0}, \rightarrow_{T_f})$, where S_I is the interval state space, $S_{I,0} \subseteq S_I$ is the set of initial interval states, $\rightarrow_{T_f} \subseteq S_I \times S_I$ is a relation defined using Taylor model forms T_f and capturing the abstract transition between interval states such that:

$$\{s \rightarrow_{T_f} s' | \exists a \in s, \exists b \in s' : b = f(a) \ \textit{and} \ f \in T_f\}$$

where $a, b \in \mathbb{R}^d$, $s, s' \in S_I$, $f = \{f_1, \ldots, f_d\}$, $T = \{T_{f_1}, \ldots, T_{f_d}\}$ with $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ is a continuous function, $i \in \{1, \ldots, d\}$ and $f_i \in T_{f_i}$, where T_{f_i} is the Taylor model of f_i.

Definition 4.1.3. ***1-Step Image Computation.***

The set of reachable states in 1-step from a given set of states $S_k \subseteq \mathbb{I}^d$, is denoted by $\mathcal{R}_1(S_k)$ and is defined as:

$$\mathcal{R}_1(S_k) \triangleq \{s' \in S_{k+1} | \exists s \in S_k : \overrightarrow{F}_1(s) = s'\}$$

where $S_{k+1} \subseteq \mathbb{I}^d$, $\overrightarrow{F} = (F_1, \ldots, F_d)$, with $F_i : \mathbb{I}^d \rightarrow \mathbb{I}$ is an interval evaluation of of the if-formula $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$, $i \in \{1, \ldots, d\}$.

Definition 4.1.4. ***k-Step Image Computation.***

The set of reachable states in less than k steps $(0 < l < k)$, from a given set of S_0 of states, is denoted by $\mathcal{R}^{<k}(S_0)$, and is defined as:

$$\mathcal{R}^{<k}(S_0) \triangleq \bigcup_{l<k} \mathcal{R}^l(S_{l-1})$$

The advantage of using Taylor model arithmetics over Interval arithmetics is based on the following points: first, Taylor model avoids or minimize common issues inherited in the interval arithmetics like the dependency problem and the wrapping effect. Second, Taylor model provides a non-convex enclosure of the concrete reachable states, hence tighter abstract reachable states leading to more precise verification results as demonstrated by Lemma 4.1.1 below. Another advantage lies in the generation and validation of counter-examples. The structure of the Taylor models allows an efficient way to analyze counter-examples as will be shown in more detail in Section 4.3.1.

Starting from the initial conditions, the reachable states of the system of recurrence equations are an overapproximation of the reachable states of the system of piecewise equations.

Statement. Given a set $X_0 \subseteq \mathbb{R}^d$ of initial states which is described as an interval of dimension d, a final time t_f and a corresponding CT-AMS Trace *Reach*, compute an interval AMS Trace $\widetilde{Reach} = abs(Reach)$, where $abs(.)$ is described as in Definition 3.2.10.

Lemma 4.1.1. A Taylor Model Transition System $\mathcal{T}_{T\mathcal{M}}$ is a refinement of Interval Transition System $\mathcal{T}_I$, such that $\mathcal{T}_I \succcurlyeq \mathcal{T}_{T\mathcal{M}} \succcurlyeq \mathcal{T}_{\mathcal{A}}$, where $\mathcal{T}_{\mathcal{A}}$ is the original CT-AMS Transition System.

The Taylor model based reachability analysis is illustrated with Algorithm 1. The function $\mathcal{TM}\_Reach(.)$ accepts as input the SREs representing the CT-AMS behavior, the maximum duration of the reachability T_f, the order O_t of the Taylor model approximation, the initial time step Δ_0 and the initial time T_0. If the reachability terminates successfully, then $\mathcal{TM}\_Reach(.)$ returns the set of reachable states $\mathcal{R}^f$, where f index denotes the analysis termination index, otherwise it returns the reachable states $\mathcal{R}^n$ up to time step $n < f$. There are two possible reasons for early termination of the algorithm; either an inclusion fixed point is reached, therefore no new states will be explored. The other reason if the precision of the approximation cannot capture accurately the complete behavior of the design equations. This is generally when the time step reaches a lower bound

The details of the algorithm are described as follows. At the beginning, the algorithm initializes the index n and the time step T_{n-1}. Initial conditions are provided as intervals written as a combination of two terms; a numerical term and symbolic term representing the variations. For example if $x[0] = [1,2]$, then this can be represented as $x[0] = 1.5 + a$, where $a = [-0.5, 0.5]$. In this way, symbolic terms can be propagated through the different cycles, without being evaluated, unless it is required[1]. This is more efficient than representing the initial condition with a single term with interval width, which is larger when evaluated. Additionally, the set of reachable states $\mathcal{R}^n$ are initialized, the time step Δ is set to the initial time step Δ_0 and the corresponding recurrence equations are generated from the ODEs system using the $SRE(.)$ function as described in Section 3.2 (Chapter 3).

[1]The choice of the evaluation of a symbolic term by its original interval value is done according the Taylor model rules R_{Tlr} described in Chapter 3, Section 3.1.4.

The reachability algorithm is applied for a maximum time T_f (Line 3) and if successful, returns the updated set of reachable states (Lines 9, 16 and 19). For each reachability step, we start by generating the Taylor model polynomial form with order O_t from the SRE equation (Line 4). Due to over-approximation nature of the method, imprecise results might be obtained, in this case a flag *Flag_Reachability-Imprecise* (Line 23) is set indicating a problem with the reachability and only reachable states up till the current cycle are returned. Otherwise, the reachability algorithm proceeds (Lines 5- 23). We check the accuracy of the reachable states using the *sufficien_approx*(.) function (Line 5) if accuracy is *bad* [2], we end the reachability as stated before, otherwise we continue the algorithm. We define the intermediate Taylor model forms; i.e., $\widetilde{\mathbf{x}}[n]$ where the time step is evaluated (Line 6) and $\widehat{\mathbf{x}}[n]$ which is the interval based evaluation of the Taylor model (Line 7). The evaluation is done by the function *eval*(.) which takes a Taylor model form and the parameters to evaluate. If an inclusion fixed point is reached (Lines 8 -10), the algorithm stops as all reachable states have been visited.

The next part of the algorithm (Lines 12 - 20) is concerned with checking for possible changes in the switching conditions using the function *Eval_Cond*(.). A trajectory of the CT-AMS design in the continuous state space can be though of as a sequence of continuous trajectories segments with discrete components describing the switching conditions defined using predicates. The valuation over interval domains of the predicates hence lead to a three valued logic; the image of *Eval_Cond*(.) is {**T**, **F**,**X**}. Therefore, starting from an initial state, there could me more than on trace as some switching conditions might not be evaluated to either true or false. If *Eval_Cond*(.) is evaluated to **F**, then the dynamics of the design are unchanged (Line 18), and the set of reachable states is updated (Line 19) before proceeding to the next time step. However, if *Eval_Cond*(.) is evaluated to **T** (Line 14), then a new initialization of the dynamics is needed (Line 15-17),

[2]We say the accuracy of the approximation is bad, if the minimum delta time step used is insufficient to capture the changes in the behavior, this is explained in more derails in Algorithm 2.

which is the states at the intersection of the last reachable states and the threshold condition [3]. When *Eval_Cond*(.) is evaluated to **X** (Line 12), a function *Switch_Check*(.) is called in order to enhance the precision of the reachability and remove spurious nondeterminism (Line 13). the function *Switch_Check*(.) is described in more detail in Algorithm 3.

Note. Concerning the termination of the algorithm, setting bounds on the maximum number of iterations ensures that the algorithm will eventually terminate in one of the possibilities described earlier. However, this is only guaranteed under the condition that each of the functions called by the algorithm (e.g., Suffic_Approx(.), Switch_Check(.)) will eventually terminate.

4.1.2 Sufficient Discretization Conditions

Time discretization is employed as a means to allow the formal verification of CT-AMS designs. Hence, the discretization must capture correctly the behavior of the CT-AMS design (See Chapter 3 for more details). In general, for the case where the time step τ is fixed, to ensure a precise coverage approximation of the reachable states, the assumption can be made that a switching condition is satisfied only at fixed instant defined in terms of τ. [4] In practice, for CT-AMS designs, a switching condition can be satisfied anywhere during the continuous trajectory. Consequently, the continuous evolution must be relaxed by allowing the time-step to change in the range [0, τ] to capture all the required behaviors in a more precise manner.

On the other hand, interval methods for solving the initial value problem (IVP) of ODEs provides a simple form for the error term of the discrete methods which can be bound as long as some enclosure of the actual solution function is provided. Moreover, the step size may be easily modified during the approximation process. One advantage of

[3] This is done using the interval-logical rules $R_{Int-Logic}$ described in Chapter 3, Section 3.1.3

[4] This constraints is similar to the constraints in the verification of DT-AMS which will be described in Chapter 6

Algorithm 1 Taylor Model Bounded Reachability: $\mathcal{TM}\_Reach(\mathbf{x}[n], T_f, O_t, \Delta_0, T_0)$

Require: $n = 1$
Require: $T_{n-1} = T_0$
Require: $\mathbf{x}[n-1] = j + \mathbf{a}$, with $j \in \mathbb{N}^d$, $\mathbf{a} \in \mathbb{I}^d$
Require: $\mathcal{R}^0 \leftarrow \vec{x}[n-1]$
Require: T_f and $\Delta \leftarrow \Delta_0$
Require: $\mathbf{x}[n] = SRE(\mathbf{x}(t))$

$\widetilde{\mathbf{x}}[n-1] = \mathbf{x}[n-1]$
$T_n = Inc\_Step(T_{n-1}, \Delta_0)$
while $T_n \leq T_f$ **do**
  $\mathbf{x}[n] = \mathcal{TM}_{o_t, \mathbf{x}[n]}(\widetilde{\mathbf{x}}[n-1])$
  if $Suffic\_Approx(\mathbf{x}[n], \mathbf{x}[n-1], \Delta_0)$ **is** *Good* **then**
    $\widetilde{\mathbf{x}}[n] = eval(\mathbf{x}[n], \{\Delta\})$
    $\widehat{\mathbf{x}}[n] = eval(\mathbf{x}[n], \{\mathbf{a}, \Delta\})$
    if $\widehat{\mathbf{x}}[n] \subseteq \mathcal{R}^{n-1}$ **then**
      $\mathcal{R}^n = \mathcal{R}^{n-1}$
      Return *Flag_Fix-Point-Reached = True*
    end if
    if $Eval\_Cond(\widehat{\mathbf{x}}[n], \widehat{\mathbf{x}}[n-1]) == \mathbf{X}$ **then**
      Call $Switch\_Check(\mathbf{x}[n], \mathbf{x}[n-1], \mathcal{R}^n)$
    else if $Eval\_Cond(\widehat{\mathbf{x}}[n], \widehat{\mathbf{x}}[n-1]) == \mathbf{T}$ **then**
      $\widehat{\mathbf{x}}[n] = \widehat{\mathbf{x}}[n] \cap \|Switch_i\|$
      $\mathcal{R}^n = Update\_Reach(\mathcal{R}^{n-1}, \widehat{\mathbf{x}}[n])$
      $\mathbf{x}[n] = j + \mathbf{a}'$
    else
      $\mathcal{R}^n = Update\_Reach(\mathcal{R}^{n-1}, \widehat{\mathbf{x}}[n])$
    end if
    $inc(n)$
    $T_n \leftarrow Inc\_Step(T_{n-1}, \Delta_0)$
  else
    Return *Flag_Reachability-Imprecise = True*
  end if
end while
Return *Flag_Reachability-Done = True*

interval based methods over conventional numerical methods is that a validation procedure for the existence of a unique solution is applied before finding the adequate enclosure of this solution between the two time steps. Usually the validation and enclosure of solutions of an ODE system between two discrete points t_i and t_{i+1} is based on the Banach fixed-point theorem [89] and the application of the Picard operator [89].

Moreover, we need to guarantee the sufficient discretization to ensure not only that the reachability guarantees covering all the reachable states, but also that it captures the main qualitative aspects of the trajectory. Enclosing the original trajectories using interval methods is sound (See Chapter 3, Section 3.1.3), but due to the associated over-approximation, the qualitative aspects of the behavior might be lost thus rendering verification of certain properties intractable. Accordingly, complementary methods are necessary in order to capture the desired qualitative properties.

An essential qualitative criterion is to guarantee that monotonicity is preserved during a time step τ. In order to check this condition, we use the *generalized mean value theorem*, which is an extension of the mean value theorem (MVT) for n-dimension that was proposed in [40]:

Theorem 4.1.1. ***Generalized Mean Value Theorem.*** Given $\mathbf{x}(t)$ that is continuous on a time interval $a \leq t \leq b$, and differentiable on $a < t < b$, assume that there exists a vector $\mathbf{V}$ orthogonal to $\mathbf{x}(a)$ and to $\mathbf{x}(b)$. Then $\exists t_c : a < t_c < b$ such that $\mathbf{V}$ is orthogonal to $\dot{\mathbf{x}}(t_c)$

For instance in the case of a 2-dimensional system, $\mathbf{x} = (x(t), y(t))$, the generalized MVT is reduced to the standard Cauchy MVT [39]:

$$\dot{x}(t_c)[y(b) - y(a)] = \dot{y}(t_c)[x(b) - x(a)]$$

For a 3-dimensional system, $\mathbf{x} = (x(t), y(t), z(t))$, we have [40]:

$$x(a)[y(b)\dot{z}(t_c) - z(b)\dot{y}(t_c)] + z(a)[x(b)\dot{y}(t_c) - y(b)\dot{x}(t_c)] = y(a)[x(b)\dot{z}(t_c) - z(b)\dot{x}(t_c)]$$

Practically, we use quantified constraint based methods [11] and symbolic algebraic techniques [83] in order to simplify (e.g., eliminate quantifiers) and decide the satisfiability

of formulas representing the mean value theorem. The procedure to check for sufficient discretization is described in Algorithm 2.

The function $Suffic\_Approx()$ is a recursive function that accepts as input the Taylor model forms $\widetilde{\mathbf{x}}[n]$ and $\mathbf{x}[n-1]$ with the last chosen time step Δ and returns one of the two possible values {Good, Bad} and when possible a time step that ensures capturing the qualitative behavior. The algorithm requires the index n of last reached state and $\varepsilon > 0$, the smallest allowed time step. In order to ensure the termination of the algorithm, we add a limit to the minimum possible value of $\Delta = \varepsilon$, beyond which the verification process is stopped. If monotonicity is preserved (Line 16), then we do not chose a smaller time step and the algorithm terminates. However, in case the monotonicity property is violated, we get τ' which violates the monotonicity criteria and refine the time step (Line 1-7 and 8-15). This is done in a recursive fashion until an adequate time step is chosen or the time step ε is reached. In such case, $Suffic\_Approx(.)$ will be evaluated to *Bad* and the verification stops as the accuracy might not lead to a precise result. This means that a sufficient approximation for the reachability cannot be found. The function $Sign(Slope(.))$ returns the sign of the vector field; whether it is increasing or decreasing on the boundaries of the time interval $[0, \tau']$.

We use $\mathcal{TM}_j(\mathbf{x}, \tau)$ to denote the Taylor polynomial of degree j relative to the solution $\mathbf{x}(t)$ centered in $\mathbf{x}(0)$ with a step size of τ. For instance, $\mathcal{TM}_1(\mathbf{x}(0), \tau)$ is the vector expression $\mathbf{x}(0) + f(\mathbf{x}(0))\tau + I$.

Note. The termination of this algorithm can be ensured if the recursion depth is not infinite. In this respect, we choose a lower bound for the time step as a main criteria to avoid such problem. Additionally, we assume the non existence of a Zeno behavior [5] when looking for an adequate time step.

[5] Informally, a Zeno behavior leads to an execution that takes an infinite number of discrete computations during a finite time interval [4].

Algorithm 2 Sufficient Approximation: $Suffic\_Approx(\mathbf{x}[n], \mathbf{x}[n-1], \Delta)$

Require: $n \in \mathbb{N}$
Require: $\varepsilon \in \mathbb{R}$
Require: $\Delta = \Delta_0$
Require: $\mathbf{x}[n] = \mathcal{TM}_{o_t, \mathbf{x}[n]}(\widetilde{\mathbf{x}}[n-1])$
Require: $\widetilde{\mathbf{x}}[n] = eval(\mathbf{x}[n], \{\Delta\})$
Require: $\widehat{\mathbf{x}}[n] = eval(\mathbf{x}[n], \{\mathbf{a}, \Delta\})$
Require: $\widehat{\mathbf{x}}[n-1] = eval(\mathbf{x}[n-1], \{\mathbf{a}, \Delta\})$
1: **if** $[\exists \tau'.\check{\mathbf{x}}[n] = eval(\mathbf{x}[n], \{\mathbf{a}, \tau'\}) \wedge 0 \leq \tau' \leq \Delta \wedge Sign(Slope(\check{\mathbf{x}}[n])) \neq Sign(Slope(\widehat{\mathbf{x}}[n-1]))] ==$ True **then**
2: **if** $\tau' \geq \varepsilon$ **then**
3: $\Delta = \tau'$
4: **Call** $Suffic\_Approx(\mathbf{x}[n], \mathbf{x}[n-1], \Delta)$
5: **else**
6: **Return** Bad
7: **end if**
8: **else if** $[\exists \tau'.\check{\mathbf{x}}[n] = eval(\mathbf{x}[n], \{\mathbf{a}, \tau'\}) \wedge 0 \leq \tau' \leq \tau \wedge Sign(Slope(\check{\mathbf{x}}[n])) == 0] ==$ True **then**
9: **if** $\tau' \geq \varepsilon$ **then**
10: $\Delta = \tau'$
11: **Call** $Suffic\_Approx(\mathbf{x}[n], \mathbf{x}[n-1], \Delta)$
12: **else**
13: **Return** Bad
14: **end if**
15: **end if**
16: **Return** Good

4.1.3 Checking Switching Condition

Due to the overapproximation nature of Taylor model evaluation, the evaluation of switching conditions in the AMS model might not be decided in a precise way. More specifically, there could be more than one successor for a given state if the decision on which switching condition holds at a given instant cannot be uniquely identified. In order to guarantee correct verification results, all possible reachability paths must be explored. On the other hand, from a correct design point of view, nondeterminism cannot exist in AMS models. In other words, we have a valid assumption that at any instant, in reality, only one switching condition (or its compliment condition) can be satisfied.

In order to check whether a switching condition occurs between two time steps, we apply the *intermediate value theorem*. In the context of abstraction, a transition between two abstract states exists if a predicate valuation changes during the execution over an interval domain. We check for such conditional abstract transitions between two states by means of the intermediate value theorem (IVT) [39] as follows:

Theorem 4.1.2. ***Intermediate Value Theorem.*** Given a predicate λ, two states S_1 and S_2 = differing only on the valuation of λ and a time step interval solution $I : \{a_1 \leq x \leq a_2\}$, there is a transition between S_1 and S_2 if $S_1 \models [\![\lambda]\!]_{a_1}$ (i.e., $\lambda(a_1) \in abs^{-1}(S_1)$), $S_2 \models [\![\lambda]\!]_{a_2}$ (i.e., $\lambda(a_2) \in abs^{-1}(S_2)$) and $[\![\lambda]\!]_{a_1} \neq [\![\lambda]\!]_{a_2} \neq 0$, $\exists x$ such that $[\![\lambda]\!]_x = 0$, with the interpretation function $[\![.]\!] : \mathbb{R}^d \rightarrow \{+, -, 0\}$

To check for the above condition, we use interval analysis to guarantee that the solution is reliable; the real solutions are enclosed by the computed intervals. Such guarantee is derived from the fundamental theorem of interval analysis [85].

The procedure for checking the switching conditions evaluation is described in Algorithm 3. The main function *Switch_Check*(.) is called whenever *Eval_Cond*(.) evaluates to **X** in Algorithm 1, in an effort to obtain more precise results concerning the evaluation of the switching conditions. The function accepts as input the Taylor model forms $\mathbf{x}[n]$

and $\mathbf{x}[n-1]$ with the updated set of reachable states $\mathcal{R}^n$ and returns one of the two possible values {Switching_Occurs, No_Switching} or call the function $Refine\_Switch(.)$ for more precise analysis. The function $Switch\_Check(.)$ requires the initial time step Δ_0, the current time T_n as well as the Taylor models evaluations $\widetilde{\mathbf{x}}[n]$ and $\widehat{\mathbf{x}}[n-1]$

Suppose that there exists a switching condition $Switch_n$ at cycle n, which is evaluated to **X**, then we make a temporary assumption that switching did not occur and we check for the reachable states at the next time step $n+1$ using the $\mathcal{TM}\_Reach\_Step(.)$ function(Line 1), which is a simplified version of the function $\mathcal{TM}\_Reach(.)$, with the assumptions that $Suffic\_Approx(.) == Good$ and $Switch_n$ is set to **F**. We have the options shown below, where $\| Switch_n \|$ denotes the set of all states that evaluate $Switch_n$ to **T**.

- if $Switch_{n+1} = \mathbf{T}$ (Lines 2-5), then indeed the switching occured at the previous time t_n. The reachable states are updated (Line 3) and an initialization is set for the newly selected dynamics (Line 4).

- if $Switch_{n+1} = \mathbf{F}$ (Lines 6-7), then indeed switching did not occur. This follows from the interval evaluation property that ensures that the evaluation at step $n+1$ encloses all previous states up to time after t_n.

- if $Switch_{i+1} = \mathbf{X}$ (Lines 9), then we allow checking with robustness, whether or not the switching occurs by calling the function Refine_Switch. Informally speaking, given a robustness measure ε, check the distance between the switching condition and the current state. If its is less than ε, then we say that there is fragile switching $\| Switch_n^{\varepsilon} \| \cap X_{n+1} \neq \emptyset$

Note. The algorithm will eventually terminate in one of the possibilities described earlier. However, this is only guaranteed under the condition that each of the functions called by the algorithm (e.g., Eval_Cond(.), Refine_Switch(.)) will eventually terminate.

Algorithm 3 Checking Switching Condition: $Switch\_Check(\mathbf{x}[n],\mathbf{x}[n-1]),\mathcal{R}^n$

Require: $\Delta \leftarrow \Delta_0$
Require: $T_f = T_n + \Delta$
Require: $\mathbf{x}[n] = \mathcal{TM}_{o_t}(\mathbf{x}[n],\widetilde{\mathbf{x}}[n-1])$
Require: $\widetilde{\mathbf{x}}[n] = eval(\mathbf{x}[n],\{\Delta\})$
Require: $\widehat{\mathbf{x}}[n] = eval(\mathbf{x}[n],\{\mathbf{a},\Delta\})$
Require: $\widehat{\mathbf{x}}[n-1] = eval(\mathbf{x}[n-1],\{\mathbf{a},\Delta\})$
1: $\widehat{\mathbf{x}}[n+1] = \mathcal{TM}\_Reach\_Step(\widehat{\mathbf{x}}[n],T_f,O_t,T_n)$
2: **if** $Eval\_Cond(\widehat{\mathbf{x}}[n+1]) ==$ **T then**
3: $\quad\mathcal{R}^n = Update\_Reach(\mathcal{R}^{n-1},\widehat{\mathbf{x}}[n])$
4: $\quad\widehat{\mathbf{x}}[n] = \widehat{\mathbf{x}}[n] \cap \|Switch_n\| = j + \mathbf{a}$
5: $\quad$**Return** Switching_ Occurs
6: **else if** $Eval\_Cond(\widehat{\mathbf{x}}[n+1]) ==$ **F then**
7: $\quad$**Return** No_ Switching
8: **else**
9: $\quad$**Call** $Refine\_Switch(\mathbf{x}[n],\Delta,\| Switch_n^{\varepsilon} \|)$
10: **end if**

Example 4.1.1. Consider the circuit in Figure 3.4, with the voltages across the capacitors described using ODEs as follows:

$$\begin{cases} Mode1: & \dot{v_{c1}} = v_{c2} \text{ and } \dot{v_{c2}} = -v_{c1} + v_{c1}^3 \\ Mode2: & \dot{v_{c1}} = v_{c1}^2 + 2v_{c1}v_{c2} + 3v_{c2}^3 \text{ and } \dot{v_{c2}} = 4v_{c1}v_{c2} + 2v_{c2}^2 \end{cases}$$

and the switching conditions as

$$Cond_1 = Cond_2 = -0.5v_{c1}(n) + v_{c2}(n) \leq 4$$

Suppose that the circuits starts at Mode 2, with initial conditions $v_{c1} = -10 + a$, where $a = [-0.3, 0.3]$ and $v_{c2} = 5 + b$, where $b = [-0.3, 0.3]$. The switching condition threshold is satisfied at voltage values $v_{c1} = -6.6 + a'$ with $a' = [-0.16361, 0.125]$ and $v_{c2} = 0.5 + b'$ with $b' = [0.118195, 0.2625]$, which are in turn the initial states for the dynamics at mode 1. The trajectory of the circuit with the switching condition are illustrated in Figure 4.2.

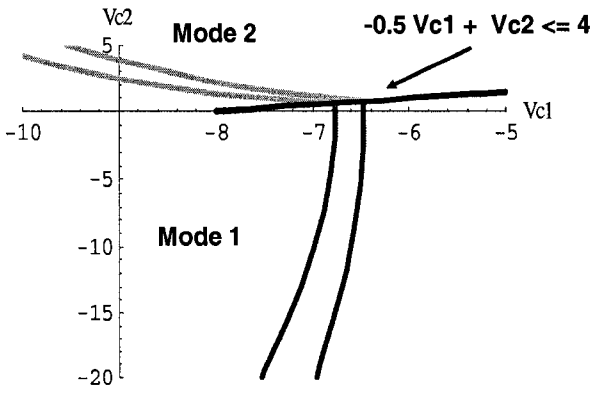


Figure 4.2: Switching Condition Satisfaction

4.2 Bounded Model Checking

Given an AMS system, an initial set X_0, and a bad set B_X, the verification problem is to determine if there is an execution of AMS, starting in X_0 and ending in B_X. If the system is safe (i.e., B_X is unreachable), a complete verification strategy should be able to demonstrate this. In such a case, the bounded model checking (BMC) technique is often used.

The general BMC problem can be encoded as follows [14]:

$$BMC(P,k) \triangleq I(s_0) \wedge \bigwedge_{i=0}^{k-1} \mathcal{T}(s_i \rightarrow s_{i+1}) \rightarrow P(s_k)$$

where $I(s_0)$ is the initial valuation for the state variables, s_i is the state variable valuation at step i, $\mathcal{T}$ defines the transition between two states and $P(s_k)$ is the property at step k. In practice, the inverse of the property ($\neg P$) under verification is used in the BMC algorithm. When a satisfying valuation is returned by the solver, it is interpreted as a counter-example of length k and the property P is proved unsatisfied ($\neg P$ is satisfied). However, if the problem is determined to be unsatisfiable, the solver produces a proof (of unsatisfiability) of the fact that there are no counter-examples of length k. For instance, the BMC problem for safety properties $P(k) \triangleq \mathbf{G}p(k)$ can be encoded as follows [14]:

$$BMC(P,k) \triangleq I(s_0) \wedge \bigwedge_{i=0}^{k-1} \mathcal{T}(s_i \rightarrow s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

while the BMC problem for liveness properties $P(k) \triangleq \mathbf{F}p(k)$ can be encoded as follows [14]:

$$BMC(P,k) \triangleq I(s_0) \wedge \bigwedge_{i=0}^{k-1} \mathcal{T}(s_i \rightarrow s_{i+1}) \wedge \bigwedge_{i=0}^{k} \neg p(s_i)$$

Bounded model checking is then defined as follows:

Definition 4.2.1. ***Bounded Model Checking.***

Given a natural number $k \geq 0$, a state transition machine $(S_I, S_{I,0}, \rightarrow_{T_f})$ as defined above, and a property P, we say that property P is verified for k steps if:

$$\forall s \in \mathcal{R}^k(S_0) : s \models P$$

where S_0 is the set of initial states.

Generally, a symbolic algorithm that computes the set of reachable states from X_0 by iteratively computing the set of states reachable in discrete (or continuous steps) cannot be guaranteed to terminate after a bounded number of iterations. In addition, unlike BMC for discrete systems, it is not possible to calculate an upper bound on the number of future/past iterations for which the formula should be checked in order to guarantee that the property holds. However, incorporating time constraints into the temporal logic property can overcome such problems, i.e., we ask if a property holds until we are no longer in the time-frame of interest, as opposed to asking if the property holds forever. In the bounded version of the model-checking task, we are only interested in the system evolution over a bounded time horizon or a bounded number of steps. This is achieved using timed temporal logic MITL as the property languages.

4.2.1 Interval Based Bounded Model Checking

In this section, we present a BMC algorithm for AMS designs. We explore a solution relying on symbolic and interval computational methods. Our BMC approach is based on modeling the transition function as SREs over the Taylor model forms. We proceed on the SREs traces using a time step $\hbar$ which implies that our answer is relative to a limited time interval. For recurrence equations, we have $\hbar = 1$. For differential equations, we approximate them using Taylor model with $\hbar \in \mathbb{R}^+$, ensuring the accumulated error due to $\hbar$-approximation is confined in the Interval part of the Taylor model. We consider properties specified in a MITL like language.

According to the standard semantics for temporal logic, the satisfaction of a formula with unbounded modalities can be hard to determine. In fact, given an atomic proposition p only the satisfaction of $\mathbf{F}p$ or violation of $\mathbf{G}p$ can be detected in finite time. By using bounded modalities we avoid the problems arising from the ambiguity of $\models$. We restrict ourselves to traces which are sufficiently long. The necessary length associated with a formula ϕ, denoted by $||\phi||$, is inductively defined on the structure of the formula.

- $||p|| = 0$
- $||\neg\phi|| = ||\phi||$
- $||\phi_1 \vee \phi_2|| = max(||\phi_1||, ||\phi_2||)$
- $||\mathbf{G}_{[a,b]}\phi_1|| = ||\phi_1|| + b$
- $||\mathbf{F}_{[a,b]}\phi_1|| = ||\phi_1|| + b$

We now have that $\sigma \models \phi$ is well defined whenever $|\sigma| > ||\phi||$

Example 4.2.1. The interpretation of the MITL properties in a bounded model checking context can be made clear with the examples below.

Case 1: τ is fixed

- $\mathbf{G}_{\leq 100}\mathbf{F}_{\leq 5}p := \bigwedge_{n_1=0}^{m_1} \bigvee_{n_2=n_1}^{m_2+n_1} p \wedge (n_1 \times \tau \leq 100) \wedge (n_2 \times \tau \leq 5)$
- $\mathbf{F}_{\leq 100}\mathbf{G}_{\leq 5}p := \bigvee_{n_1=0}^{m_1} \bigwedge_{n_2=n_1}^{m_2+n_1} p \wedge (n_1 \times \tau \leq 100) \wedge (n_2 \times \tau \leq 5)$
- $\mathbf{G}_{\leq 100}(q \rightarrow \mathbf{F}_{\leq 5}p) := \bigwedge_{n_1=0}^{m_1} (\neg q \vee \mathbf{F}_{\leq 5}p) = \bigwedge_{n_1=0}^{m_1} (\neg q \vee \bigvee_{n_2=n_1}^{m_2+n_1} p) \wedge (n_1 \times \tau \leq 100) \wedge (n_2 \times \tau \leq 5)$

Case 2: τ is Variable

- $\mathbf{G}_{\leq 100}\mathbf{F}_{\leq 5}p := \bigwedge_{n_1=0}^{m_1} \bigvee_{n_2=n_1}^{m_2+n_1} p \wedge (\sum_0^{n_1} \tau_{n_1} \leq 100) \wedge (\sum_0^{n_2} \tau_{n_2} \leq 5)$
- $\mathbf{F}_{\leq 100}\mathbf{G}_{\leq 5}p := \bigvee_{n_1=0}^{m_1} \bigwedge_{n_2=n_1}^{m_2+n_1} p \wedge (\sum_0^{n_1} \tau_{n_1} \leq 100) \wedge (\sum_0^{n_2} \tau_{n_2} \leq 5)$
- $\mathbf{G}_{\leq 100}(q \rightarrow \mathbf{F}_{\leq 5}p) := \bigwedge_{n_1=0}^{m_1} (\neg q \vee \mathbf{F}_{\leq 5}p) = \bigwedge_{n_1=0}^{m_1} (\neg q \vee \bigvee_{n_2=n_1}^{m_2+n_1} p) \wedge (\sum_0^{n_1} \tau_{n_1} \leq 100) \wedge (\sum_0^{n_2} \tau_{n_2} \leq 5)$

As $n_i \in \mathbb{N}$, $\tau \in \mathbb{R}^+$ and the clock constraint is in $\mathbb{N}$, then in the general case, we can only have n_i and n_j such that $j = i+1$ and $(n_i \times \tau < C)$ and $(n_j \times \tau > C)$. We need to add the notion of time tolerance, where we check for properties with clocks $C+\varepsilon$, where $\varepsilon < \tau$ and $C+\varepsilon < n_j \times \tau$. It is worth noting that $\mathbf{Q}_{\leq T_f}$ is equivalent to $\mathbf{Q}_{[0,T_f]}$, where $\mathbf{Q}$ is a quantifier $\mathbf{F}$ or $\mathbf{G}$ and T_f is the maximum time length associated with the temporal quantifier.

4.2.2 BMC Algorithms

The bounded timed safety verification is illustrated with Algorithm 4. The function *G_Verify*(.) accepts as input the SREs representing the CT-AMS behavior, the order O_t of the Taylor model approximation, the initial time step Δ_0 and the property predicate p. The verification terminates successfully, if the time steps chosen captures the necessary behavior of the design. This is ensured using the function *Suffic_Approx*(.) (Line 4). In this case, either the property is verified to True (Lines 5 - 8), otherwise an abstract counter-example is generated (Lines 9 - 11) demonstrating the violation of the property. The function *Generate_CE*(.) (Line 11) is used to generate and validate the counter-example. In

case the function $Suffic\_Approx(.)$ cannot capture the behavior correctly, the verification stops in a failed state (Line 21).

The details of the algorithm are described as follows. The algorithm starts by re-setting the index n and the time step T_{n-1}. Initial conditions described as intervals are written as a combination of two terms; a numerical term and symbolic term representing the variations. The next step is the generation of the corresponding recurrence equations from the ODEs system using the $SRE(.)$ function and the time step Δ is set to the initial time step Δ_0. The maximum time length of the verification is measured according to the rules in Section 4.2.2. The loop (Lines 4 -13) describes the verification procedure for a period of time equal to the length of the property under verification.

The function Prop_Check is described as follows: Given the Taylor model forms representing the transition function and the property $\neg Prop()$, apply symbolic algebraic techniques [83] to check for satisfiability. The safety verification at a given step n can be defined with the following formula:

$$Prop\_Check \triangleq (\mathbf{x}[n] = T_{o_t,x[n]}(\tilde{\mathbf{x}}[n-1])) \wedge \neg Prop(\mathbf{x}[n]) \wedge \mathbf{x}[n-i] \in \mathbb{I}^d$$

Note. The algorithm will eventually terminate in one of the possibilities described earlier. However, this is only guaranteed under the condition that each of the functions called by the algorithm (e.g., Suffic_Approx(.), Prop_Check(.) and $\mathcal{TM}$_Reach(.)) will eventually terminate.

The bounded timed liveness verification for checking $\mathbf{F}_{<T_f} p$ properties is illustrated with Algorithm 5. The function $F\_Verify(.)$ accepts as input the SREs representing the CT-AMS behavior, the order O_t of the Taylor model approximation, the initial time step Δ_0 and the property predicate p. The loop (Lines 4 - 13) describes the verification procedure for a period of time equal to the length of the property under verification. The verification terminates successfully, if the time steps chosen captures the necessary behavior of the design. This is ensured using the function $Suffic\_Approx(.)$ (Line 4). In this case, either the property is verified to True (Lines 10 -11), or is verified to false at the current verification step (Lines 5 - 8) and the time step is incremented.

Algorithm 4 Bounded Timed Safety Verification $\mathbf{G}_{<T_f}p$: $G\_Verify(p, \mathbf{x}[n], O_t, \Delta_0, T_0)$

Require: $n = 1$
Require: $T_{n-1} = T_0$
Require: $\mathbf{x}[n-1] = j + \mathbf{a}$, with $j \in \mathbb{N}^d$, $\mathbf{a} \in \mathbb{I}^d$
Require: $\mathcal{R}^{n-1} \leftarrow \mathbf{x}[n-1]$
Require: $\mathbf{x}[n] = SRE(\mathbf{x}(t))$
Require: $\Delta \leftarrow \Delta_0$
Require: $T_f = Length(\mathbf{G}_{<T_f}p)$
Require: $G\_Verify\_flg == 1$
1: $\widetilde{\mathbf{x}}[n-1] = \mathbf{x}[n-1]$
2: $\mathbf{x}[n] = \mathcal{TM}_{o_t,\mathbf{x}[n]}(\widetilde{\mathbf{x}}[n-1])$
3: $Prop[n] = Symbolic\_Comp(\{p, \mathbf{x}[n]\})$
4: **while** $T_n \leq T_f$ and *Flag_Fix-Point-Reached* == *False* and $Suffic\_Approx(\mathbf{x}[n], \widehat{\mathbf{x}}[n-1])$ **is** *Good* **do**
5: **if** $Prop\_Check(Prop[n], \mathbf{x}[n], \mathcal{R}^{n-1}) == True$ **then**
6: $\mathcal{R}^n = \mathcal{TM}\_Reach(\mathbf{x}[n], T_{n-1} + \Delta, O_t, \Delta, T_{n-1})$
7: $inc(n)$
8: $t_n = Inc\_Step(t_{n-1}, \Delta)$
9: **else**
10: $G\_Verify\_flg = 0$
11: **Call** $Generate\_CE(x[n])$
12: **end if**
13: **end while**
14: **if** *Flag_Reachability-Imprecise* == *False* **then**
15: **if** $G\_Verify\_flg == 1$ **then**
16: **return** Property_is_True
17: **else**
18: **return** Verification_Failed
19: **end if**
20: **else**
21: **return** Verification_Failed
22: **end if**

If the maximum time step is reached or an inclusion fixpoint occurs having reached no state satisfying the property, then an abstract counter-example is generated (Lines 15 - 16) demonstrating the violation of the property. The function *Generate_CE*(.) (Line 16) is used for the generation and validation of the counter-example. In case the function *Suffic_Approx*(.) cannot capture the behavior correctly, the verification stops in a failed state (Line 23). Other details concerning the algorithms are the following. The algorithm starts by resetting the index n and the time step T_{n-1}. Initial conditions described as intervals are written as a combination of a numerical and symbolic terms. The time step Δ is set to the initial time step Δ_0. The maximum time length of the verification is measured according to the rules in Section .

Note. Similar to Algorithm 4, the liveness algorithm will eventually terminate in one of the possibilities described earlier. However, this is only guaranteed under the condition that each of the functions called by the algorithm (e.g., Suffic_Approx(.), Prop_Check(.) and $\mathcal{TM}$_Reach(.)) will eventually terminate.

The Algorithms 4 and 5 define the procedures for checking basic properties of CT-AMS designs. However, the verification approach we propose supports properties that can be written using the MITL subset defined in Section 3.3 (Chapter 3). For instance, general time bounded safety property can be described using the Algorithm below.

In Algorithm 6. The function *G_Verify_*ϕ(.) accepts as input the SREs representing the CT-AMS behavior, the order O_t of the Taylor model approximation, the initial time step Δ_0 and the property ϕ. Similar to Algorithm 4, the verification terminates successfully, if the time steps chosen captures the necessary behavior of the design. This is ensured using the function *Suffic_Approx*(.) (Line 3). In this case, either the property is verified to True using the function ϕ_*Verify*(.)(Lines 4 - 7), otherwise an abstract counter-example is generated (Lines 9 - 10) demonstrating the violation of the property. The function *Generate_CE_*ϕ(.) (Line 10) is used for the generation and validation of the counter-example. In case the function *Suffic_Approx*(.) cannot capture the behavior correctly, the verification stops in a failed state (Line 17).

Algorithm 5 Timed Liveness Verification $\mathbf{F}_{<T_f}p$: $F\_Verify(p, \mathbf{x}[n], O_t, \Delta_0, T_0)$

Require: $n = 0$
Require: $T_{n-1} = T_0$
Require: $\mathbf{x}[0] = j + \mathbf{a}$, with $j \in \mathbb{N}^d$, $\mathbf{a} \in \mathbb{I}^d$
Require: $\mathcal{R}^{n-1} \leftarrow \mathbf{x}[n-1]$
Require: $\mathbf{x}[n] = SRE(\mathbf{x}(t))$
Require: $\Delta \leftarrow \Delta_0$
Require: $F\_Verify\_flg = 0$
Require: $T_f = Length(\mathbf{F}_{<T_f}p)$
1: $\widetilde{\mathbf{x}}[n-1] = \mathbf{x}[n-1]$
2: $\mathbf{x}[n] = \mathcal{TM}_{o_t, \mathbf{x}[n]}(\widetilde{\mathbf{x}}[n-1])$
3: $Prop[n] = Symbolic\_Comp(\{p, \mathbf{x}[n]\})$
4: **while** $T_n \leq T_f$ and *Flag_Fix-Point-Reached* == *False* and $Suffic\_Approx(\mathbf{x}[n], \widehat{\mathbf{x}}[n-1])$ **is** *Good* **do**
5: **if** $Prop\_Check(Prop[n], \mathbf{x}[n], \mathcal{R}^{n-1}) == False$ **then**
6: $\mathcal{R}^n = \mathcal{TM}\_Reach(\mathbf{x}[n], T_{n-1} + \Delta, O_t, \Delta, T_{n-1})$
7: $inc(n)$
8: $t_n = Inc\_Step(t_{n-1}, \Delta_0)$
9: **else**
10: $F\_Verify\_flg = 1$
11: **return** Property_is_True
12: **end if**
13: **end while**
14: **if** *Flag_Reachability-Imprecise* == *False* **then**
15: **if** (*Flag_Fix-Point-Reached* == *False* or $T_n > T_f$) & $F\_Verify\_flg = 0$ **then**
16: **Call** $Generate\_CE(x[n])$
17: **else**
18: **if** $F\_Verify\_flg == 1$ **then**
19: **return** Property_is_True
20: **end if**
21: **end if**
22: **else**
23: **return** Verification_Failed
24: **end if**

The functions ϕ_*Verify*(.) and *Generate_CE_*ϕ(.) are functions that are chosen based on the property ϕ. For example, if the main property to verify is $\mathbf{G}p$, then ϕ refers to p and ϕ_*Verify*(.) corresponds to *G_Verify*(.), while *Generate_CE_*ϕ(.) corresponds to *Generate_CE*(.) which be described in the next section.

Algorithm 6 Bounded Timed Safety Verification $\mathbf{G}_{<T_f}\phi$: *G_Verify_*$\phi(\phi, \mathbf{x}[n], O_t, \Delta_0, T_0)$

Require: $n = 1$
Require: $T_{n-1} = T_0$
Require: $\mathbf{x}[n-1] = j + \mathbf{a}$, with $j \in \mathbb{N}^d$, $\mathbf{a} \in \mathbb{I}^d$
Require: $\mathcal{R}^{n-1} \leftarrow \mathbf{x}[n-1]$
Require: $\mathbf{x}[n] = SRE(\mathbf{x}(t))$
Require: $\Delta \leftarrow \Delta_0$
Require: $T_f = Length(\mathbf{G}_{<T_f}\phi)$
Require: $G\_Verify\_flg\_\phi = 1$
1: $\widetilde{\mathbf{x}}[n-1] = \mathbf{x}[n-1]$
2: $\mathbf{x}[n] = \mathcal{TM}_{o_t,\mathbf{x}[n]}(\widetilde{\mathbf{x}}[n-1])$
3: **while** $T_n \leq T_f$ and *Flag_Fix-Point-Reached* == *False* and $Suffic\_Approx(\mathbf{x}[n], \widehat{\mathbf{x}}[n-1])$ **is** *Good* **do**
4: **if** $\phi\_Verify(\mathbf{x}(n), O_t, \Delta, T_{n-1}) == True$ **then**
5: $\mathcal{R}^n = \mathcal{TM}\_Reach(\mathbf{x}(n), T_{n-1} + \Delta, O_t, \Delta, T_{n-1})$
6: $inc(n)$
7: $t_n = Inc\_Step(t_{n-1}, \Delta_0)$
8: **else**
9: $G\_Verify\_flg\_\phi == 0$
10: $Generate\_CE\_\phi(x[n])$
11: **end if**
12: **end while**
13: **if** *Flag_Reachability-Imprecise* == *False* **then**
14: **if** $G\_Verify\_flg\_\phi == 1$ **then**
15: **return** Property is True
16: **else**
17: **return** Verification Failed
18: **end if**
19: **end if**

Note. Similar to Algorithm 4, the general safety algorithm will terminate in one of the above mentioned possibilities under the condition that the functions called by the algorithm (e.g., Suffic_Approx(.), ϕ_Verify(.)) will eventually terminate.

Example 4.2.2. Oscillators play a critical role in communication systems, providing the periodic signals needed for the timing of digital circuits and for frequency translation. While an oscillator can mean anything that exhibits periodically time-varying characteristics, we are concerned with the type that provides an electrical signal (voltage or current) at a specific frequency when supplied only with DC power. An electrical oscillator generates a periodically time-varying signal when only supplied with DC power

For instance, consider the circuit in Example 4.1.1, with one of the dynamics is described by $\dot{v_{c1}} = v_{c2}$ and $\dot{v_{c2}} = -v_{c1} + v_{c1}^3$. The oscillation property can be formally described as:

$$Prop_1 : \mathbf{G}_{[0,7e^{-3}]}(\mathbf{F}_{[0,2e^{-3}]}p_2) \wedge \mathbf{G}_{[0,7e^{-3}]}(\mathbf{F}_{[0,2e^{-3}]}p_1)$$

where $p_1 = \neg p_2 := V_{c1} < V_{c2}$.

Applying the Algorithm 1 for building the Taylor models based reachable states, we can observe the oscillation behavior as illustrated in Figure 4.3. Where the reachable states are bounded by the corresponding Taylor model polynomials.

In order to check the satisfaction of the oscillation property, we apply the Algorithm 6.

We also checked several safety properties, e.g.,

$$Prop_2 : \mathbf{G}(-0.5 < V_{c1} < 0.5) \wedge (-0.5 < V_{c2} < 0.5)$$

and

$$Prop_3 : \mathbf{G}(-1 < V_{c2} < 1)$$

which are verified by applying Algorithm 4.

For the illustration purposes, we provided two different sets of initial states $x[0]$ and $y[0]$ as well as a fixed step size h as shown below:

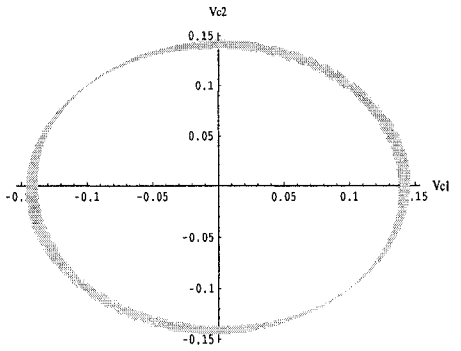


Figure 4.3: Oscillation Behavior for Circuit in Example 3.4 (Chapter 3)

$$Parameters_1 \rightarrow \begin{cases} a \rightarrow [-0.03, 0.03] & b \rightarrow [-0.03, 0.03] \\ h \rightarrow 0.01 & \\ x[0] = 0.3 + a & y[0] = -0.3 + b \end{cases}$$

$$Parameters_2 \rightarrow \begin{cases} a \rightarrow [-0.03, 0.03] & b \rightarrow [-0.03, 0.03] \\ h \rightarrow 0.01 & \\ x[0] = 1 + a & y[0] = 0.2 + b \end{cases}$$

The verification algorithms we implemented in Mathematica and applied on the design. The verification results for the two possible switching cases of this circuit (we refer to these as circuit 1 and circuit 2) are shown in Table 4.1. For the first set of initial conditions shown above, we find that the circuit is behaving in accordance with the properties, hence the properties are satisfied. For the second set of initial conditions, the safety properties $Prop_2$ and $Prop_3$ are violated while divergence prevents us from checking whether the circuits are oscillating or not [6].

When a property is not verified, a counter-example is generated to help identify the reasons for the property violation. Due to the over-approximation of the BMC algorithms, the generated counter-example is an abstract one. Therefore, the counter-example must

[6]The experiments were performed on Intel Core2 1900 MHz processor with 2GB of RAM

Table 4.1: Oscillator Verification Results

Circuit & Properties	**BMC Verification for $k = 0$ to N_{max} Steps**	**CPU & Memory Used**
Circuit 1 (Parameters 1)	$N_{max} = 700$	107.39 sec
Oscillation Property	*Proved True*	7.93 MB
$Prop_2$	*Proved True*	
$Prop_3$	*Proved True*	
Circuit 1 (Parameters 2)	$N_{max} = 700$	108.41 sec
Oscillation Property	Not Verified (Divergence)	7.14 MB
$Prop_2$	*Proved False* at $k = 18$	
$Prop_3$	*Proved False* at $k = 18$	
Circuit 2 (Parameters 1)	$N_{max} = 1200$	583.75 sec
Oscillation Property	*Proved True*	51.15 MB
$Prop_2$	*Proved True*	
$Prop_3$	*Proved True*	
Circuit 2 (Parameters 2)	$N_{max} = 1200$	584.05 sec
Oscillation Property	Not Verified (Divergence)	50.60 MB
$Prop_2$	*Proved False* at $k = 4$	
$Prop_3$	*Proved False* at $k = 9$	

be validated and when possible, in case it is a spurious one, use the information from it in order refine the abstract reachable states. In this respect, we extend the BMC algorithm with a counter-example analysis engine as shown in Figure 4.1.

4.3 Finding Counter-example

This section present the counter-example analysis for safety properties. In the verification approach, safety of an over-approximation implies safety of the actual system. On the other hand, if the over-approximation is unsafe, it is not necessarily the case that the design is faulty; in this case, the generated counter-examples might be spurious. A counter-example is defined as follows:

Definition 4.3.1. ***Counter-example.***

A trace $\Omega = (\sigma, \tau, \lambda)$ of the AMS system is called an abstract counter-example with respect

to the property $\mathbf{G}p$, if $\sigma_n \cap \Upsilon(p) \neq \emptyset$, where Υ is the concretization function abs^{-1}. Ω is a corresponding abstract counter-example of a concrete one if $\exists \rho \in \Upsilon(\sigma)$ and $\rho = \Upsilon(\sigma_0), \Upsilon(\sigma)_1, \ldots, \Upsilon(\sigma)_n$ is a real trajectory of the system and $\rho_n \cap \Upsilon(p) \neq \emptyset$.

The validation algorithm as proposed has two possible outcomes: either it is proved that a forbidden state cannot be reached within the time limit considered or that there exists a counter-example that cannot be refuted. Since the validation procedure relies on over-approximations, it cannot be guaranteed that this abstract counter-example corresponds to a concrete one. An abstract counter-example is true if it includes a concrete one, otherwise it is spurious. This fact is due to the over-approximation of the abstraction. Informally speaking, a concretization of a counter-example adds more trajectories that might not correspond to real ones. We say that a counter-example is spurious according to the following definition:

Definition 4.3.2. ***Spurious Counter-example.***
A trace $\Omega = (\sigma, \tau, \lambda)$ of the AMS system is a spurious counter-example with respect to the property $\mathbf{G}p$, if $\sigma_n \cap \Upsilon(p) \neq \emptyset$ but $\nexists \rho \in \Upsilon(\sigma)$ and $\rho_n \cap \Upsilon(p) \neq \emptyset$.

When using over-approximations, there is no guarantee that a spurious counter-example can be refuted. Technically, this happens if the approximation is too coarse because the current bounds are too large and permit behaviors that are impossible in reality. It is indicative of a very slim error margin separating the reachable states from the bad ones. The likelihood of refuting spurious counter-examples can be increased, however, by using tighter approximations. Hence, refining the over-approximation is necessary until the system is proven safe after closer analysis, or the system is considered fragile because it is unsafe for a sufficiently small value of bound tolerance ε. In other words, if a counter-example that reaches a bad state with a distance $< \varepsilon$ has been found, we say that the concrete system is unsafe with fragility [20].

Definition 4.3.3. A counter-example is called fragile if any disturbance of arbitrarily

small positive tolerance level of its states makes it safe.

Such property is of great importance in the termination of the counter-example refinement as proposed in [34] and hinted in [20]. If we have a trace of counter-example, before going to refinement procedure, we measure the fragility of the trace, if it is fragile, then we conclude that the design is overall fragile with respect to the safety property and therefore we need to redesign the parameters.

4.3.1 Counter-example Generation and Validation

The straightforward method to obtain tighter enclosure of the reachable flow is to increase the order of the Taylor polynomial expansion of the dynamics. Starting from an abstract initial set of states and with increased polynomial order check the validity of the trace. If bad states are not reachable, then we are done and verification terminates. If bad states are reached, a counter-example is generated. If the counter-example is a valid one then verification terminates; otherwise, a refinement procedure is applied, and verification is re-applied.

Inevitably, increasing the order of the Taylor expansion, will require the symbolic analysis algorithms to deal with more polynomial terms which can be expensive in terms of memory and time resources. Instead, we propose a counter-example procedure that takes advantage of the symbolic representation of the structure of Taylor models in order to generate counter-examples and validate them.

As was described before, at any time instant, the system of equations are functions only of the initial states represented symbolically using first order polynomial terms. Thus, we are not obliged to generate a whole trace for the counter-example, it is only sufficient to identify the initial states that might cause the bad behavior. A validation procedure validates whether those initial states will eventually lead to bad states violating the property of interest or that the counter-example was spurious due to over-approximation.

The AMS behavior can be described using a concatenation of continuous traces according to switching rules (discrete) as described in Chapter 3. Thus showing that any

one of the discrete transitions in the counter-example is spurious is a sufficient condition for the non-existence of a corresponding concrete trace. This is clear from the fact that given an initial condition, if a state cannot be reached using the Algorithm 1, then no trace can exists that includes this state and starting from the same initial condition. Technically, two procedures for the refinement of the discrete and continuous dynamics can be used to implement this observation. Refinement of the discrete dynamics is is based on checking whether a switching condition changes from **X** to **F**. If this is the case, then the counter-example is refuted. The refinement of the continuous dynamics first subdivides of the the initial states and then calls the Liveness Verification $\mathbf{F}_{<T_f}p$ function $F\_Verify(.)$ for validation. If the function returns True, then the counter-example is a concrete one, otherwise we call a procedure to check wether the counter-example is spurious or fragile.

The counter-example procedure is described in Algorithm 7. Given the reachable states that are a subset of the bad states (Line 1), we identify the corresponding initial interval states $\overleftarrow{\mathbf{a}} \in \mathbf{a}$ (Line 2). Next, we verify whether those initial states will truly lead to a bad behavior or not (Lines 3 -16). This can be done through two complementary methods. First, we check the switching conditions (Lines 6 - 8). If the valuation of a switch is proved not satisfied, then we conclude that no trajectory initiated from the selected initial condition will lead to a property violation. Otherwise, we construct the corresponding trajectory starting(Lines 12 - 13). If the bad region is reached (Line 12), then we have a concrete counter-example. Otherwise a fragility based refinement and analysis of the trace is applied (Lines 17 - 19).

Note. Counter-example generation and validation for $\mathbf{F}p$ can be obtained by validating the dual property $\mathbf{G}\neg p$. If $\mathbf{G}\neg p$ if *True*, then the reachable states form a non-spurious counter-example, this is due to the over-approximation of the reachable states. If the property is *False*, then get a counter-example. If the counter-example is proved not spurious, then $\mathbf{F}p$ is *True*, otherwise, the counter-example is refined to check its validity.

Example 4.3.1. Consider the circuit in Figure 3.4, where we would like to check the safety property that the voltage will never go below a certain value $\mathbf{G}Vc_2 > -0.60$ for

Algorithm 7 Counter-example Generation and Refutation for Safety Properties: $CE\_Analysis(p, x[k], t_k)$

Require: $\mathbf{X}[n] = \{x[n] | n \in \mathbb{N} \ \& \ n < k\}$
Require: $\widetilde{\mathbf{x}}[k] = eval(\mathbf{x}[k], \{\mathbf{a}, \Delta\})$
Require: $\mathbf{B} = \| \ p \ \|$
1: $\widehat{B}_k = \widehat{\mathbf{x}}[k] \cap B$
2: $Q = \{ \overleftarrow{\mathbf{a}} \mid \exists \overleftarrow{\mathbf{a}} . \widetilde{\mathbf{x}}[k] \subseteq \widehat{B}_k \wedge \overleftarrow{\mathbf{a}} \in \mathbf{a} \}$
3: **for** $m = |Q|$ Down To 1 **do**
4: **for** $n = 0$ to $k - 1$ **do**
5: $\widehat{\mathbf{x}}_{CE}[n] = eval(\widetilde{\mathbf{x}}[n], \{\overleftarrow{\mathbf{a}}_m\})$
6: **if** $Eval\_Cond(\widehat{\mathbf{x}}_{CE}[n]) ==$ **F** and $Eval\_Cond(\widehat{\mathbf{x}}[n]) ==$ **X then**
7: $Q = Q / \overleftarrow{\mathbf{a}}_m$
8: **Exit_Loop**
9: **end if**
10: **end for**
11: **if** $\overleftarrow{\mathbf{a}}_m \in Q$ **then**
12: **if** $F\_Verify(p, \mathbf{x}_{CE}[n], O_t, \Delta_0, T_0) ==$ True **then**
13: **Return** $CE = \overleftarrow{\mathbf{a}}_m$
14: **end if**
15: **end if**
16: **end for**
17: **if** $Q \neq \emptyset$ **then**
18: **Call** $Check\_Fragile(\widetilde{\mathbf{x}}_{CE}[n], \Delta, \| \ p \ \|^{\epsilon})$
19: **end if**

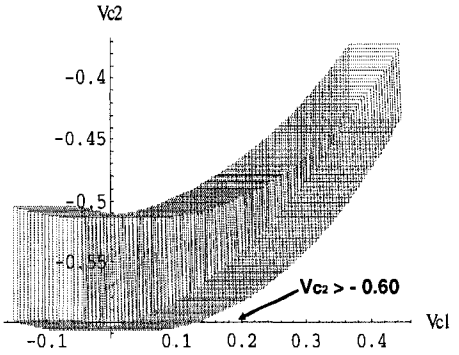
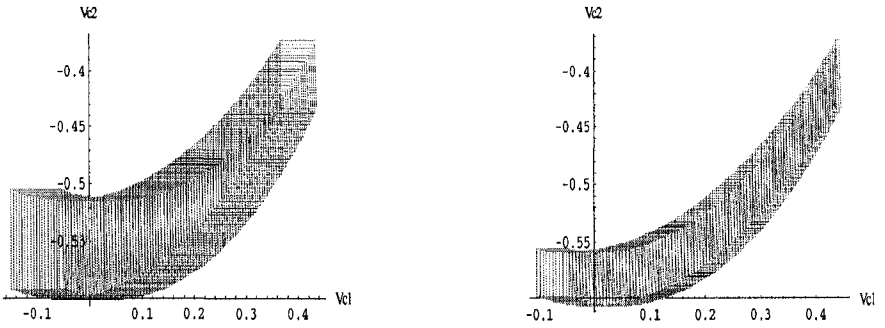


Figure 4.4: Behavior Violation for Circuit in Example 3.4

a given set of initial condition $a \in [-0.03, 0.05]$ and $b \in [-0.03, 0.03]$. We see that the property is violated as shown in Figure 4.4.

By applying the counter-example algorithm, we can identify that the property is verified for $a \in [-0.03, 0.04034[$ (See Figure 4.5(a)). Left is to check whether counter-examples in $a \in [0.04034, 0.05]$ are spurious or not. Using the notion of fragility, by measuring the distance from the bad states, we find that the initial constraint $a \in [0.04034, 0.05]$ leads to a counter-example as shown in Figure 4.5(b).



(a) Safe Behavior (b) Counter-Example

Figure 4.5: Behavior Analysis for Circuit in Example 3.4

In general the efficiency of the counter-example validation depends on the algorithms used in order to minimize the possible counter-example candidates. In this chapter,

we propose a validation algorithm based on checking fragments of the provided counter-example. If one can refute a fragment of a counter-example, e.g., a single transition, then the entire counter-example is spurious.

4.4 Applications

We have implemented the algorithms described in this chapter in Mathematica (See Appendix A for more details). We have applied the proposed verification methodology to different classes of AMS designs representing various design levels, e.g., continuous-time $\Delta\Sigma$ modulator at the behavioral level, Schmitt trigger at the macro-level and oscillators at the circuits level.

4.4.1 Tunnel Diode Circuit

The tunnel diodes exploit a phenomenon called resonant tunneling to provide interesting forward-bias characteristics, due to its negative incremental resistance characteristic at very low forward bias voltages. This means that for some range of voltages, the current decreases with increasing voltage. This is in contrast with conventional diodes that have a non-linear I-V characteristic, but the slope of the curve is always positive. This characteristic makes the tunnel diode useful in oscillator circuits. When a small forward-bias voltage is applied across a tunnel diode, it begins to conduct current. As the voltage is increased, the current increases and reaches a peak value called the peak current. If the voltage is increased a little more, the current actually begins to decrease until it reaches a low point called the valley current. If the voltage is increased further yet, the current begins to increase again, this time without decreasing into another valley.

We focus on the current I_L and the voltage V_C across the tunnel diode in parallel with the capacitor of a serial RLC circuit (see Figure 4.6). The state equations of the circuits are given as

$$\dot{V_C} = \frac{1}{C}(-I_d(V_C) + I_L)$$

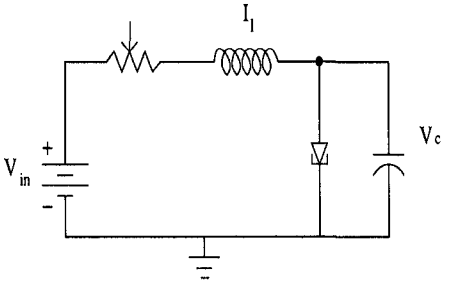


Figure 4.6: Tunnel Diode Oscillator

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 behavior. We analyze the circuit in two modes. The first when the circuit is in stable oscillation for a given set of parameters, the other case when this oscillation dies out. We chose these two different sets of parameters values of the oscillator circuit $\{C = 1000e^{-12},\ L = 1e^{-6},\ G = 5000e^{-3},\ Vin = 0.3\}$ and $\{C = 1000e^{-12},\ L = 1e^{-6}, G = 2000e^{-3}, Vin = 0.3\}$ along with the set of initial values of voltages $[0.8\ V,\ 0.9\ V]$ and currents $0.04\ mA$ and the analysis region of interest $-1\ V \leq V_C \leq 1\ V$ and $0.01\ mA \leq I_L \leq 0.9\ mA$. Suppose we want to verify the following property on the set of trajectories [50]:

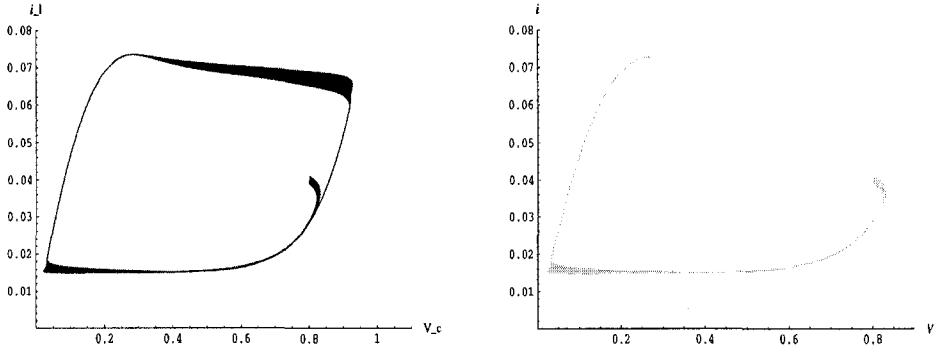
$$\mathbf{G}_{[0,1e^{-6}]}(\mathbf{F}_{[0,6e^{-7}]}(I_L \leq 0.02)) \wedge \mathbf{G}_{[0,1e^{-6}]}(\mathbf{F}_{[0,6e^{-7}]}(I_L \geq 0.06))$$

which can be understood as within the time interval $[0, 1e^{-6}]$ on every computation path, the I_L amplitude will always reach 0.02 within the time interval $[0, 0, 6e^{-7}]$, the same goes for the I_L amplitude 0.06. This property checks for oscillation behavior of the circuit.

By applying Algorithm 6, with the first set of parameters, we have the property satisfied, which means that the circuit is oscillating for the given set of initial conditions, within the specified time interval. The Taylor model based reachable states are shown in Figure 4.7.a.

By following the same procedure for the system with the second set of parameters, but with the same initial conditions, we can find out that the circuit is non oscillating.

Physically, when the circuit starts up, the energy of the system is lost due to the positive circuit resistance. Starting from any point in the analysis region, the oscillations die down to the equilibrium point as illustrated in Figure 4.7.b.



a. Oscillations b. No Oscillation

Figure 4.7: Oscillator Behavior

4.4.2 Schmitt Trigger

In electronics, a Schmitt trigger is a comparator circuit that incorporates positive feedback. When the input is higher than a certain chosen threshold, the output is high; when the input is below another (lower) chosen threshold, the output is low; when the input is between the two, the output retains its value, until the input changes sufficiently to trigger a change. This dual threshold action is called hysteresis, and implies that the Schmitt trigger has some memory. Schmitt trigger can be used as an oscillator as shown in Figure 4.8 with the following configuration (The state equations):

$$C_1 \frac{dv_{C_1}}{dt} = \frac{v_{out} - v_{C_1}}{R_2} - \frac{v_{C_1}}{R_1}$$

and

$$C_2 \frac{dv_{C_2}}{dt} = \frac{v_{out} - v_{C_2}}{R_s}$$

where

$$v_{out} = V_{MAX} Tanh(\frac{V_{C_1} - V_{C_2}}{V_T})$$

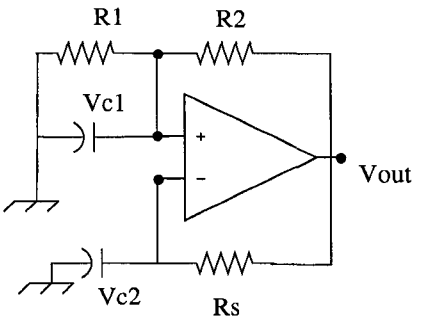


Figure 4.8: Schmitt Trigger Oscillator

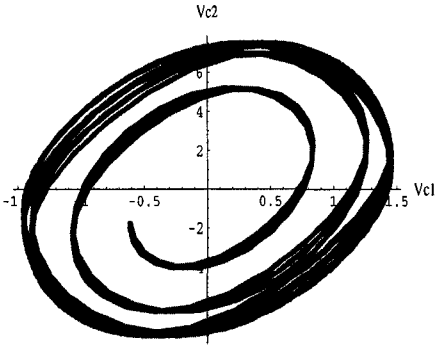


Figure 4.9: Schmitt Trigger Oscillator Behavior

with $V_{max} = 5$ and $V_T = 0.025$.

Similar to the tunnel diode, we check for the oscillation property:

$$\mathbf{G}_{[0,2e^{-3}]}(\mathbf{F}_{[0,0.2e^{-3}]}(V_{c2} \leq -4)) \wedge \mathbf{G}_{[0,2e^{-3}]}(\mathbf{F}_{[0,0.2e^{-3}]}(V_{c2} \geq 4))$$

which can be understood as within the time interval $[0, 2e^{-3}]$ on every computation path, whenever the V_{c2} amplitude will reach -4 Volts, it will reach this value again within the time interval $[0, 0.2e^{-3}]$, the same goes for V_{c2} reaching this amplitude 4 Volts. By applying Algorithm 6, we have the property satisfied, which means that the circuit is oscillating for the given set of initial conditions, within the specified time interval. The possible Taylor model based reachable states are shown in Figure 4.9.

4.4.3 Continuous-Time ΔΣ Modulator

Data converters are needed at the interface of analog and digital processing units. The principle of the ΔΣ architecture is to make rough evaluations of the signal over several stages, to measure the error, integrate it and then compensate for that error.

A ΔΣ modulator is said to be stable if the integrator output remains bounded under a bounded input signal, thus avoiding overloading the quantizer in the modulator. This property is of a great importance since the integrator saturation can deteriorate circuit performance, hence leading to instability. The quantizer in the modulator is a one-bit quantizer with two quantization levels, $+1$V and -1V. Hence, the quantizer input should be between -2V and $+2$V in order to avoid overloading. The Continuous-time ΔΣ shown in Figure 4.10 can be represented by the following equations:

$$\frac{dx_0}{dt} = b_0 x_1 - k_0 x_0 - b_0 a_0 M tanh \frac{p x_0(t-\tau)}{M}$$

and

$$\frac{dx_1}{dt} = b_1 u(t) - k_1 x_1 - b_1 a_1 M tanh \frac{p x_1(t-\tau)}{M}$$

Stability criteria can be formalized as a safety property ensuring that the integrators' output voltage will never exceed certain bounds. The property can be stated as follows:

$$\mathbf{G} - 1 < V_{c_2} < 3.5$$

The reachable states for different initial conditions and input voltages are shown in Figure 4.11.

As illustrated in Figure 4.11(a), the voltage V_{c_2} will be confined with the region specified in the property and applying Algorithm 4, we find that the property will be satisfied. Increasing the input signal voltage leads to instability and the property is not verified as illustrated in Figure 4.11(b).

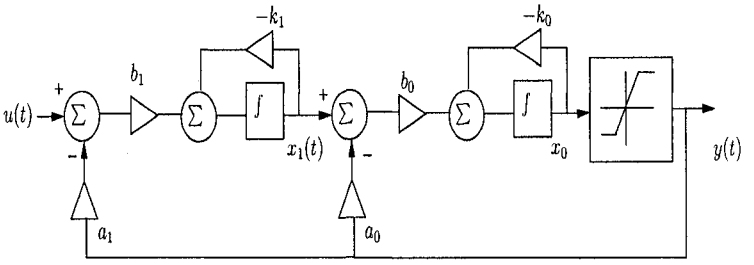
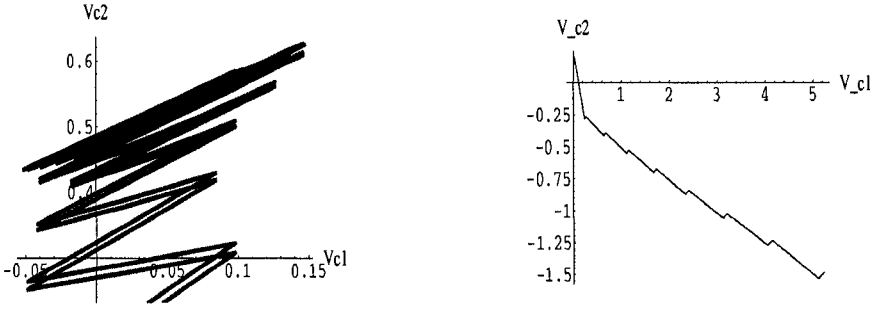


Figure 4.10: Continuous-Time $\Delta\Sigma$ Modulator



(a) Stability (b) Instability

Figure 4.11: *DSM* Modulator

4.5 Summary

In this chapter, we have defined a bounded model checking approach for AMS systems modeled using a combination of SREs and differential equations. We have proposed a symbolic-interval modeling of the state space using the principle of Taylor models which provide a way for representing a combination of representation using a combination of polynomials and interval terms. The main advantage of such modeling is the fact, that the polynomial representation helps slowing the divergence due to the over-approximated intervals, while the interval part provides an important abstraction to handle the continuous behavior. In order to enhance the methodology, we extended the verification is a counter-example generation/refinement procedure. We have implemented our methodology using libraries for symbolic computation available in *Mathematica*. Experimental results have shown the feasibility and the utility of the approach.

The proposed BMC algorithm can verify properties for only a bounded time, however, confidence in the verification process would be increase by removing this constraint. To this end, in the next chapter, we complement the BMC algorithm by an abstraction methodology based on using invariant checking and predicate abstraction.

Chapter 5

Qualitative Abstraction for CT-AMS Verification

5.1 Overview

Bounded model checking is an attractive method for verifying properties by partial exploration of the state space for a finite time period. This approach was shown in the previous chapter to be successful in proving properties such as oscillatory behavior. Nevertheless, confidence in the verification is limited due to the incompleteness of the verification. Consider for instance, the proof of nonexistence of oscillatory behavior. Such an example among others, motivate the development of a complementary methods to increase confidence in the verification process.

Predicate abstraction is one of the most successful abstraction approaches originally developed in [45], for the verification of systems with infinite state space. In this approach, the state space is divided into a finite set of regions and a set of rules is used to build the transition relation between these regions in a way that the generated state transition system can be verified using model checking. Among the proposed enhancements of predicate abstraction is the lazy abstraction approach [58]. The basic idea here is instead of generating the entire abstract model, a region is abstracted only when it is needed in

the verification step. Refinement is applied starting from the earliest state at which the abstract counterexample fails to have a concrete counterpart.

Inspired by the concept of lazy abstraction, we propose a qualitative abstraction approach for continuous-time AMS designs, such that satisfaction of the property in the abstract model guarantees its satisfaction in the original design. In the proposed abstraction, the state space is initially partitioned based on the qualitative properties of the analog behavior and symbolic constrained based methods are applied to check for property validation. In case of failure, an iterative verification/refinement process is applied where the regions violating the property are refined and symbolic model checking is applied for the property validation.

The verification methodology we propose is illustrated in Figure 5.1. Starting with a circuit description as a system of ODEs (See Definition 3.2.3, Chapter 3), along with specification properties provided in computational temporal logic ($\forall$CTL) (See Section 3.3.2, Chapter 3), we symbolically extract qualitative predicates of the system. The abstract model is constructed in successive steps. In the basis step, we only consider predicates that define the invariant regions for the system of equations based on the Darboux theory of integrability [43]. Informally, the Darboux theory is concerned with the identification of the different qualitative behaviors of the continuous state space of the system. We make use of this idea to divide the analog behaviors of the design into qualitatively distinct regions where no transition is possible between states of the different regions. Satisfaction of properties is verified on these regions using constraint based methods, which rely on qualitative properties of the system, by generating new constraints that prove or disprove a property. The property verification hence provides the advantage of avoiding explicit computation of reachable sets.

If the property cannot be verified at this stage, refinement is needed only for the non-verified regions by adding more predicates. Conventional model checking is then applied on the newly generated abstract model. The extraction of the predicates is incremental in the sense that more precision can be achieved by adding more information to the original

construction of the system. When the property is marked violated, one possible reason is because of the false negative problem due to the over-approximation of the abstraction. In this case, refinement techniques may be introduced.

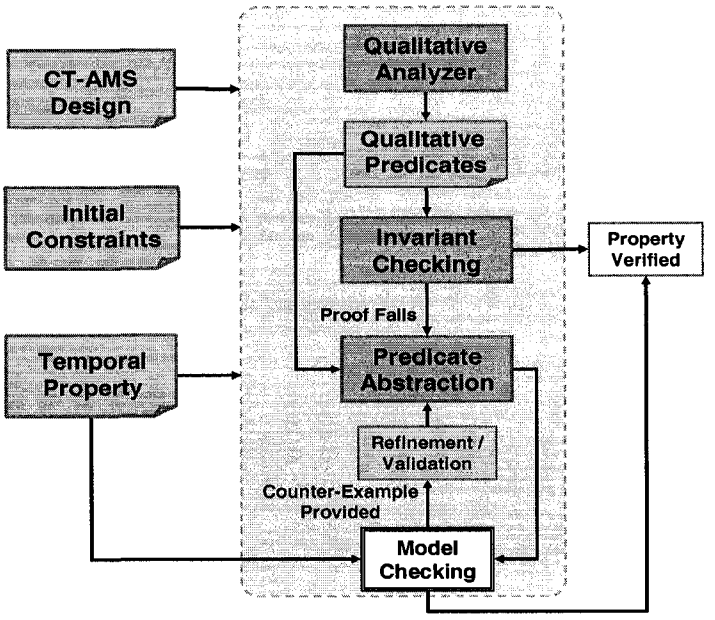


Figure 5.1: Qualitative Abstraction based Verification Methodology

5.1.1 Predicate Abstraction

In the abstraction method, we start first by defining the abstract states and the maps from concrete to abstract states. An abstract transition system is then created by constructing the abstract initial states and abstract transition relations. In order to fulfill these steps a sound relationship between the concrete and abstract domain should be defined.

Predicate abstraction is a method where the set of abstract states is encoded by a set of Boolean variables representing each a concrete predicate. Based on [5], we define a discrete abstraction of the CT-AMS model with respect to a given n-dimensional vector of predicates $\Psi = (\psi_1, \ldots, \psi_n)$, where $\psi_n : \mathbb{R}^d \rightarrow \mathbb{B}$, with $\mathbb{B} = \{0, 1\}$ and d is the dimension of the system of ODEs. A polynomial predicate is of the form $\psi(x) := \mathcal{P}(x_1, \ldots, x_d) \sim 0$,

where $\sim \in \{<, \geq\}$. Hence, the infinite state space $\mathcal{X}$ of the system is reduced to 2^n states in the abstract system, corresponding to the 2^n possible Boolean truth valuations of Ψ.

Definition 5.1.1. ***Abstract Transition System.***

An abstract transition system is a tuple $\mathcal{T}_\Psi = (Q_\Psi, \rightsquigarrow, Q_{\Psi,0})$, where:

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

We define the set of reachable states as: $Reach_\Psi = \bigcup_{i \geq 0} Reach_\Psi^{(i)}$, where $Reach_\Psi^{(0)} = Q_{\Psi,0}$, $Reach_\Psi^{(i+1)} = Post_c(Reach_\Psi^{(i)}), \forall i \geq 0$ and $Post_c(l, b) := \{(l', b') \in Q_\Psi | (l, b) \rightsquigarrow (l', b')\}$. We can then deduce the following property between concrete and abstract reachable states.

Statement. Given a CT-AMS transition system (See Definition 3.2.5) and an abstract transition system with a vector of predicates Ψ, the following holds: $Reach \subseteq \{q \in Q | \exists (l, b) \in Reach_\Psi : x \in \Upsilon_\Psi(b) \wedge L_x(q) = x\}$

5.1.2 Abstraction Based Verification

Given a CT-AMS model transition system $\mathcal{T}_{\mathcal{AMS}}$ and a property φ expressed in $\forall$CTL, the problem of checking that the property holds in this model written as $\mathcal{T}_{\mathcal{AMS}} \models \varphi$ can be simplified to the problem of checking that a related property holds on an approximation of the model $\mathcal{T}_\Psi$, i.e., $\mathcal{T}_\Psi \models \tilde{\varphi}$, with $\tilde{\varphi} = \mu(\varphi)$, where μ is a mapping function: $\mu : \mathbb{R}^d \rightarrow \mathbb{B}$ which is a function associating to each predicate $\lambda(x_1, \ldots, x_d)$ an atomic proposition P. The main preservation theorem can be stated as follows [20]:

Theorem 5.1.1. Suppose $\mathcal{T}_\Psi$ is an abstract model of $\mathcal{T}_{\mathcal{AMS}}$, then for all $\forall$ CTL state formulas describing $\mathcal{T}_\Psi$ and every state of $\mathcal{T}_{\mathcal{AMS}}$, we have $\tilde{s} \models \tilde{\varphi} \Rightarrow s \models \varphi$, where $s \in \gamma(\tilde{s})$. Moreover, $\mathcal{T}_\Psi \models \tilde{\varphi} \Rightarrow \mathcal{T}_{\mathcal{AMS}} \models \varphi$.

If a property is proved on an abstract model $\mathcal{T}_\Psi$, then we are done. If the verification of $\mathcal{T}_\Psi$ reveals $\mathcal{T}_\Psi \not\models \tilde{\varphi}$, then we cannot conclude that $\mathcal{T}_{\mathcal{AMS}}$ is not safe with respect to $\tilde{\varphi}$, since the counterexample for $\mathcal{T}_\Psi$ may be spurious. In order to remove spurious counterexamples, refinement methods on the abstract model can be applied [20].

5.1.3 Invariants

Usually, a system with continuous dynamics (e.g., an AMS design) has a behavior that varies in different regions of the phase space whose boundaries are defined by special system solutions known in the literature as Darboux invariants [43]. These invariants partition the concrete state space into a set of qualitative distinctive regions [1].

Definition 5.1.2. Given the system of ODEs $\frac{dx_k}{dt} = \mathcal{P}_k(x_1(t),\ldots,x_d(t))$, with $k = 1,\ldots d$ ($\frac{d\mathbf{x}}{dt} = \mathbf{P}(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{P} = (\mathcal{P}_1,\ldots,\mathcal{P}_d)$) is a polynomial vector field, we define the corresponding vector field as $\mathcal{D}_\mathbf{P} = \mathbf{P}.\partial_\mathbf{x} = \sum_{k=1}^{d} \mathcal{P}_k \frac{\partial}{\partial x_k}$.

The correspondence between the system of ODEs and the vector field $\mathcal{D}_\mathbf{P}$ is obtained by defining the time derivative of functions of $\mathbf{x}$ as follows. Let $\mathcal{G}$ be a function of $\mathbf{x}$: $\mathcal{G} : \mathbb{R}^k \rightarrow \mathbb{R}$, then $\frac{d\mathcal{G}}{dt} := \dot{\mathcal{G}} = \mathcal{D}_\mathbf{P}(\mathcal{G}) = \mathbf{P}.\partial_\mathbf{x}\mathcal{G}$. The time derivative is called the derivative along the flow since it describes the variation of function $\mathcal{G}$ of $\mathbf{x}$ with respect to t as $\mathbf{x}$ evolves according to the differential system. When $D_\mathbf{P}(\mathcal{G}) = 0$, $\forall \mathbf{x} \in \mathbb{R}^k$, we have a time independent first integral of $D_\mathbf{P}$. Several methods were developed recently based on Darboux integrability theory [43], which is a theory concerned with finding closed form solutions of system of ODEs, to tackle the problem by looking for a basis set of invariants, i.e., Darboux invariants. Rather than looking at functions which are constant

[1] We will focus on the analog part of the AMS design. Therefore, from now on, when we mention ODEs, we will assume a system of equation with no discrete part.

on all solutions, we look at functions which are constant on their zero level set. Darboux polynomials $\mathcal{J}_i$ provide the essential skeleton for the phase space from which all other behaviors can be qualitatively determined.

Definition 5.1.3. ***Darboux Polynomials*** **[43].**
Given a vector field $\mathcal{D}_{\mathcal{P}} = \sum_{i=1}^{d} \mathcal{P}_i \frac{\partial}{\partial x_i}$ associated with the system $\frac{d\mathbf{x}}{dt} = \mathbf{P}(\mathbf{x})$, a Darboux polynomial is of the form $\mathcal{J}(\mathbf{x}) = 0$ with $\mathcal{J} \in \mathbb{R}[\mathbf{x}]$, with $D\mathcal{J} = \mathcal{K}\mathcal{J}$, where $\mathcal{K} = \mathcal{K}(\mathbf{x})$ is a polynomial called the cofactor of $\mathcal{J} = 0$.

Lemma 5.1.1. [43] Given a system of ODEs and a vector field $\mathcal{D}_{\mathbf{f}}$, $\mathcal{J}$ is an invariant of the system if $\mathcal{J}$ divides $\mathcal{D}_{\mathbf{f}}$, more formally, if there exists $\mathcal{K} \in \mathbb{R}[\mathbf{x}]$ such that $\mathcal{D}_{\mathbf{f}}(\mathcal{J}) = \mathcal{K}\mathcal{J}$. The solution set of the system vanishes on the curve of $\mathcal{J}$.

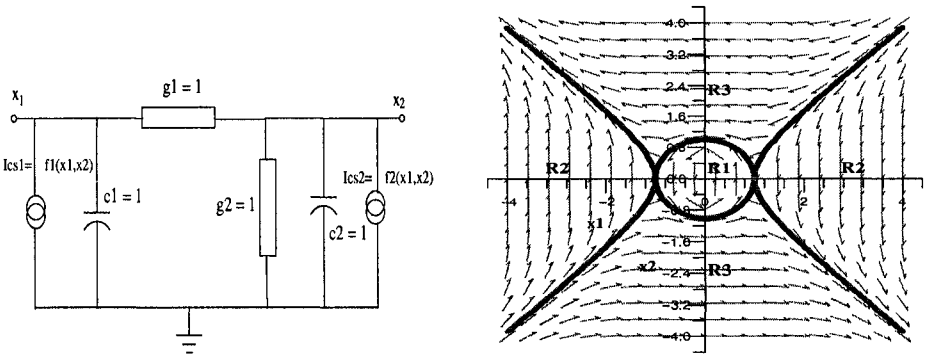
Proof. We can always represent the system by the associated vector field at each point $\mathcal{F}(\mathbf{x}) = \mathbf{P}(\mathbf{x})$ and $\nabla\mathcal{J} \cdot \mathcal{F} = k\mathcal{J}$, where $\nabla\mathcal{J}$ denotes the gradient vector related to $\mathcal{J}(\mathbf{x})$ and $\cdot$ is the scalar product. When $\mathcal{J} = 0$, $\nabla\mathcal{J} \cdot \mathcal{F} = 0$, meaning that $\nabla\mathcal{J}$ is orthogonal to the vector field $\mathcal{F}$ at these points. Therefore $\mathcal{F}$ is tangent to $\mathcal{J} = 0$.

In the context of abstraction, we define the invariant regions as conjunctions of Darboux invariant predicates. An invariant region can be considered as an abstraction of the state space that confines all the system dynamics initiated in that region:

Definition 5.1.4. We say that a region $\mathcal{V}$ is an invariant region of a CT-AMS model such that $\mathcal{P}(\mathbf{x}(0)) = s_0 \models \mathcal{V}$, $\mathcal{P}(\mathbf{x}(\varsigma)) = \mathbf{s}_\varsigma \models \mathcal{V}$ *and* $\forall t \in [0, \varsigma], \mathcal{P}(\mathbf{x}(t)) = s_t \models \mathcal{V}$. Let $\mathcal{V} = \{x \in \mathbb{R}^k | x \models \Gamma\}$, be an invariant region, where Γ is a conjunct of Darboux predicates (each is of the form $p(\mathbf{x}) \sim 0$, where p is a polynomial function and $\sim \in \{<, \geq\}$). If $\mathbf{x}(0)$ is some initial state, then $\mathcal{V} = \mathcal{V}(\mathbf{x}(0))$ denotes an over-approximation of the set of states reachable from $\mathbf{x}(0)$.

Example 5.1.1. Consider the non-linear circuit shown in Figure 5.2.a, where the non-linearity comes from the voltage controlled current sources that produce currents I_{cs1} and I_{cs2} , respectively, with $f_1 = -x_2^3 + x_1 - x_2$ and $f_2 = -x_1^3 + 2x_2$. The voltages across

the capacitors c_1 and c_2 can be described using ODEs, respectively, as follows: $\dot{x_1} = -x_2^3$ *and* $\dot{x_2} = x_1 - x_1^3$. We identify the corresponding invariants: $j_1 = 1 - x_1^2 - x_2^2$ and $j_2 = 1 - x_1^2 + x_2^2$, which are used to form three invariant regions: $R_1 = j_1 \geq 0 \wedge j_2 \geq 0$, $R_2 = j_1 < 0 \wedge j_2 < 0$ and $R_3 = j_1 < 0 \wedge j_2 \geq 0$ as shown in Figure 5.2.b. Note that $j_1 \geq 0 \wedge j_2 < 0$ is infeasible and therefore discarded.



(a) Circuit Schematic

(b) Darboux Invariants

Figure 5.2: Illustrative Non-linear Circuit

5.2 Invariants Based Verification

In this section, we propose a qualitative verification approach for the AMS designs using constraint based methods. The basic idea is to apply quantified constraint based techniques to answer questions about qualitative behaviors of the designs, by constructing functions that validate or falsify the property. The idea is different from conventional approaches as it does not require an explicit reachable states computation. We consider two types of properties that can be verified using this approach, namely safety and switching properties.

5.2.1 Safety Properties

Safety properties can be expressed in *CTL* [22] as $\forall\mathbf{G}p$; meaning that always on all executions the constraint predicate p is satisfied for a set of initial conditions. The verification starts by getting the negated property $\exists\mathbf{F}\neg p$ (which means that there is an execution falsifying the constraint p) and applies constraint solving on the dual property within the invariant regions of interest. In case of unsatisfiability, we conclude that the original property is satisfied in the region, otherwise we cannot conclude the truth of the property and a refinement model providing more details of the region is constructed.

Proposition 5.2.1. Safety Property Verification.
$\forall\mathbf{G}\mathcal{P}$ is always satisfied in an invariant region $\mathcal{V}$, if its dual property $\exists\mathbf{F}\neg\mathcal{P}$ is never satisfied in that region.

Proof. The proof is straightforward as $\exists\mathbf{F}\neg\mathcal{P}$ is the complement of $\forall\mathbf{G}\mathcal{P}$. One and only one of both properties can be satisfied in a given invariant region.

Example 5.2.1. Consider the circuit in Example 5.1.1, with initial conditions $x_1(0) \in [-1.1,-0.7]$ and $x_2(0) \in [0.5,0.9]$. Suppose the property to check is $\forall\mathbf{G}\mathcal{P} := x_1^2+x_2-3<0$ (see Figure 5.3 for details), meaning that all flows initiated from $\mathbf{x}(0)=(x_1(0),x_2(0))$, will be bounded by $x_1^2+x_2-3$. The following regions satisfy the initial conditions $R_1 = j_1 \geq 0 \wedge j_2 \geq 0$ and $R_3 = j_1 < 0 \wedge j_2 \geq 0$. We check whether $\exists\mathbf{F}\mathcal{P} := x_1^2+x_2-3 \geq 0$ is satisfiable in the invariant regions R_1 and R_3. By applying constraint solving in *Mathematica*, we find that for the region R_3, the constraints system is satisfiable, hence the original property cannot be verified, and the state space of the region needs to be refined. For the region R_1, the constraints system is infeasible, therefore we conclude that the safety property is satisfied.

It is worth noting that the barrier-certificate method developed in [92] can be applied as complementary to our method. In fact, Darboux predicates used as basis of invariant regions can be considered as natural barrier certificates that are constructed without the

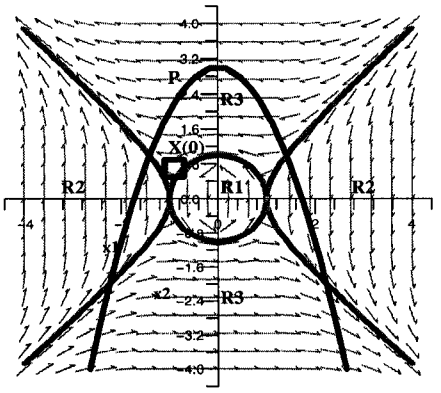


Figure 5.3: Safety Verification (Example 5.2.1)

need of initial and final constraints. Therefore the main advantage is that they can be used in the verification for several initial and properties, hence reducing computational efforts.

5.2.2 Switching Properties

A special case of the reachability verification $\exists \mathbf{F} Q$ is the switching condition verification, i.e., starting from a set of initial conditions, the system will eventually cross a threshold, triggering a switching condition. Such property is of great importance, for instance, a MOS transistor acting as switch changes states based on the voltage condition applied on its gate. We consider here a restricted form of the switching property, where we assume that threshold predicates divide the invariant region by intersecting the invariant region boundaries (at least two Darboux predicates). Given an invariant region $\mathcal{V}$, a predicate Q is a *switching condition* if $\bigwedge_{i=0}^{k} \exists \mathbf{x}.(Q(\mathbf{x}) = 0) \wedge (I_i(\mathbf{x}) = 0)$, where $k \leq 2$ and I is a Darboux invariant. The switching verification can be stated as follows:

Proposition 5.2.2. ***Switching Property Verification.***
$\exists \mathbf{F} Q$ is satisfied in a region $\mathcal{V}$, if $Q(\mathbf{x}(0)) < 0$ and $\mathcal{D}_{\mathbf{P}}(Q) > 0$ or if $Q(\mathbf{x}(0)) > 0$ and $\mathcal{D}_{\mathbf{P}}(Q) < 0$, in the region $\mathcal{V}$. If these conditions are satisfiable, we conclude that the property is verified and switching occurs.

Proof. proof by contradiction. Suppose that:

1. The condition that $\bigwedge_{i=0}^{k} \exists \mathbf{x}.(Q(\mathbf{x}) = 0) \wedge (I_i(\mathbf{x}) = 0)$ holds
2. $Q(\mathbf{x}(0)) < 0$ and $\mathcal{D}_{\mathbf{P}}(Q) > 0$ is satisfied
3. $\exists \mathbf{F} Q$ is not satisfied.

From the condition in (1) and the vector field behavior in (2), we deduce that there exists a trajectory starting from a state $\mathbf{x}(0)$ to a state $\mathbf{x}(f)$ such that $\mathbf{x}(f) \models Q$. Therefore, contradicting assumption (3). The proof is similar for a vector field with the following behavior: $Q(\mathbf{x}(0)) > 0$ and $\mathcal{D}_{\mathbf{P}}(Q) < 0$.

Example 5.2.2. Consider the circuit shown in Figure 5.2.a, where the voltages across the capacitors c_1 and c_2 are described, respectively, as follows: $\dot{x_1} = x_1^2 + 2x_1x_2 + 3x_2^2$ and $\dot{x_2} = 4x_1x_2 + 2x_2^2$. Suppose that the switching condition property to check is $\exists \mathbf{F} x_1 + x_2 - 5 = 0$, meaning that switching occurs when a certain trajectory will cross the threshold $Q_1 := x_1 + x_2 - 5 = 0$ (see Figure 5.4). We construct the Darboux functions: $j_1 := x_2, j_2 := x_1 + x_2, j_3 := x_1 - x_2$. The region $R_1 = j_1 > 0 \wedge j_2 > 0 \wedge j_3 > 0$ satisfies the initial conditions. In addition, the predicate $x_1 + x_2 - 5 < 0$ satisfies the initial condition and $\mathcal{D}_{\mathbf{P}}(x_1 + x_2 - 5) > 0$ because $\mathcal{D}_{\mathbf{P}}(x_1 + x_2 - 5) = (x_1 + x_2)(x_1 + 5x_2)$ is always positive in R_1. Consider the initial conditions $\mathbf{X}(0)_1 := (x_1(0) \in [-10, -8]$ and $x_2(0) \in [4, 5])$ and $\mathbf{X}(0)_2 := (x_1(0) \in [-0.5, -1]$ and $x_2(0) \in [0.3, 0.5])$ in the invariant region $R_2 = j_1 > 0 \wedge j_2 < 0 \wedge j_3 < 0$. For the switching condition $Q_2 := -x_1 + x_2 - 5 = 0$, we find that the initial condition $\mathbf{X}(0)_1$ satisfies $-x_1 + x_2 - 5 > 0$, and $\mathbf{X}(0)_2$ satisfies $-x_1 + x_2 - 5 < 0$ while $\mathcal{D}_{\mathbf{P}}(-x_1 + x_2 - 5) = -(x_1 - x_2)^2$ will be always negative in region R_2, therefore we conclude that the switching will occur for the initial condition $\mathbf{X}(0)_1$ but not for $\mathbf{X}(0)_2$.

5.2.3 Reachability Verification

A failure in safety verification does not guarantee that the final set is reachable from the initial set. This is the problem of reachability verification, which is concerned with

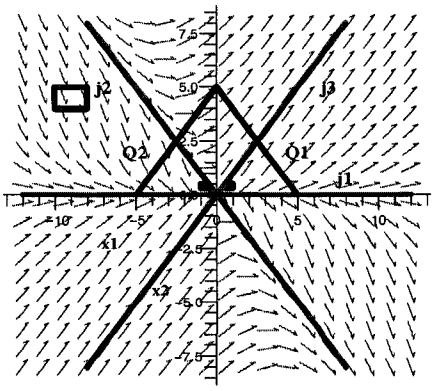


Figure 5.4: Switching Verification (Example 5.2.2)

proving that at least one trajectory of the system starting from a set of initial states will reach another given set of states in a finite time. The reachability property is expressed as $\exists \mathbf{F}\mathcal{P}$, which means, eventually, there exists an execution that will satisfy the constraint P. The main idea of the verification is to find bounds that include a trajectory from an initial to a final state. Reachability can be verified using the following proposition:

Proposition 5.2.3. Sufficient Condition for Reachability.

Given initial (S_{in}) and reachable (S_{fn}) states bounded by convex functions, $\mathbf{B}_{in}$ and $\mathbf{B}_{fn}$ such that

$$\forall s \in S_{in}.\mathbf{B}_{in}(s) \leq 0 \; with \; \mathcal{D}(\mathbf{B}_{in}) > 0|_{\mathbf{B}_{in}=0}$$

and

$$\forall s \in S_{fn}.\mathbf{B}_{fn}(s) \leq 0 \; with \; \mathcal{D}(\mathbf{B}_{fn}) < 0|_{\mathbf{B}_{fn}=0}$$

respectively, construct two functions $\mathbf{B}_{r1}$ and $\mathbf{B}_{r2}$, such that their existence implies the existence of trajectory Φ: $\exists s_0 \in S_{in} \exists s_1 \in S_{fn}.\Phi(t_0) = s_0$ and $\Phi(t_f) = s_1$, where t_0 and t_f are time points with $t_0 < t_f$, bounded by

$$\mathbf{B}_{r1} < 0 \wedge \mathbf{B}_{r2} > 0 \;\; or \;\; \mathbf{B}_{r1} > 0 \wedge \mathbf{B}_{r2} < 0$$

with the following conditions:

1. $(\mathbf{B}_{r1} = 0) \cap (\mathbf{B}_{in} = 0) \neq \emptyset$ and $(\mathbf{B}_{r1} = 0) \cap (\mathbf{B}_{fn} = 0) \neq \emptyset$

2. $(\mathbf{B}_{r2} = 0) \cap (\mathbf{B}_{in} = 0) \neq \emptyset$ and $(\mathbf{B}_{r2} = 0) \cap (\mathbf{B}_{fn} = 0) \neq \emptyset$

3. $\mathcal{D}(\mathbf{B}_{r1}) \geq 0|_{\mathbf{B}_{r1}=0} \wedge \mathcal{D}(\mathbf{B}_{r2}) \leq 0|_{\mathbf{B}_{r2}=0}$ or $\mathcal{D}(\mathbf{B}_{r2}) \geq 0|_{\mathbf{B}_{r2}=0} \wedge \mathcal{D}(\mathbf{B}_{r1}) \leq 0|_{\mathbf{B}_{r1}=0}$.

Proof. Assume that there exists functions $\mathbf{B}_{r1}$ and $\mathbf{B}_{r2}$ satisfying the conditions $(1-3)$, while at the same time there are no reachable states from $\mathbf{B}_{in}$ to $\mathbf{B}_{fn}$. We have four cases:

1. $\mathcal{D}(\mathbf{B}_{r1}) \geq 0|_{\mathbf{B}_{r1}=0} \wedge \mathcal{D}(\mathbf{B}_{r2}) \leq 0|_{\mathbf{B}_{r2}=0}$ and all the flow crossing $\mathbf{B}_{r1}$ and $\mathbf{B}_{r2}$ is going out of the bounded region $\mathbf{B}_{r1} < 0 \wedge \mathbf{B}_{r2} > 0$.

2. $\mathcal{D}(\mathbf{B}_{r1}) \geq 0|_{\mathbf{B}_{r1}=0} \wedge \mathcal{D}(\mathbf{B}_{r2}) \leq 0|_{\mathbf{B}_{r2}=0}$ and all the flow crossing $\mathbf{B}_{r1}$ and $\mathbf{B}_{r2}$ is going inside the bounded region $\mathbf{B}_{r1} > 0 \wedge \mathbf{B}_{r2} < 0$.

3. $\mathcal{D}(\mathbf{B}_{r2}) \geq 0|_{\mathbf{B}_{r2}=0} \wedge \mathcal{D}(\mathbf{B}_{r1}) \leq 0|_{\mathbf{B}_{r1}=0}$ and all the flow crossing $\mathbf{B}_{r1}$ and $\mathbf{B}_{r2}$ is going out of the bounded region $\mathbf{B}_{r1} < 0 \wedge \mathbf{B}_{r2} > 0$.

4. $\mathcal{D}(\mathbf{B}_{r2}) \geq 0|_{\mathbf{B}_{r2}=0} \wedge \mathcal{D}(\mathbf{B}_{r1}) \leq 0|_{\mathbf{B}_{r1}=0}$ and all the flow crossing $\mathbf{B}_{r1}$ and $\mathbf{B}_{r2}$ is going inside the bounded region $\mathbf{B}_{r1} > 0 \wedge \mathbf{B}_{r2} < 0$.

Assume that all flows crossing $\mathbf{B}_{r1}$ and $\mathbf{B}_{r2}$ are going inside a bounded region and that this bounded region does not include a fixpoint, then, we will have at least a function with $\mathcal{D}(\mathbf{B}_{r3}) = k\mathbf{B}_{r3}$, confined in the region and connecting the initial and final regions, hence leading to contradiction. Similar argument for the case where all flows are going outside the bounded region.

It is worth noting that this method gives sufficient condition to prove the existence of a reachable flow. This is a loose condition for the sufficient condition which states that a reachable flow exists in the confined region if there exists $\mathcal{D}(\mathbf{F}) = 0$ in that region. However, this latest condition is hard to implement as such a condition corresponds to finding a first integral as discussed in Section 5.1.3. We limit ourselves in this thesis to the first sufficient condition only.

Example 5.2.3. Consider the non-linear circuit shown in Figure 5.2(a), connected to different current sources with the voltages across the capacitors c_1 and c_2 described using ODEs, respectively, as follows:

$$\dot{x}_1 = 3(x_1^2 - 4) \ \textit{and} \ \dot{x}_2 = 3 + x_1 x_2 - x_2^2$$

Suppose we provide the initial condition $\mathbf{B}_{in} := (-1 + x_1)^2 + (-4 + x_2)^2 \leq 0.5$. We want to verify the following property

$$\exists \mathbf{FB}_{fn}$$

where

$$\mathbf{B}_{fn} := (2 + x_1)^2 + (-1.8 + x_2)^2 \leq 0.5$$

Using quantified constraint solving capabilities in *Mathematica*, we constructed the following bounds:

$$\mathbf{B}_1 := 2.4 + 89x_1^2 + 235.8x_2 = 1000$$

and

$$\mathbf{B}_2 := -74x_1 + 1.3x_2^4 = 130$$

Therefore, we can deduce that the reachability property will indeed be satisfied (a sample reachable trajectory is shown inside the region Figure 5.5).

Sometimes constraint based verification fails to provide answers for the verification problem due to several reasons:

1. The above mentioned verification methods are not complete in general.
2. Sometimes the constraint solver fails to provide an answer (e.g., not able to construct bounds for reachability).
3. More complex properties like oscillation cannot be proved using the above method.

We complement the approaches described in this section, by the predicate abstraction method allowing conventional model checking to be applied. In the next section, we

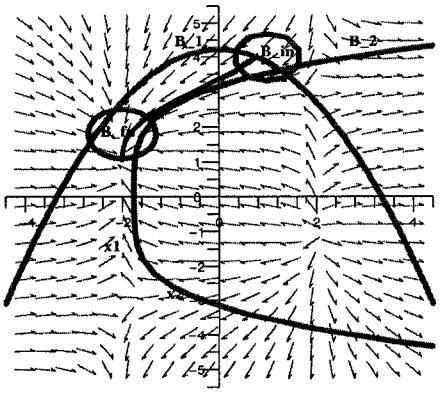


Figure 5.5: Reachability (Example 5.2.3)

will describe how to find useful predicates of the abstract states to refine the regions of interest, and to identify rules to build transitions between the abstract states. Symbolic model checking can then be applied on the constructed model.

5.3 Predicate Abstraction

5.3.1 Abstract State Space

In general, the effectiveness of the predicate abstraction method depends on the choice of predicates. In addition to using Darboux predicates as described in Section 5.1.3, we choose predicates identified in the properties of interest. In addition to temporal property predicates, basic ideas from the qualitative theory of continuous systems can be adapted within the predicate abstraction framework. The termination of the predicate generation phase is not necessary for creating an abstraction. We can stop at any point and construct the abstract model. A larger predicate set yields a finer abstraction as it results in a larger state space in the abstract model.

We define first the notion of *critical point* as follows:

Definition 5.3.1. A fixed point is a point at which the vector field vanishes. For the ODE

system $\dot{\mathbf{x}} = \mathbf{P}(\mathbf{x}(t))$, we look for solutions $\mathbf{x}(t) = \mathbf{v}, \mathbf{v} \in \mathbb{R}^n$ such that $\mathbf{P}(\mathbf{v}) = \mathbf{0}$.

A set of predicates can be constructed using the notion of *critical forms*, which are special functions along which, the vector field direction is either vertical or horizontal. In between these forms, there can be no vertical nor horizontal vectors. In a region (abstract state) determined by the critical forms, all vectors follow one direction. These predicates can be obtained easily by setting $\dot{\mathbf{x}} = 0$.

A generalization of critical forms is the concept of *isoclines*. Isoclines are functions over which the system trajectories have a constant slope.

Definition 5.3.2. A predicate π is an isocline of ODEs system if and only if $\exists a_i \in \mathbb{R}$ with $i = 1, \ldots d$ such that

$$\Sigma_{i=1}^{d} a_i \mathcal{P}_i(\mathbf{x})|_{\pi} = 0$$

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

Another kind of predicate, we propose, is referred to as a *conditioned predicate*. These predicates have the property that under specific conditions, they provide certain information about the solution flow. For instance, consider the 3-dimensional system with the state variables $x, y, z.$ and the property predicate $z > 1$. We can construct another predicate that intersects $z > 1$ at a specific condition, say $\frac{\dot{y}}{\dot{x}} = 0$. Then, the new predicate is of the form

$$\dot{y} - (z-1)\dot{x} = 0$$

These kind of predicates are important during refinement, when an abstract state needs to be subdivided into a new set of abstract states. The conditioned predicates are defined

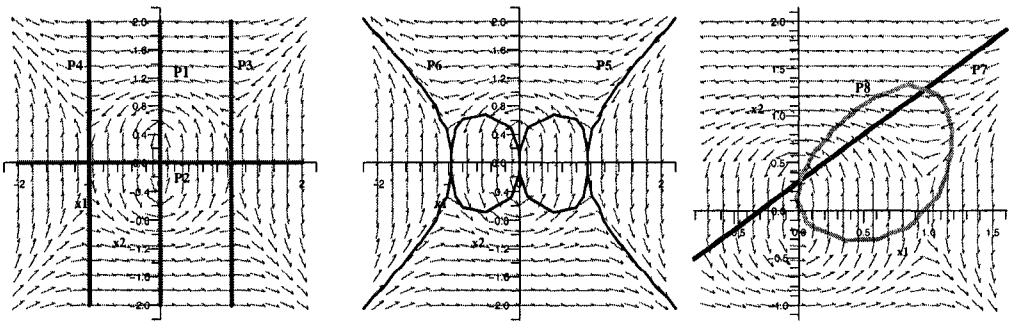
formally as follows:

Definition 5.3.3. A predicate π is a conditioned predicate of an ODEs system with conditions $\Gamma_1, \ldots, \Gamma_d$, if it is of the form

$$\Sigma_{i=1}^{n} \Gamma_i \mathcal{P}_i(\mathbf{x})|_{\pi} = 0$$

where the conditions Γ_i are polynomials with $i = 1, \ldots d$ and d is the system dimension.

Example 5.3.1. Consider the analog circuit in Example 5.1.1. The critical forms predicates are $p_1 := x_1$, $p_2 := x_2$, $p_3 := 1 - x_1$ and $p_4 := 1 + x_1$, as shown in Figure 5.6.a. For illustration purposes, we choose two isocline predicates $p_5 := x_1 - x_1^3 + x_2^3$ and $p_6 := x_1 - x_1^3 - x_2^3$ as shown in Figure 5.6.b. Suppose, we wish to verify a property including the predicate $p_7 := x_2 - x_1 > 0.3$. We can construct the conditioned predicate $p_8 := \dot{x_2} - (x_2 - x_1 - 0.3)\dot{x_1} = 0$ as shown in Figure 5.6.c. To build the abstract state space, we have three invariant regions and eight predicates. As certain combination of predicates are infeasible, the number of abstract states is $< 2^8$ abstracts states. In fact, region $R_1 = j_1 \geq 0 \wedge j_2 \geq 0$ is subdivided into 29 abstract states.



(a) Critical Forms Predicates (b) Isocline Predicates (c) Conditioned Predicates

Figure 5.6: Predicates for the Circuit in Figure 5.2.a

Other methods for finding useful predicates were developed in [106], where the authors proposed a way to extract predicates from polynomial ODEs by looking at higher

derivatives. If $p \in P$, then add $\dot{p}$, the derivative (with respect to time) of p, to the set P unless $\dot{p}$ is a constant or a constant factor multiple of some existing polynomial in P.

5.3.2 Computing Abstract Transitions

One main issue in constructing abstract state transition systems is the identification of the possible transitions. As we divide the state space into invariant regions, we need only to construct transitions between abstract states within a region. Therefore, we do not need to construct an abstract model for the whole state space. In general, information from the solution of the ODEs is required to describe transitions between abstract states. In practice, the abstract transition relation is initialized to the trivial relation relating all states and then stepwise refined by eliminating unfeasible transitions. This guarantees that any intermediate result represents an overapproximating abstraction and the refinement can be stopped at any point of time. In the remainder of this section, we use a set of different rules to construct transitions between abstract states.

The simplest rule to use is the *Hamming distance rule* [106]. The Hamming distance (HD) is the number of predicates for which the corresponding valuations are different in different abstract states. For instance, the Hamming distance between state $s_1 := (p_1 = 1 \wedge p_2 = 0 \wedge p_3 = 1 \wedge p_4 = 1)$ and state $s_2 := (p_1 = 1 \wedge p_2 = 0 \wedge p_3 = 0 \wedge p_4 = 1)$ is 1, written $HD(s_1, s_2) = 1$. Given two abstract states s_1 and s_2, we say that a transition can exist between two abstract states only if $HD(s_1, s_2) = 1$. The next rule we apply is based on the *generalized mean value theorem* [40], which is an extension of the mean value theorem (MVT) for n-dimension (See Definition 4.1.1, Chapter 4).

We use quantified constraint based methods to check whether the MVT condition is satisfied between two abstract states. If the MVT is not satisfied, we deduce that no transition exists between the two states. The above rules give an over-approximation of the transition system as no information about the vector field direction is used. In order to remove such redundant transitions in the region of interest, we complement the above rules by applying the *intermediate value theorem* (See Definition 4.1.2, Chapter 4) as a

way to identify the flow direction. In the context of abstraction, a transition between two abstract states exists if a predicate valuation changes during the execution over an interval domain. This can be checked using the intermediate value theorem.

5.3.3 Abstract Model Refinement

In general, if the abstract model is not suitable for the property analysis, then a global refinement procedure is required in order to increase the precision of the model. In fact, the refinement procedure is applied iteratively until verification reveals whether or not the property in question is satisfied. Practically, this is based on the abstract counter-example validation and refinement as explained in Section 4.3.

The main task for the counter-example validation procedure is the computation of the exact successor states starting from the initialization of the counter-example. The outcome of the procedure is either that a bad state is reached or a transition is determined to be spurious. Unfortunately, the required concretization of the given counterexample adds more trajectories that might not correspond to real ones. Therefore, only an over-approximation of the exact set of states can be defined.

The intuitive method to validate a counter-example is based on applying the bounded reachability analysis described by Algorithm 1.

Statement. Given an abstract counter-example trace $\Omega = (\sigma, \tau, \lambda)$ (See Definition 4.3.1, Chapter 4) [2] and the trace corresponding to the set of reachable states $\widehat{\Omega} = (\widehat{\sigma}, \widehat{\tau}, \widehat{\lambda})$. $\widehat{\Omega}$ is a concrete counterpart of Ω if both traces are related according to Definition 4.3.1.

Because the applied reachability analysis (using Algorithm 1) is time bounded, therefore it is not always possible to validate an abstract counter-example. In this case, a refinement procedure is required.

The reachability based validation cannot always establish the nonexistence of an abstract transition. However, we propose a practical method to remove redundant transitions by considering a transition across the boundary of two abstract states as a switching

[2] In the current definition, τ is sequence of steps $n \in \mathbb{N}$

condition problem as described in Section 5.2.2.

5.4 Applications

In Chapter 4, most of the properties we were interested in verifying were positive behaviors (e.g., something good will eventually happens like occurrence of oscillation). In this chapter we are interested in verifying safety properties (e.g., something bad will never happen such as transistor will never go to a certain mode of operation). In this respect, we apply the verification methodology proposed in this chapter to a variety of circuits including a BJT Colpitts circuit, a Tunnel diode oscillator in addition to other basic RLC circuits. Implementation details are described in Appendix A.

5.4.1 BJT Colpitts Circuit

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

Consider the BJT based Colpitts oscillator shown in Figure 5.7. Correct functionality ensures that the BJT will never go into saturation region [64]. In fact, the BJT will either be in the Cut-off mode or Forward active mode. The state space is subdivided into four regions according to the BJT modes of operations (Cut-off, Reverse active, Forward active and Saturation) with threshold voltage $Vth = 0.75$. For instance, the property that no transition can occur from Forward active (m_1) to Saturation (m_3), can be validated by proving that $\forall \mathbf{G}\ V_{C_2} < 0.75 \wedge V_{C_1} + V_{C_2} < 0$ is False, where V_{C_1} and V_{C_2} are voltages across the capacitors C_1 and C_2.

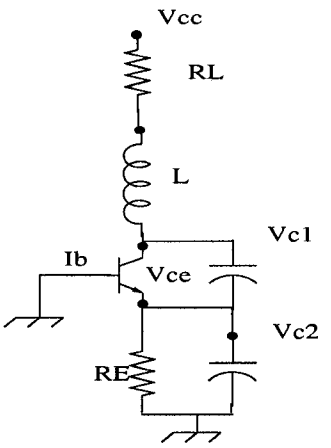


Figure 5.7: BJT Colpitts Circuit

5.4.2 Non-Linear Analog Circuit

Consider the circuit in Example 5.1.1, with initial conditions $x_1(0) \in [-0.7, -1.1]$ and $x_2(0) \in [0.5, 0.9]$. We want to verify the following ∀CTL property on the set of trajectories:

$$\forall \mathbf{F}\mathcal{P} := x_1^2 + x_2 - 3 \geq 0$$

which can be understood given the set of initial conditions, on every computation path, in the future the vector field will always cross a threshold condition. We already verified in Example 5.2.1 that this cannot happen for the initial conditions inside Region R_1, but with the invariant checking method used, we could not deduce information regarding the behaviour in region R_3. After providing the required set of predicates, we only construct corresponding abstract state transition graphs (ASTG) for regions R_1, R_3. Using the SMV model checker [22], we find that given the initial conditions such property will be indeed satisfied in region R_3.

5.4.3 RLC Circuit Oscillator

Checking for occurrence of oscillation is not always possible using predicate abstraction, due to the difficulty of generating an abstract model with no spurious transitions. In some

cases we succeeded in accomplishing the verification.

We verified the oscillation property for the circuit shown in Figure 5.8(a), with non-linear voltage source *vs* and non-linear current source *cs* described using ODEs, respectively, as follows:

$$\dot{I}_l = -V_c - \frac{1}{5}V_c^2 \; and \; \dot{V}_c = -2I_l - I_l^2 + I_l^3$$

After that we generate using Mathematica the following invariants:

$$j_1 = 1 - 5I_l^3 - 15I_l^2 + V_c^3 + \frac{15}{2}V_c^2 + \frac{15}{4}I_l^4$$

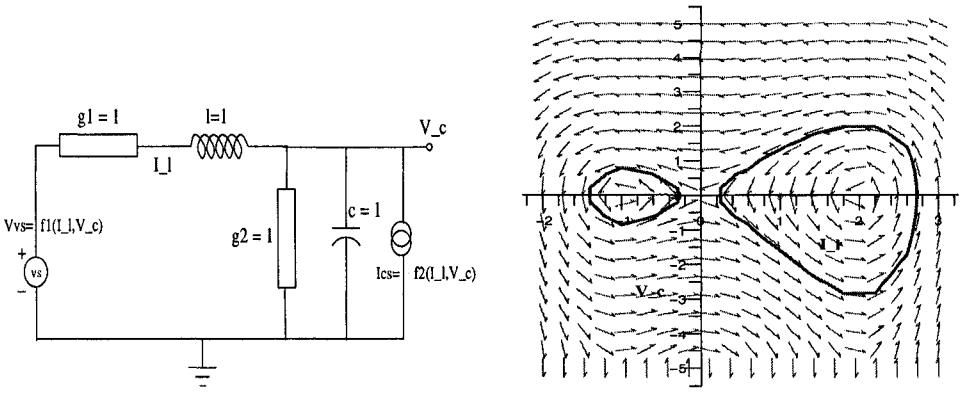
We can therefore construct two invariant regions $R1 := j_1 \leq 0$ and $R2 := j_1 > 0$. Given the state space and invariant regions as shown in Figure 5.8(b), we verify the following ∀CTL property on the set of trajectories:

$$\forall\mathbf{G}(\forall\mathbf{F}(V_c > I_l)) \wedge \forall\mathbf{G}(\forall\mathbf{F}(V_c < I_l))$$

which can be understood as on every computation path, whenever the capacitor voltage V_c value exceeds the inductor current value I_l, it will eventually decrease below I_l again and vise-versa. This property checks for oscillatory behaviour of the circuit. We constructed the abstract transition graph for each region and verified the property using SMV. We found indeed that the circuit will always oscillate only inside the bounded regions as illustrated in Figure 5.8.

5.5 Summary

In this chapter, we developed a qualitative verification approach of continuous-time AMS designs circuits. The approach is based on abstracting and verifying the qualitative behavior of the circuits using a combination of techniques from predicate abstraction and constraint solving along with model checking. The principle novelties in this work are:



(a) Circuit Schematic (b) Phase Portrait and Invariant Regions

Figure 5.8: Non-Linear Oscillator

- We adapted the concept of lazy abstraction for the verification of CT-AMS designs. To this aim, we identified a set of basic qualitative predicates (Darboux polynomials) as invariance predicates which helps avoid the construction of an abstract model for the whole state space.

- We proposed a constraint solving approach for the verification of safety and reachability properties. This method does not require explicit representation of the state space but relies on generating functions that prove or disapprove the properties.

  Our methodology overcomes the time bound limitations of exhaustive methods developed in related work.

Up till now, we addressed the verification of CT-AMS designs using a variety of model checking techniques. The remaining contribution in the thesis which will be presented in next chapter, is devoted to the verification of another important class of AMS designs, that is the discrete-time(DT) AMS.

Chapter 6

Verification of DT-AMS Designs

In this chapter, we are concerned with the class of AMS designs described using discrete-time models. This category of designs are usually developed as simulation models at a high level of abstraction in order to gain insight at the main properties of the AMS systems. In addition, discrete-time models are used to describe the behavior of switched capacitor based designs or clocked AMS designs.

In this chapter, we define a bounded model checking algorithm on the SRE model by means of an algebraic computation theory based on Interval Arithmetics [85]. We associate the bounded model checking with a powerful and fully decidable equational theorem proving method to verify properties for unbounded time using induction. We applied the verification on several AMS designs including $\Delta\Sigma$ modulators and switched capacitor circuits.

Our methodology aims to prove that an AMS description satisfies a set of properties. This is achieved in two phases: modeling and verification, as shown in Figure 6.1.

Starting with an AMS description and a set of properties, the symbolic simulator performs a set of transformations using rewrite rules in order to obtain the generalized system of recurrence equations (SREs). These are combined recurrence relations that describe each property blended directly with the behavior of the system. The next step is to prove these properties using an algebraic verification engine that combines Bounded

Model Checking over Interval Arithmetic [85] and induction over the normal structure of the generalized recurrence equations.

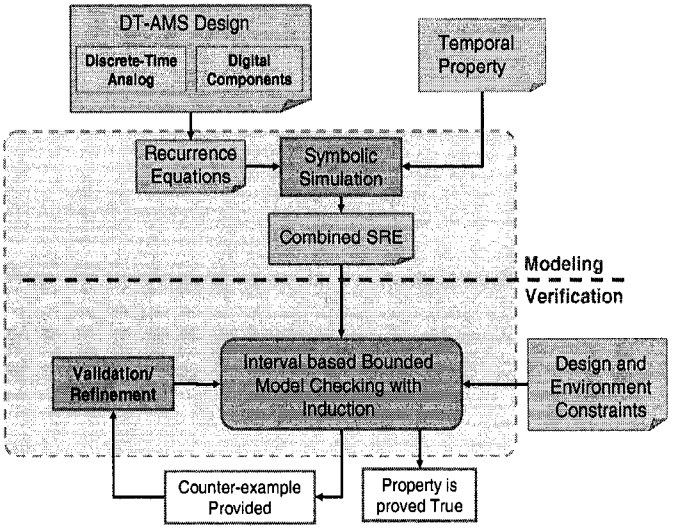


Figure 6.1: DT-AMS Verification Methodology

In summary, the verification loop terminates in one of the following situations:

- Complete verification:
  - The property is proved by induction for all future states.
  - The property is false and a concrete counterexample is found.
- Bounded Verification:
  - The resource limits have been attained (memory or CPU) as the verification can grow exponentially with the number of reachability analysis steps.
  - The constraints extracted from the interval states are divergent with respect to some pre-specified criteria (e.g., width of computed interval states).

In the following, we will describe the two main verification engines we propose, namely bounded model checking using interval arithmetics and inductive verification.

We will also provide an algorithmic view of how to combine both of them together as proposed in our methodology.

6.1 The Verification Algorithms

6.1.1 Interval based BMC

Interval arithmetics based algorithms are an attractive tool to use in the verification of the behavior of systems with uncertainty on the design parameters or initial conditions. Interval arithmetics as explained before provide an overapproximation of the possible reachable states of the system, hence guaranteeing the soundness of the verification results. In this section, we propose a BMC verification algorithm for DT-AMS design. The algorithm is based on modeling the transition relation as an SRE and modeling the state space using intervals. The recurrence model makes it possible to handle continuous behaviors like those of current and voltages, but in discrete time, which cover a non-trivial class of mixed behaviors. The basics of BMC have already been discussed in Chapter 4, Section 4.2. In the following, we will introduce the verification algorithm[1].

The image computation is the set of states reachable during one execution step.

Definition 6.1.1. ***Image Computation.***

The set of reachable states in one step from a given set of states $S_k \subseteq \mathbb{I}^d$, is denoted by $\mathcal{R}_{\mathbb{I}}(S_k)$ and is defined as:

$$\mathcal{R}_{\mathbb{I}}(S_k) \triangleq \{s' \in S_{k+1} | \exists s \in S_k : \overrightarrow{F}(s) = s'\}$$

where $S_{k+1} \subseteq \mathbb{I}^d$, $\overrightarrow{F} = (F_1, \ldots, F_d)$ and $F_i : \mathbb{I}^d \rightarrow \mathbb{I}$ is an interval evaluation of the *if-formula* $f_i : \mathbb{R}^d \rightarrow \mathbb{R}, i \in \{1, \ldots, d\}$.

[1]For compactness purposes, in the remaining of the chapter, we will deal with properties of the form $\mathbf{G}p(k)$. Verifying properties of the form $\mathbf{F}p(k)$ can be easily derived. This is due to the duality of the **G** and **F** operators [23].

The bounded forward reachability algorithm starts at the initial states and at each step computes the image, which is the set of reachable interval states. This procedure is continued until either the property is falsified in some state or no new states are encountered. We evaluate the reachable states over interval domains, at arbitrary time steps, according to the following definition:

Definition 6.1.2. The set of reachable states in less than k steps $(0 < l < k)$, from a given set S_0 of states, is denoted by $\mathcal{R}^{<k}(S_0)$, and is defined as:

$$\mathcal{R}^{<k}(S_0) \triangleq \bigcup_{l<k} \mathcal{R}_1^l(S_{l-1})$$

The bounded model checking over interval domains is then defined as follows:

Definition 6.1.3. ***Interval based Bounded Model Checking.***
Given a natural number $k \geq 0$, an interval based state machine $\mathcal{T}_I = (S_I, S_{I,0}, \rightarrow_{\delta_I})$ (See Definition 3.2.11, Chapter 3), and a property *Prop*, we say that *Prop* is verified for k steps if:

$$\forall s \in \mathcal{R}^k(S_0) : s \models Prop$$

where S_0 is the set of initial states and $\mathcal{R}^k(S_0)$ is the set of states reachable from S_0 in k steps.

The verification steps for safety properties are shown in Algorithm 8. The AMS modeling described as a set of recurrence equations is provided along with the (negated) property $\neg Prop[n]$ under verification. Initial and environment constraints *Env_Const* are also defined prior to the verification procedure described in lines (1-12) as a loop for N_{max} time steps. At each step n, we check whether the property is satisfied or not (Line 2). If $\neg Prop[n]$ is satisfied then a counterexample is generated (Line 9), if not, then we check if fixpoint inclusion is reached (Line 3), otherwise, we update the reachable states (Line 11) and go to the next time step of the verification. The functions *Prop_Check*, *Find_Counterexample* and *Update_Reach* are described below.

Algorithm 8 Safety BMC

Require: $x[n]$
Require: $\neg Prop(x[n])$
Require: $\mathcal{R}^0 = S_0$
Require: Env_Const
1: **for** $n = 1$ to N_{max} **do**
2: **if** $Prop\_Check(\neg Prop[n], x[n]) == False$ **then**
3: **if** $Reach[T_{o_t, x[n]}] \subseteq \mathcal{R}^{n-1}$ **then**
4: **return** fixpoint reached
5: **else**
6: $Inc\_Step(n)$
7: $\mathcal{R}^{n-1} = Update\_Reach(\mathcal{R}^{n-2}, Reach[x[n-1]])$
8: **end if**
9: **else**
10: $Find\_Counterexample(\neg Prop[n], x[n], Env\_Const)$
11: **end if**
12: **end for**

Prop_Check: Given the property $\neg Prop()$, apply algebraic decision procedures to check for satisfiability. The safety verification at a given step n can be defined with the following formula:

$$Prop\_Check \triangleq \mathbf{x}[n] = f(\mathbf{x}[n-1]) \wedge \neg Prop(\mathbf{x}[n]) \wedge x[n-1] \in \mathbb{I}^d$$

Practically, this can be done using equational theorem proving capabilities as will described in Appendix A.

Update_Reach(R_1, R_2): This function returns the union of the states in the sets R_1 and R_2.

Reach[$x[n]$] evaluates the reachable states over interval domains at an arbitrary time step.

Find_Counterexample($\neg Prop[n], x[n], Env\_Const$): This function returns a counterexample indicating a violation of the property within the environment constraints (cf. Appendix A).

Setting bounds on the maximum number of iterations ensures that the algorithm will eventually terminate with one of the following possibilities. If at a given time step $n \leq N_{max}$, no new interval states are explored, then fixpoint inclusion guarantees that the property will be always satisfied; otherwise, if the property is proved to be incorrect, then a counterexample is generated. If we reach the maximum number of steps $n = N_{max}$, and no counterexample is generated, then the property is verified up to bounded step N_{max}.

Example 6.1.1. Given the $\Delta\Sigma$ design and the safety property in Example 3.4.1, we apply Algorithm 8. For instance, the correctness of the property $P(k+1)$ depends on the parameters A, B and C shown in Figure 3.5, the values of variables $x_1(k)$, $x_2(k)$ and $x_3(k)$, the time k, and the input signal $u(k)$ (See Table 6.1). Using an implementation of the Algorithm 8 in Mathematica, we verify the $\Delta\Sigma$ modulator for the following set of parameters inspired from the analysis in [50]:

$$\begin{cases} a = 1 & a_1 = 0.044 & a_2 = 0.2881 & a_3 = 0.7997 \\ b_1 = 0.07333 & b_2 = 0.2881 & b_3 = 0.7997 & \\ c_1 = 1 & c_2 = 1 & c_3 = 1 & \end{cases}$$

The initial constraints define the set of test cases over which interval based simulation is applied. If the property is *false*, as in the first and third cases in Table 6.1, then the verification is completed and a counterexample is generated from the simulated intervals. On the contrary, when the property is *True*, we have a partial verification result as it is bounded in terms of simulation steps. The second case in Table 6.1 illustrates this limitation. Counter-examples on the third column are generated using the function $Find\_Counterexample(.)$.

Unfortunately, we note that in some cases (last row in Table 6.1), divergence happens quickly, so we cannot deduce useful information on the property. We tackle this problem by extending the bounded model checking with an induction engine as proposed in the methodology.

Table 6.1: Interval Based BMC Verification Results for ΔΣ Modulator in Example 6.1.1

Initial Constraints	**Property Evaluation for $n = 0$ to N_{max} Cycles**	**CPU Time Used Counter-Example**
$0.028 \leq x_1(0) \leq 0.03$ $-0.03 \leq x_2(0) \leq -0.02$ $0.8 \leq x_3(0) \leq 0.82, u := 0.8$	$N_{max} = 40$ $n = 0$ to 15 True $n > 15$ False	1.5 sec $x_1[16] \mapsto 0.263$ $x_2[16] \mapsto 1.25$, $x_3[16] \mapsto 2.42$
$0.012 \leq x_1(0) \leq 0.013$ $0.01 \leq x_2(0) \leq 0.02$ $0.8 \leq x_3(0) \leq 0.82, u := 0.54$	$N_{max} = 38$ True	31 sec
$0.163 \leq x_1(0) \leq 0.164$ $-0.022 \leq x_2(0) \leq -0.021$ $0.8 \leq x_3(0) \leq 0.82, u := 0.6$	$N_{max} = 40$ $n = 0$ to 17 True $n > 17$ False	0.8 sec $x_1[19] \mapsto 0.163$ $x_2[19] \mapsto 0.88, x_3[19] \mapsto 2.47$
$0.012 \leq x_1(0) \leq 0.013$ $0.01 \leq x_2(0) \leq 0.02$ $0.8 \leq x_3(0) \leq 0.82, 0.58 \leq u \leq 0.6$	Divergent at Timestep 4	0.5 sec

6.1.2 Constrained Induction based Verification

In formal verification, induction has been used to prove a property P in a transition system by showing that P holds in the initial states of the system and that P is maintained by the transition relation of the system. In the following, we will define an induction engine over SREs for the safety property verification of AMS designs. The inductive proof for verifying a safety property $P(n) = Gp(n)$ can be derived by checking the formula $Ind_{proof} \triangleq \psi_{base} \wedge \psi_{induc}$, where ψ_{base} is the induction base and ψ_{induc} is the induction step defined as follows:

$$\psi_{base} \triangleq \forall s \in S_0 : I(s_0) \Rightarrow p(s_0)$$

and

$$\psi_{induc} \triangleq \forall s_k, s_{k+1} \in S : \mathcal{T}(s_k, s_{k+1}) \wedge p(s_k) \Rightarrow p(s_{k+1})$$

The core of the induction engine is a decision procedure function that checks satisfiability of algebraic formulas under certain constraints on quantified state variables.

Definition 6.1.4. ***The Prove Function.***

$Prove(quant(X, cond, expr)) =$

$If(Prop\_Verify(quant(X, cond, expr))) = True,$

$True,$

$Find\_Counterexample(cond \wedge \neg expr)$

The decision procedure function *Prove* tries to prove a property of the form $quant(X, cond, expr)$, using $Prop\_Verify$, otherwise it gives a counterexample using the function $Find\_Counterexample$, where $quant \in \{\forall, \exists\}$ define quantifiers over a set of state variables x, *cond* is a logical combination of comparison formulae constructed over the variables x describing initial and environment constraints and *expr* is an *If-formula* expression representing the property of interest, obtained after applying the symbolic rule outlined earlier.

Similar to *Prop_Check*, $Prop\_Verify$ applies algebraic decision procedures to check for satisfiability, but for all time steps n. The safety verification can be defined with the following formula:

$$Prop\_Verify \triangleq \forall n.(\mathbf{x}[n] = SRE(x[n])) \wedge Prop(\mathbf{x}[n])$$

The *Prove* Function uses $Find\_Counterexample(cond \wedge \neg expr)$ to generate a counterexample if the property of interest cannot be proved to hold. If a proof cannot be obtained, then we may need to find a particular combination of inputs and local signals values for which the property is not satisfied.

The properties verification using *Prove* starts by checking the validity at time $t = 1$, then at time $t = n$ assuming the properties are satisfied at time $t = n - 1$. Case splitting divides the property into subproperties for which validation results are conjuncted to check the validation of the original property.

Let P be a property of the form $quant(X, cond, expr)$. We define the function *Split-Prove* that depending on the *If-formula* structure of *expr*, applies the function *Prove* or

splits the verification. *SplitProve* is defined recursively as follows:

Definition 6.1.5. ***The SplitProve Function.***

According to the nature of *expr*, *SplitProve* can be one of the following:

- *expr* is a comparison formula C, $SplitProve(quant(X, cond, C)) =$
$$Prove(quant(X, cond, C))$$

- *expr* is a logical formula of the form $a \diamond b$, with $\diamond \in \{\neg, \wedge, \vee, \oplus, \ldots\}$ and a, b are *If-formulae* that take values in $\mathbb{B}$

$SplitProve(P)) \leq$
$$SplitProve(quant(X, cond, a))$$
$$\diamond$$
$$SplitProve(quant(X, cond, b))$$

- *expr* is an expression of the form $IF(q, l, r)$ $SplitProve(P) =$
$$SplitProve(quant(X, cond \wedge q, l))$$
$$\vee$$
$$SplitProve(quant(X, cond \wedge \neg q, r))$$

According to algebraic laws of the quantifiers, we have the following four cases:

- For $a \wedge b$ and $quant := \exists$

$SplitProve(P)) \Rightarrow$
$$SplitProve(\exists(X, cond, a))$$
$$\wedge$$
$$SplitProve(\exists(X, cond, b))$$

- For $a \wedge b$ and $quant := \forall$

$SplitProve(P)) \Leftrightarrow$

$$SplitProve(\forall(X, cond, a))$$
$$\wedge$$
$$SplitProve(\forall(X, cond, b))$$

- For $a \vee b$ and $quant := \exists$

  $SplitProve(P)) \Leftrightarrow$

  $$SplitProve(\exists(X, cond, a))$$
  $$\vee$$
  $$SplitProve(\exists(X, cond, b))$$

- For $a \vee b$ and $quant := \forall$

  $SplitProve(P)) \Rightarrow$

  $$SplitProve(\forall(X, cond, a))$$
  $$\vee$$
  $$SplitProve(\forall(X, cond, b))$$

Let $P(n)$ be the recurrence equation of the property P written as an *If-formula*. Let $cond_{n_0}$ be the initial condition at time n_0, $cond_n$ the constraints that are true for all $n > n_0$, and X the set of dependency variables of $P(n)$. The proof by induction over n is defined as follows:

Definition 6.1.6. ***Proof by Induction.***

$SplitProve(ForAll(X_{n_0}, cond_{n_0}, P(n_0)))$

$\wedge$

$SplitProve(ForAll(n > n_0 \wedge X_n, n \in \mathbb{N} \wedge cond_n \wedge P(n), P(n+1)))$

Example 6.1.2. We verify the $\Delta\Sigma$ modulator in Example 3.4.1 for two sets of parameters inspired from the analysis in [50]:

$$Param_1 \rightarrow \begin{cases} a=1 & a_1=0.044 & a_2=0.2881 & a_3=0.7997 \\ b_1=0.044 & b_2=0.2881 & b_3=0.7997 & \\ c_1=1 & c_2=1 & c_3=1 & \end{cases}$$

$$Param_2 \rightarrow \begin{cases} a=1 & a_1=0.044 & a_2=0.2881 & a_3=0.7997 \\ b_1=0.07333 & b_2=0.2881 & b_3=0.7997 & \\ c_1=1 & c_2=1 & c_3=1 & \end{cases}$$

We apply the induction implemented in Mathematica, in order to verify the $\Delta\Sigma$ modulator stability for the above sets of parameters and for two cases of conditions (state space constraints). Table 6.2 summarizes the verification results. The property is *True* if it is proved under the set of conditions and the set of parameters for all $k > 0$. If there is no k for which the property is valid then it is *False*, and a counterexample is provided. When the property is valid for some values of k and not for other values, we say that the property is not proved and counterexamples are provided.

6.2 d-Induction BMC Methodology

The proposed verification algorithm is based on combining induction and bounded model checking to generate a correctness proof for the system. This method is an algebraic version of the induction based bounded model checking developed recently for the verification of digital designs [6]. We start with an initial set of states encoded as intervals as shown in Figure 6.2. Then iteratively the possible reachable successor states from the previous states are evaluated using interval analysis based computation rules over the SREs, i.e., the output of this step is an *If-formula* where all variables are substituted by intervals. If there exits a path that evaluates the property to false, then we search for a

Table 6.2: Induction based Verification Results for $\Delta\Sigma$ modulator in Example 6.1.2

	State Space Constraints	Property with Param_1	Property with Param_2
case1	Values at t=0	True	True
	$0 \leq x_1(0) \leq 0.01$ $-0.01 \leq x_2(0) \leq 0$ $0.8 \leq x_3(0) \leq 0.82, u := 0.6$		
	Values at t=n		
	$-0.1 \leq x_1(n) \leq 0.1$ $-0.5 \leq x_2(n) \leq 0.5$ $0.5 \leq x_3(n) \leq 1.5, u := 0.6$		
case2	Values at t=0	False	False
	$0 \leq x_1(0) \leq 0.02$ $-0.03 \leq x_2(0) \leq -0.01$ $1 \leq x_3(0) \leq 1.4, u := 0.8$		
	Values at t=n		
	$-0.1 \leq x_1(n) \leq 0.1$ $-1 \leq x_2(n) \leq 0.5$ $-1 \leq x_3(n) \leq 2.5, u := 0.8$	$x_2[k] \mapsto 0.4237$ $x_3[k] \mapsto 1.8378$	$x_2[k] \mapsto 0.2103$ $x_3[k] \mapsto 2$

concrete counterexample. Otherwise, if all paths give true, then we transform the set of current states to constraints and we try to prove by induction that the property holds for all future states. If a proof is obtained, then the property is verified. Otherwise, if the proof fails then, the BMC step is incremented; we compute the next set of interval states and the operations are re-executed.

6.2.1 d-induction

In formal verification, induction has been used to prove a property $\mathbf{G}P(n)$ in a transition system by showing that P holds in the initial states of the system and that P is maintained by the transition relation of the system. As such, the induction hypotheses are typically much simpler than a full reachable state description. Besides being a complete proof technique, when it succeeds, induction is able to handle larger models than bounded model checking, since the induction step has to consider only paths of length 1, whereas

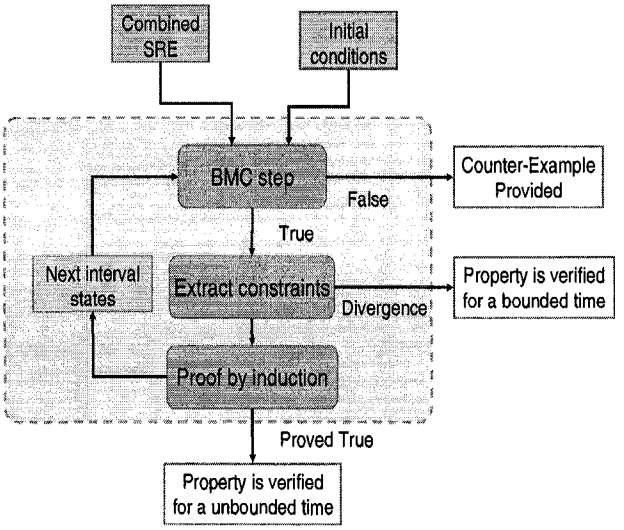


Figure 6.2: Overview of the Verification Algorithm

bounded model checking needs to check sufficiently long paths to get a reasonable confidence. Hence, simple induction is not powerful enough to verify many properties.

d-induction [6] is a modified induction technique, where one attempts to prove that a property holds in the current state, assuming that it holds in the previous d consecutive states. Essentially, induction with depth corresponds to strengthening the induction hypothesis, by imposing the original induction hypothesis on d consecutive time-frames. Given a state transition system $(S, I, \mathcal{T})$, where S is the set of states, $I \subseteq S$ is the set of initial states, $\mathcal{T} \subseteq S \times S$, the d-induction proof is defined as

$$d - Ind_{proof} \triangleq \psi_{d-base} \wedge \psi_{d-induc}$$

where ψ_{d-base} is the induction base and $\psi_{d-induc}$ is the induction step defined as follows:

$$\psi_{base} \triangleq I(s_0) \wedge \bigwedge_{i=0}^{d-1} \mathcal{T}(s_i, s_{i+1}) \Rightarrow \bigwedge_{i=0}^{d} p(s_i)$$

and

$$\Psi_{d-induc} \triangleq \bigwedge_{i=k}^{k+d} \mathcal{T}(s_i, s_{i+1}) \wedge \bigwedge_{i=k}^{k+d} p(s_i) \Rightarrow p(s_{k+d+1})$$

It is worth noting that when $d = 1$, we have exactly the basic induction steps defined in classical induction.

Similar to the general induction methods, (un)satisfiability based induction $d - Ind_{sat}$ is the dual of the induction proof $\neg Ind_{sat} = d - Ind_{proof}$. Checking the formula $d - Ind_{sat} \triangleq \phi_{d-base} \vee \phi_{d-induc}$ for unsatisfiability, where the formulas ϕ_{d-base} (the base step) and $\phi_{d-induc}$ (the induction step) are defined as follows:

$$\phi_{d-base} \triangleq I(s_0) \wedge \bigwedge_{i=0}^{d-1} \mathcal{T}(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{d} \neg p(s_i)$$

and

$$\phi_{d-induc} \triangleq \bigwedge_{i=k}^{k+d} \mathcal{T}(s_i, s_{i+1}) \wedge \bigwedge_{i=k}^{k+d} p(s_i) \wedge \neg p(s_{k+d+1})$$

The d-induction based verification (Algorithm 9 as in [6]) is an incremental algorithm, where the depth bound d (Line 10) is incremented at each step and induction (Lines 3, 6) is applied on the new formulas until a d-length counterexample is generated (Line 4) or the property is proved over a suitable length (Line 7).

Algorithm 9 d-induction based procedure [6]

```
1:  initialize d = 0
2:  for d = 0 to d_max do
3:    if φ_{d-base} is True then
4:      return counterexample
5:    else
6:      if φ_{d-induc} is False then
7:        return verified
8:      end if
9:    end if
10:   d = d + 1
11: end for
```

The advantage of d-induction over classical induction is that it provides the user with ways of strengthening the induction hypothesis by lengthening the time steps d computed. Practically speaking, ϕ_{d-base} is a bounded model checking ($\overline{BMC}$) as defined earlier in this section. For the case of systems with variables interpreted over real domains like AMS designs, the satisfiability of the formulae with a given set of initial conditions, requires algorithms to produce bounded envelopes for all the reachable states at the discrete time points.

6.2.2 Combining d-induction and Interval based BMC

The d-induction based verification algorithm is an incremental algorithm, where depth is incremented at each step and induction is applied on the new formulas until a d-length counterexample is generated or the property is proved. The verification steps are given in Algorithm 10.

The AMS model described as a set of recurrence equations is provided along with the (negated) property $\neg Prop[n]$ under verification. Initial and environment constraints are also defined prior to the the verification procedure described in lines (1-18) as a loop of depth N_{max} steps. For each depth $d < N_{max}$, we first check the initial d-induction step by verifying whether the property is verified for all steps up to this depth d (Line 3). If the property is false, we generate a counterexample (Line 4). Before checking the induction step (Line 10), we verify whether an inclusion fixed point is reached. If so, the verification ends as it will be trivial to check for the induction step as no new verification information can be implied. When we apply the induction step, either the property is verified for unbounded time (Line 11), otherwise, we conclude that the current depth is not enough to verify the property and the depth in incremented (Line 14).

It is worth noting, that constraints used in the induction steps are extracted from the previous reachable states. Hence, we strengthen the induction hypothesis by lengthening the time steps d computed. In case a counterexample needs to be generated, the extracted

Algorithm 10 d-induction based BMC

Require: $x[n] := SRE(\mathcal{A})$
Require: $\neg Prop(x[n])$
Require: $\mathcal{R}^0 = S_0$
Require: Env_Const
1: initialize $d = 1$
2: **for** $d = 1$ to N_{max} **do**
3: **if** $Prop\_Check(\neg \bigwedge_{i=0}^{d} Prop[i], x[n]) == True$ **then**
4: $Find\_Counterexample(\neg Prop[n], x[n], Env\_Const)$
5: **else**
6: **if** $Prop\_Check(\neg Prop[d], x[d]) == False$ **then**
7: **if** $Reach[x[d]] \subseteq \mathcal{R}^{d-1}$ **then**
8: **return** fixpoint reached
9: **else**
10: **if** $Prop\_Verify(\neg \bigwedge_{i=n}^{d+n} Prop[i], \bigwedge_{i=n}^{d+n} x[i]) == False$ **then**
11: return verified
12: **end if**
13: **end if**
14: $Inc\_Step(d)$
15: $\mathcal{R}^{n-1} = Update\_Reach(\mathcal{R}^{n-2}, Reach[x[n-1]])$
16: **end if**
17: **end if**
18: **end for**

constraints allow for finding a partial path violating the property.

Setting bounds on the maximum number of iterations ensures that Algorithm 10 will eventually terminate in one of the following possibilities. If the initial induction step fails, a counterexample is generated; otherwise if at a given time step $n \leq N_{max}$, no new interval states are explored, then fixpoint inclusion guarantees that the property will be always verified. In this case, the induction step is verified as true, and the algorithm terminates. Otherwise we increase the induction depth and restart the verification. If we reach the maximum number of steps $n = N_{max}$, and no counterexample is generated, then the property is verified up to bounded step N_{max}.

6.3 Applications

We have applied the verification methodology proposed in this chapter to different classes of DT-AMS designs spanning various design levels, e.g., $\Delta\Sigma$ modulator at the functional level, digitally controlled analog computers at the macromodel level, and switched capacitor designs at the circuit level.

We implemented the algorithms described in this Chapter in Mathematica. As an input to the algorithms, we supply the recurrence equations and the initialization constraints (plus environment constraints for the induction method). The output is either a message signaling that verification succeeds, divergence occurs (only in BMC or D-induction verification) or a counter-example is provided.

6.3.1 Third-order $\Delta\Sigma$ Modulator

We extended the verification results outlined throughout the chapter and summarized in Tables 6.1 and 6.2 by applying the d-induction algorithm to verify the stability of the third-order $\Delta\Sigma$ modulator for different combinations of design parameters, inputs and initial conditions. Using a Mathematica implementation for Algorithm 10, we were able to prove

properties using the inductive BMC method, that we were unable to verify perviously using the conventional BMC method. In row 2 (Table 6.1), we were able only to verify the property for a bounded time step, with the d-induction BMC method, however, we were able to prove that the property will always hold (second row with $param_2$ in Table 6.3). On the other hand, in row 4 (Table 6.1), the divergence occurs quickly, however, the property is proven *True* as shown in Table 6.3, row 4 with $param_2$. On the other hand, when comparing the d-induction verification results with the induction based verification results in Table 6.2, we get the expected results with the exception of Table 6.3, row 2 with $param_1$. The verification in Table 6.2 (Case2 with $param_1$) identifies a counter-example, while in Table 6.3, we were unable to complete the verification because of divergence of the interval calculations. The fact that simple induction was successful was due to an appropriate choice of environment constraints which are supplied manually, unlike in d-induction, where the constraints are extracted automatically from previous verification steps. A better implementation of interval arithmetics would allow therefore an enhancement in the verification results.

6.3.2 Non-Linear Voltage Switching Circuit

We studied the applicability of our methodology to the verification of a simple non-linear analog computer constructed from different components like opamp and voltage multipliers (Figure 6.3). For instance, a voltage multiplier is a non-linear analog system, which can be constructed using voltage controlled current sources like transconductance as shown in Figure 6.3.b followed by current to voltage converters. The design under verification is shown in Figure 6.3.a. We propose a circuit where the positive and negative feedbacks are externally controlled digitally, hence providing different configurations of the circuit. The circuit extends the design in [38] by adding a positive feedback section and supporting voltage multiplication making the circuit verification more challenging to

Table 6.3: d-induction BMC Verification Results for $\Delta\Sigma$ modulator

	State Space Constraints	**Verification Results**	**Verification Details**
Design	Third order $\Delta\Sigma$ modulator		
$Param_1$	$0 \leq x_1(0) \leq 0.01$ $-0.01 \leq x_2(0) \leq 0$ $0.8 \leq x_3(0) \leq 0.82, u := 0.6$	Proved True by d-induction	k-step= 3
	$0 \leq x_1(0) \leq 0.02$ $-0.03 \leq x_2(0) \leq -0.01$ $1 \leq x_3(0) \leq 1.4, u := 0.8$	Proved True by BMC then divergent	k-step= 14
$Param_2$	$0 \leq x_1(0) \leq 0.01$ $-0.01 \leq x_2(0) \leq 0$ $0.8 \leq x_3(0) \leq 0.82, u := 0.6$	Proved True by d-induction	k-step= 3
	$0.012 \leq x_1(0) \leq 0.013$ $0.01 \leq x_2(0) \leq 0.02$ $0.8 \leq x_3(0) \leq 0.82, u := 0.54$	Proved True by d-induction	k-step= 3
	$0 \leq x_1(0) \leq 0.02$ $-0.03 \leq x_2(0) \leq -0.01$ $1 \leq x_3(0) \leq 1.4, u := 0.8$	Proved False by Counterexample	k-step= 16
	$0.012 \leq x_1(0) \leq 0.013$ $0.01 \leq x_2(0) \leq 0.02$ $0.8 \leq x_3(0) \leq 0.82, 0.58 \leq u \leq 0.6$	Proved True by d-induction	k-step= 3

achieve. The circuit SRE is described as follows:

$$
\begin{aligned}
v2[n+1] &= if[vd[n], times[v1[n], v0[n]], times[-2, v1[n]]] \\
vin[n+1] &= divide[(times[r1[n], v2[n+1]]), (plus[1000, r1[n]])] \\
v0[n+1] &= divide[(times[vin[n+1], (plus[r2[n], 1000])]), r2[n]]] \\
r1[n+1] &= if[rd1[n], a, b] \\
r2[n+1] &= if[rd2[n], c, d]
\end{aligned}
$$

where $v1[n]$ is the input signal, a, b, c, d are different resistors values, chosen according to the logical conditions $rd1[n]$ and $rd1[n]$, which can be specified using a controller. Suppose we want to check the bounds on output voltage amplitude. We need to make sure

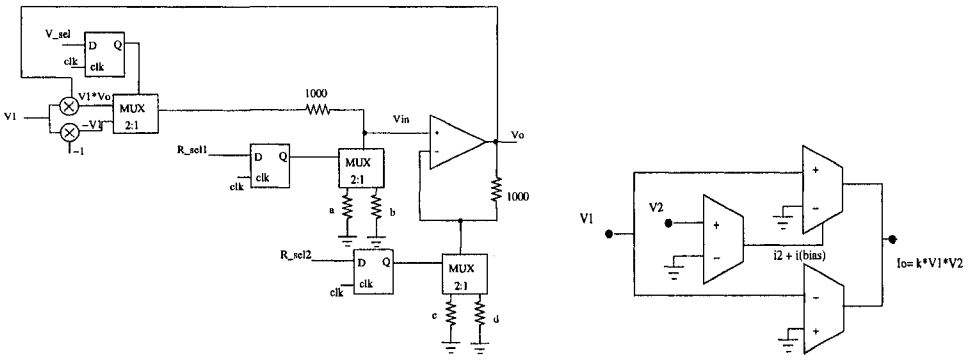
that a correct controller will ensure that the output voltage will never increase infinitely and will always be within certain range. This can be written as:

$$\mathbf{G}(P(k) = -5 \leq Vo(k) \leq 5)$$

After symbolic simulation, we obtain the following SREs.

$$-5 \leq \text{if}[\text{vd}(n), \frac{\frac{\text{r1}(n)\text{r2}(n)\text{v0}(n)\text{v1}(n)}{\text{r1}(n)+1000} + \frac{1000\text{r1}(n)\text{v0}(n)\text{v1}(n)}{\text{r1}(n)+1000}}{\text{r2}(n)}, \frac{-\frac{2\text{r1}(n)\text{r2}(n)\text{v1}(n)}{\text{r1}(n)+1000} - \frac{2000\text{r1}(n)\text{v1}(n)}{\text{r1}(n)+1000}}{\text{r2}(n)}] \leq 5$$

We choose several selector control frequencies to control the resistor as well as the input signal. The verification results for a different set of variable resistors ($\{250, 500, 1000, 2000\}$), initial values and inputs are shown in Table 6.4.



(a). Main Circuit (b). Voltage Multiplier

Figure 6.3: Digitally Controlled Analog Computer

6.3.3 Discussions

In this section, we highlighted some experimental studies we conducted on different classes of AMS designs that can be described using the SRE model proposed in this thesis. From the experimental results, we observed that the choice of the initial intervals

Table 6.4: d-induction BMC Verification Results for Analog Computer

	State Space Constraints	**Verification Results**	**Verification Details**
Design	Digitally Control Analog Computer		
Parm-s1	$-0.5 \leq v1(0) \leq 1.5$ $0.02 \leq v0(0) \leq 2.21$ $0.1 \leq v2(0) \leq 0.2$, $a,b,c,d \in \{500,2000\}$	Proved False by Counterexample	k-step= 10
Parm-s2	$-0.5 \leq v1(0) \leq 1.5$ $0.02 \leq v0(0) \leq 2.21$ $0.1 \leq v2(0) \leq 0.2$, $a,b,c,d \in \{500,2000\}$	Proved False by Counterexample	k-step= 2
Parm-s3	$-0.5 \leq v1(0) \leq 1.5$ $0.02 \leq v0(0) \leq 0.21$ $0.1 \leq v2(0) \leq 0.2$, $a,b,c,d \in \{500,2000\}$	Proved True by BMC then divergent	k-step= 22
	$-0.5 \leq v1(0) \leq 1.5$ $0.02 \leq v0(0) \leq 0.21$ $0.1 \leq v2(0) \leq 0.2$ $a,b,c,d \in \{250,1000,500,2000\}$	Proved True by d-induction	k-step= 3
Parm-s4	$-0.5 \leq v1(0) \leq 1.5$ $0.02 \leq v0(0) \leq 0.21$ $0.1 \leq v2(0) \leq 0.2$, $a,b,c,d \in \{500,2000\}$	Proved True by d-induction	k-step= 3
	$-0.5 \leq v1(0) \leq 1.5$ $0.02 \leq v0(0) \leq 0.21$ $0.1 \leq v2(0) \leq 0.2$ $a,b,c,d \in \{250,1000,500,2000\}$	Proved True by d-induction	k-step= 3

for the parameters and the state variables affect greatly the divergence, rather than the size of the designs (number of equations). This is due to the over-approximation nature of the interval arithmetics. We have used some simplification rules such as the Horner rule in order to have a better narrow bound for the reachable states.

6.4 Summary

In this chapter, we have defined and implemented an induction based bounded model checking technique that traverses the structure of the normalized properties and provides a formal correctness proof or a counterexample, otherwise. Image computations have

been achieved using interval arithmetics over these symbolic expressions. We have implemented our methodology using standard libraries for symbolic computation available in Mathematica, allowing the development of a fully automated verification engine. Experimental results have shown the feasibility of the approach. To the best of our knowledge, this is the first proposal for a d-induction approach for the verification of analog and mixed signals designs.

Chapter 7

Conclusion

The need for formal verification methods in the design flow of embedded systems is becoming more of a requirement rather than a luxury. That was motivated by the previous successes in the verification of corner cases in digital designs and the tight time-to-market constraints. In fact, the verification of AMS designs is a great challenge because of two main obstacles: infinite continuous state space and the density of the time space. In this thesis, we have presented a formal verification methodology that addresses both obstacle.

We proposed a recurrence equation (SRE) modeling framework for AMS designs based on the concept of *generalized If-formula*. Such model is adequate to describe the designs at several levels of abstraction and well suited for symbolic analysis in addition to formal verification. In fact, generalized system of recurrence equations (SREs)are a mathematical model that can represent both the digital behavior using *If-formulae* and the analog continuous state space using symbolic algebra. The symbolic computation algorithm produces a set of recurrence relations for each property that we wish to verify. For discrete-time systems, the design equations can be directly expressed by the SRE; while for continuous-time systems, a Taylor polynomials based approximation is applied with the necessary conditions to ensure preservation of the original behavior of the design.

For the verification, we developed bounded model checking algorithms for continuous-time AMS designs. We have proposed a semi-symbolic modeling of the state space using

the principle of Taylor models which provide a way for representing a combination of representations using a combination of polynomials and interval terms. The main advantage of such modeling is the fact, that the polynomial representation helps slowing the divergence due to the over-approximated intervals. Moreover, the interval part provides an important abstraction to handle the continuous behavior.

To overcome the time bound limitations of exhaustive methods associated with the bounded verification presented, we complement the approach with a qualitative abstraction verification approach. The approach is based on abstracting and verifying the qualitative behavior of the circuits using a combination of techniques from predicate abstraction and constraint solving along with bounded model checking. The principle novelties in this work is adapting the concept of lazy abstraction for the verification of CT-AMS designs. To this aim, we identified a set of basic qualitative predicates (Darboux polynomials) as invariance predicates which helps avoid the construction of an abstract model for the whole state space. We also proposed a constraint solving approach for the verification of safety, switching and reachability properties. This method does not require explicit representation of the state space but relies on generating functions that prove or disapprove the properties.

To tackle the verification of discrete-time AMS designs, we have defined an induction based bounded model checking technique that traverses the structure of the normalized properties and provides a formal correctness proof or a counterexample. Image computations for induction are performed using interval arithmetics over these symbolic expressions.

We have applied the verification methodology proposed in this thesis to example from several classes of AMS designs spanning various abstraction levels. We have implemented our methodology using standard libraries for symbolic computation available in Mathematica, allowing the development of a fully automated verification engine. Experimental results have proven the feasibility of the proposed approach.

Future Work.

The formal verification of AMS designs is a relatively young research field and still under-developed, which is a bad and a good sign at the same time. It is bad because this shows the lack of extensive research which is due mainly to the complexity of the verification process and the challenging problems mostly inherited from the hybrid systems. Also, it is due to the different scientific backgrounds between AMS engineers, control engineers and computer scientists. However, this can motivate interdisciplinary collaborations. The good news is that room for exploration is yet wide open. Among the interesting directions is developing an AMS theory with high-order logic, process algebraic languages for AMS designs and formalizing the AMS theory within a formal theory like abstract interpretation, and developing specification logics for frequency properties among others. Another important direction is incorporating formal verification within the design flow, hence complementing simulation, testing and symbolic analysis. Also, the problem of extending classical temporal logics to derive suitable descriptions of analog properties is of great interest.

From our point of view, our priority future work can be summarized as follows:

- More investigation is needed to improve the implementation to verify more complex circuits and to measure the limitation of the proposed methodology. Another challenge is to define and to verify more important properties related to industrial problems like audio and RF systems.

- Investigating alternative implementations to improve the experimental capacity over more complex systems and to measure the limitation of the proposed methodology.

- Also, an important effort is needed to classify the kind of properties and AMS systems that can be verified using this verification approach.

- Extraction of the design equations from the circuit descriptions (Schematic diagrams or HDL-AMS designs). We are currently looking for methods to extract and

simplify the system equations using Bond graph analysis.

Extracting the system equations to be used in behavioral modeling is a challenging task in the AMS design process. Nodal analysis techniques have been developed to this aim by extracting equations from the circuit netlist. However such extracted equation are very large in general and complicated to be used for behavioral analysis required at higher level in the design process. To overcome such problem, abstraction techniques have been developed as to generate simplified models preserving some characteristics of the initial designs.

Appendix A

Mathematica Implementations

A.1 Mathematica Functions

We have implemented a prototype for the presented verification algorithms using symbolic algebraic manipulation and real number theorem proving developed inside the computer algebra tool *Mathematica* [114]. Proposed verification functions like *Prop_Check* and *Prop_Verify* can be done using equational theorem proving function in Mathematica such as *Reduce*. *Reduce*[*expr*, *vars*] simplifies the statement *expr* by solving equations or inequalities for *vars* and by eliminating quantifiers. The statement expr can be any logical combination of:

- $lhs = rhs$ Equations
- $lhs \diamond rhs$, where $\diamond \in \{\neq, \leqslant, <, >, \geqslant\}$ Inequalities
- $expr \in dom$ Domain Specifications
- $ForAll[x, cond, expr]$ Universal Quantifiers
- $Exists[x, cond, expr]$ Existential Quantifiers

Reduce gives *True* if the *expr* is proved to be always true, *False* if *expr* is proved

to be always false and a reduced *expr* otherwise. The Mathematica implementation of *Reduce* is inspired by a real polynomial decision algorithm defined in [101].

Example A.1.1. For example, the safety verification problem in Example 5.2.1 can be formulated using *Reduce* as follows:

$$Reduce[Exists[\{x_1,x_2\}, 1-x_1^2-x_2^2 \geq 0 \&\& 1-x_1^2+x_2^2 \geq 0, -3+x_1^2+x_2 \geq= 0], \{x_1,x_2\}]$$

Example A.1.2. For simplicity of visualization, we provide details about applying the induction for the verification of first order $\Delta\Sigma$ modulator of one-bit with two quantization levels, $+1$V and -1V. The quantizer (input signal $y(n)$) should be between -2V and $+2$V in order not to be overloaded. The SRE of the $\Delta\Sigma$ is :

$$y(n) = y(n-1) + u(n) - v(n-1)$$
$$v(n-1) = IF(y(n-1) > 0, 1, -1)$$

Stability is expressed with the following property: $G|y(n)| \leq 2$, with the input $|u| \leq 1$ and the initial condition $|y(0)| \leq 1$. Informally, the property means that to ensure that the modulator will always be stable starting from initial conditions, we must ensure that the modulator quantizer is in the interval $[-2,2]$, if the input of the quantizer initially bounded in the interval $[-1,1]$ and the modulator input in the interval $[-1,1]$. The property proof at time n can be formulated as follows:

```
in:= Reduce[
      ForAll[{u,y[n-1]}, And[-1 < u < 1,
                                  -2 < y[n-1] < 2 ],
            And[(-1+u+y[n-1] ⩽ 2),
               (1+u+y[n-1]⩾ -2)]],
            {u,y[n-1]}, Reals]
out:= True
```

The function *FindInstance*[*expr*, *vars*, *assum*] finds an instance of *vars* that makes *expr True* if an instance exists, and gives {} if it does not. The result of *FindInstance* is of the form:

$$\{\{v_1 \rightarrow instance_1, v_2 \rightarrow instance_2, \ldots, v_m \rightarrow instance_m\}\}$$

where $vars = \{v_1, v_2, \ldots, v_m\}$. Furthermore, *FindInstance* may be able to find instances even if *Reduce* cannot give a complete reduction. The Mathematica implementation of *FindInstance* is based on variants of Newton's, Secant and Brent's methods [17].

Example A.1.3. Consider the First-order $\Delta\Sigma$ Modulator in Example A.1.2, with the input signal $\forall|u| > 1$ and initial condition $|y(0)| \leq 1$. The property: $G|y(n)| < 2$ fails to be verified. In fact, since the input to the modulator does not conform to the stability requirement, the modulator indeed will be unstable. For this property, we can find a counter-example:

in:= FindInstance[And[$1 < u, 1 > y[n-1] > 0$,
$(-1 + u + y[n-1] > 2)$],u,y[n-1]]
out:= {u $\rightarrow \frac{7}{2}$, y[n-1] $\rightarrow \frac{1}{2}$ }

The problem of finding invariants is an important part of the methodology. We need to find Darboux invariants and in the case of reachability verification, we look for invariants bounding the reachable states. Finding invariants is based on the evaluation of the coefficients of the predefined forms of polynomials. In this algorithm, we start with an invariant form with an initial degree and check if such invariant exists, if not, we increase the degree to form a new polynomial. A bound on the degree must also be specified to ensure termination of the search of the invariants. An arbitrarily assigned bound at the beginning of the algorithm is usually proposed hence ensuring termination. This is possible using the Mathematica *FindInstance* function, for example. For example, to find the Darboux invariants j we apply *FindInstance* as follows:

$$FindInstance[ForAll[\{x, y\}, \mathcal{D}j == \mathcal{K}j], \{coefs\}]$$

where j is a polynomial in x, y, with unknown coefficients $coefs$ and K is the cofactor.

Bibliography

[1] Accellera Property Specification Language Reference Manual (2004). Available: http://www.accellera.org

[2] M. Allam, H. Alla. From Hybrid Petri Nets to automata, Journal Europen des Systmes Automatiss, Hermes, 32(9-10):1165-85, 1998.

[3] G. Al-Sammane. Simulation Symbolique des Circuits Decrits au Niveau Algorithmique. PhD thesis, Université Joseph Fourier, Grenoble, France, July 2005.

[4] R. Alur, C. Courcoubetis, T. A. Henzinger, N. Halbwachs, P.H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. Theoretical Computer Science 138(1):3-34, Elsevier, 1995

[5] R. Alur, T. Dang, F. Ivancic. Reachability Analysis via Predicate Abstraction. In Hybrid Systems: Computation and Control, LNCS 2289, pp. 35-48. Springer, 2002.

[6] N. Amla, X. Du, A. Kuehlmann, R.P. Kurshan, K.L. McMillan. An Analysis of SAT-Based Model Checking Techniques in an Industrial Environment. Correct Hardware Design and Verification Methods, LNCS, 3725, pp. 254-268, Springer, 2005.

[7] D.K. Arrowsmith and C.M. Place. Ordinary Differential Equations: A Qualitative Approach with Applications. Chapman & Hall, 1982.

[8] E. Asarin, T. Dang, O. Maler. The d/dt Tool for Verification of Hybrid Systems. In Computer Aided Verification, LNCS 2404, pp. 365-370, Springer, 2002.

[9] A. Balivada, Y.V. Hoskote, J.A. Abraham, Verification of Transient Response of Linear Analog Circuits. In IEEE VLSI Test Symposium, pp. 42-47, 1995.

[10] S. Banerjee, D. Mukhopadhyay, D.R. Chowdhury. Computer Aided Test (CAT) Tool for Mixed Signal SOCs. In IEEE VLSI Design, pp. 787-790, 2005.

[11] S. Basu, R. Pollack, M.F. Roy. Algorithms in Real Algebraic Geometry, Springer, 2003.

[12] A. Bemporad and M. Morari, Verification of Hybrid Systems Via Mathematical Programming. In Hybrid Systems: Computation and Control, LNCS 1569, pp.31-45, Springer, 1999.

[13] M. Berz, G. Hoffstätter. Computation and Application of Taylor Polynomials with Interval Remainder Bounds, Reliable Computing, 4(1): 83-97, Springer, 1998.

[14] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded Model Checking. Advances in Computers, 58:118-149, Academic Press, 2003.

[15] M. S. Branicky, V. S. Borkar, and S. K. Mitter, A Unified Framework for Hybrid Control, In IEEE Proc. of Decision and Control, pp. 4228-4234, 1994.

[16] Cadence Design Systems. Using a SoC Functional Verification Kit to Improve Productivity, Reduce Risk, and Increase Quality. White Paper.

[17] F. Cellier, E. Kofman. Continuous System Simulation, Springer, 2006.

[18] A. Chutinan. Hybrid System Verification Using Discrete Model Approximations. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, May 1999.

[19] A. Chutinan and B. H. Krogh. Computational Techniques for Hybrid System Verification. IEEE Trans. on Automatic Control, 48(1):64-75, 2003.

[20] E. Clarke, A. Fehnker, Z. Han, B.H. Krogh, O. Stursberg, M. Theobald. Verification of Hybrid Systems based on Counterexample-Guided Abstraction Refinement. In Tools and Algorithms for the Construction and Analysis of Systems, LNCS 2619, pp. 192-207, Springer, 2003.

[21] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. Counterexample-Guided Abstraction Refinement. In Computer Aided Verification, LNCS 1855, pp. 154-169, Springer, 2000.

[22] E.M. Clarke, O. Grumberg, and D.A. Peled. Model Checking. MIT Press, 2000.

[23] E. Clarke, D. Kroening, J. Ouaknine, O. Strichman. Computational Challenges in Bounded Model Checking. Journal on Software Tools for Technlogy Transfer, 7(2): 174–183, Springer, 2005.

[24] P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In ACM Proc. on Principles of Programming, pp. 84-97, 1978.

[25] P. Cousot, R. Cousot. Abstract interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In ACM Symposium on Principles of Programming Languages, pp. 238252, 1977/

[26] D. Cox, J. little, and D. O'Shea. Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer, 1991.

[27] D. Dams. Abstraction in Software Model Checking: Principles and Practice, LNCS 2318, pp.14-21, Springer, 2002.

[28] T. Dang, A. Donze, O. Maler, Verification of Analog and Mixed-signal Circuits using Hybrid System Techniques. In Formal Methods in Computer-Aided Design, LNCS 3312, pp.14-17, Springer, 2004.

[29] T. R. Dastidar, P. P. Chakrabarti. Verification System for Transient Response of Analog Circuits Using Model Checking. In IEEE International Conference on VLSI, pp. 195-200, 2005.

[30] T. R. Dastidar, P. P. Chakrabarti. A Verification System for Transient Response of Analog Circuits. In ACM Trans. Design Automation of Electronic Systems, 12(3):1-39, 2007.

[31] C. Daws, A. Olivero, S. Tripakis, S. Yovine. The Tool KRONOS. Hybrid Systems: Verfication and Control, LNCS 1066, pp.208-219, 1996

[32] A. Emerson. Temporal and Modal Logic. Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics, pp. 995-1072, MIT Press, 1990

[33] I. Filanovsky, C. Verhoeven and M. Reja. Remarks on Analysis, Design and Amplitude Stability of MOS Colpitts Oscillator. In IEEE Tran. on Circuits & Systems 2, 54(9):800-804, 2007.

[34] M. Franzle. What Will Be Eventually True of Polynomial Hybrid Automata? In Theoretical Aspects of Computer Software, LNCS 2215, pp. 340-359, Springer, 2001.

[35] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. In Hybrid Systems: Computation and Control, LNCS 3414, Springer, pp. 258-273, 2005.

[36] G. Frehse, B. Krogh, R. Rutenbar, O. Maler, Time Domain Verification of Oscillator Circuit Properties, Electronic Notes Theoretical Computer Science, 153(3):9-22, 2006.

[37] G. Frehse, B. H. Krogh, R. A. Rutenbar. Verifying Analog Oscillator Circuits Using Forward/Backward Abstraction Refinement. In IEEE/ACM Design, Automation and Test in Europe, pp. 257-262, 2006.

[38] M. Freibothe, J. Schoenherr, and B. Straube. Formal Verification of the Quasi-Static Behavior of Mixed-Signal Circuits by Property Checking, Electronic Notes Theoretical Computer Sci., Elsevier, 153(3):23-35, 2006.

[39] W. Fulks. Advanced Calculus: An Introduction to Analysis. Wiley, 1978.

[40] M. Furi, M. Martelli. A Multidimensional Version of Rolle's Theorem. The American Mathematical Monthly, 102(3), 1995, pp. 243-249.

[41] A. Ghosh and R. Vemuri, Formal Verification of Synthesized Analog Circuits, In ACM/IEEE Int. Conference on Computer Design, pp. 40-45, 1999.

[42] G.G. Gielen and R. A. Rutenbar, Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits, Proceedings of the IEEE, 88(12):1825-1852, 2000.

[43] A. Goriely. Integrability and Nonintegrability of Ordinary Differential Equations, Advanced Series on Nonlinear Dynamics, Vol 19 World Scientific 2001.

[44] D. Grabowski, D. Platte, L. Hedrich, E. Barke. Time Constrained Verification of Analog Circuits using Model-Checking Algorithms. Electronic Notes Theoretical Computer Science, 153(3):37-52, 2006.

[45] S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In Computer Aided verification, LNCS 1254, pp. 72-83. Springer, 1997.

[46] P.R. Gray, P.J. Hurst, S.H. Lewis, and R.G. Meyer. Analysis and Design of Analog Integrated Circuits, Wiley, 2001

[47] M. R. Greenstreet, I. Mitchell: Integrating Projections. In Hybrid Systems : Computation and Control, LNCS 1386, pp. 159-174, Springer, 1998.

[48] M. R. Greenstreet: Verifying Safety Properties of Differential Equations. In Computer Aided Verification, LNCS 1102, pp. 277-287, Springer, 1996

[49] M. R. Greenstreet, I. Mitchell: Reachability Analysis Using Polygonal Projections. In Hybrid System: Computation and Control, LNCS 1569, pp.103-116, Springer, 1999.

[50] S. Gupta, B.H. Krogh, R.A. Rutenbar: Towards Formal Verification of Analog Designs, In Proc. IEEE/ACM Conference on Computer Aided Design, pp. 210-217, 2004.

[51] N. Halbwachs, P. Raymond, and Y. Proy. Verification of Linear Hybrid Systems by Means of Convex Approximations. In Symposium on Static Analysis, LNCS 864, pp. 223-237, 1994.

[52] K. Hanna, Reasoning about Real Circuits, In Theorem Proving in Higher Order Logics LNCS 859, pp. 235-253, Springer, 1994.

[53] K. Hanna. Automatic Verification of Mixed-Level Logic Circuits. In Formal Methods in Computer-Aided Design, LNCS 1522, pp.133-166, Springer, 1998.

[54] K. Hanna, Reasoning About Analog-Level Implementations of Digital Systems. Formal Methods in System Design, 16(2): 127-158, Kluver, 2000.

[55] W. Hartong, L. Hedrich, and E. Barke, Model Checking Algorithms for Analog Verification. In ACM/IEEE Design Automation Conference, pp. 542-547, 2002.

[56] W. Hartong, L. Hedrich, and E. Barke. On Discrete Modelling and Model Checking for Nonlinear Analog Systems. In Computer Aided Verification, LNCS 2404, pp. 401-413, Springer, 2002.

[57] W. Hartong, R. Klausen, and L. Hedrich. Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking, Advanced Formal Verification, pp. 205-245, Kluwer, 2004.

[58] T.A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In Symp. on Principles of Programming Languages, ACM, pp. 58–70, 2002.

[59] T. A. Henzinger, P. Ho, and Howard Wong-Toi. Algorithmic Analysis of Nonlinear Hybrid Systems. IEEE Transactions on Automatic Control 43:540-554, 1998.

[60] T.A. Henzinger and P. Ho. A Note on Abstract-Interpretation Strategies for Hybrid Automata. In Hybrid Systems II, Lecture Notes in Computer Science 999, Springer-Verlag, 1995, pp. 252-264.

[61] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems. Software Tools for Technology Transfer, 1(1–2):110-122, Kluwer, 1997.

[62] L. Hedrich and E. Barke, A Formal Approach to Nonlinear Analog Circuit Verification. In IEEE/ACM Intl. Conference on Computer Aided Design, pp. 123-127, 1995.

[63] L. Hedrich and E. Barke, A Formal Approach to Verification of Linear Analog Circuits with Parameter Tolerances. In IEEE/ ACM Design, Automation and Test in Europe, pp. 649-654, 1998.

[64] M.P. Kennedy. Chaos in the Colpitts Oscillator, In IEEE Transactions on Circuits and Systems 1, 41:77174, 1994.

[65] P. Kopke, T. Henzinger, A. Puri and P. Varaiya. What's Decidable About Hybrid Automata?. In ACM Symposium on Theory of Computing, pp. 372-382, 1995.

[66] T. Kropf. Introduction to Formal Hardware Verification, Springer, 2000.

[67] K. Kundert, H. Chang, D. Jefferies, G. Lamant, E. Malavasi, F. Sendig, Design of Mixed-signal Systems-on-a-chip, IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, 19(12):1561-1571, 2000.

[68] R.P. Kurshan. Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach, Princeton University Press, 1995.

[69] G. Lafferriere, G. J. Pappas, and S. Yovine. Reachability Computation for Linear Hybrid Systems. In Proc. of IFAC World Congress, pp. 7-12, 1999.

[70] G. Lafferriere, G. J. Pappas, and S. Yovine. Symbolic Reachability Computation of Families of Linear Vector Fields. Journal of Symbolic Computation, 32(3):231-253, Academic Press, 2001.

[71] J. Le Bail, H. Alla, R. David. Hybrid Petri Net, In Proc. of European Control Conference, pp. 1472-7, 1991.

[72] W. Lee, A. Pardo, J.-Y. Jang, G. Hachtel, and F. Somenzi. Tearing based automatic abstraction for CTL model checking. In IEEE/ACM International Conference on Computer-Aided Design, pp.76-81, 1996.

[73] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda. Verification of Analog/mixed-signal Circuits using Labeled Hybrid Petri Nets. In International Conference on Computer-Aided Design, pp.275-282, 2006

[74] S. Little, D. Walter, N. Seegmiller, C.J. Myers, T. Yoneda. Verification of Analog and Mixed-Signal Circuits Using Timed Hybrid Petri Nets. In Proc. of Automated Technology for Verification and Analysis, LNCS 3299, pp. 426-440, Springer, 2004.

[75] S. Little, D. Walter, K. Jones, C. J. Myers. Analog/Mixed-Signal Circuit Verification Using Models Generated from Simulation Traces. In Automated Technology for Verification and Analysis, LNCS 4762, pp. 114-128, Springer, 2007.

[76] R.P. Kurshan and K.L. McMillan. Analysis of Digital Circuits Through Symbolic Reduction. IEEE Trans. on Computer-Aided Design 10:13501371, 1991.

[77] K. Makino, M. Berz. Remainder Differential Algebras and their Applications. In Computational Differentiation:

[78] O. Maler, D. Nickovic, Monitoring Temporal Properties of Continuous Signals. In Formal Modelling and Analysis of Timed Systems, LNCS 3253, pp.152-166, Springer, 2004.

[79] O. Maler, D. Nickovic and A. Pnueli, Real-Time Temporal Logic: Past, Present, Future. In Formal Modelling and Analysis of Timed Systems, LNCS 3829, pp. 2-16, Springer, 2005

[80] O. Maler, D. Nickovic, Amir Pnueli, From MITL to Timed Automata, In Formal Modelling and Analysis of Timed Systems, LNCS 4202, pp. 274-289, Springer, 2006.

[81] O. Maler, A. Pnueli. Reachability Analysis of Planar Multi-limear Systems. In Computer Aided Verification, LNCS 697, 194-209, Springer, 1993.

[82] L. Mendona de Moura, B. Dutertre, N. Shankar. A Tutorial on Satisfiability Modulo Theories. In Computer Aided Verification, LNCS 4590, pp. 20-36, Springer, 2007.

[83] B. Mishra. Algorithmic Algebra, In Texts and Monographs in Computer Science Series, Springer, 1993.

[84] J.S. Moore. Introduction to the OBDD Algorithm for the ATP community. Journal of Automated Reasoning, 12(1):33–45, Springer, 1994. Techniques, Applications, and Tools, pp. 63-75, SIAM, 1996.

[85] R.E. Moore. Methods and Applications of Interval Analysis, Society for Industrial and Applied Mathematics, 1979.

[86] C.J. Myers, R. R. Harrison, D. Walter, N. Seegmiller, S. Little, The Case for Analog Circuit Verification. Electronic Notes Theoretical Computer Science, 153(3):53-63, 2006.

[87] V. Mysore, C. Piazza, B. Mishra. Algorithmic Algebraic Model Checking II: Decidability of Semi-algebraic Model Checking and Its Applications to Systems Biology.

Automated Technology for Verification and Analysis, LNCS 3707, pp. 217-233, Springer, 2005.

[88] N.S. Nedialkov, V. Kreinovich and S.A. Starks. Interval Arithmetic, Affine Arithmetic, Taylor Series Methods: Why, What Next? In Numerical Algorithms, 37:325-336, Springer, 2004.

[89] N.S. Nedialkov, K.R. Jackson, and G.F. Corliss. Validated Solutions of Initial Value Problems for Ordinary Differential Equations. Applied Mathematics and Computation, Elsevier, 105(1):21-68, 1999.

[90] D. Nickovic, O. Maler. AMT: a Property-based Monitoring Tool for Analog Systems. In Formal Modelling and Analysis of Timed Systems, Austria, LNCS 4763, pp. 304-319, Springer, 2007.

[91] F. Pecheux, C. Lallement, and A. Vachoux, VHDL-AMS and Verilog-AMS as Alternative Hardware Description Languages for Efficient Modeling of Multidiscipline Systems. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 24(2):204-225, 2005.

[92] S. Prajna, A. Jadbabaie. Safety Verification of Hybrid Systems Using Barrier Certificates. In Hybrid Systems: Computation and Control, LNCS 2993, pp. 477-492, Springer, 2004.

[93] S. Ratschan, Z. She. Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement. In Hybrid System: Computation and Control, LNCS 3414, pp. 573-589, Springer, 2005.

[94] E. Rodrguez-Carbonell, Ashish Tiwari. Generating Polynomial Invariants for Hybrid Systems. Hybrid Systems: Computation and Control, LNCS 3414, pp. 590-605, Springer, 2005.

[95] J. Roll, A. Bemporad, and L. Ljung. Identification of Piecewise Affine Systems via Mixed-integer Programming, Automatica, 40(1): 37-50, Elsevier, 2004.

[96] R.A. Rutenbar, G.G. Gielen, B.A. Antao. Computer-Aided Design of Analog Integrated Circuits and Systems, IEEE Press, 2002.

[97] A. Salem. Semi-formal verification of VHDL-AMS Descriptions. In IEEE Int. Symposium on Circuits and Systems, pp. 333-336, 2002.

[98] S. Sankaranarayanan, H. Sipma, Z. Manna. Constructing Invariants for Hybrid Systems. In Hybrid Systems: Computation and Control, LNCS 2993, pp 539-554, Springer, 2004.

[99] S. Seshadri, J.A. Abraham, Frequency Response Verification of Analog Circuits Using Global Optimization Techniques, Journal of Electronic Testing, 17(5): 395-408, Springer, 2001.

[100] S. Steinhorst, A. Jesser, L. Hedrich. Advanced Property Specification for Model Checking of Analog Systems. In Analog'06, pp. 63-68, 2006,

[101] A. Strzebonski. Real Polynomial Decision Algorithm Using Arbitrary-Precision Floating Point Arithmetic. Reliable Computing, 5(3):337-346, Springer, 1999.

[102] O. Stursberg, S. Kowalewski, I. Hoffmann, and J. Preuig. Comparing Timed and Hybrid Automata as Approximations of Continuous Systems. In Hybrid Systems: Computation and Control,LNCS 1273, pp. 361-377, Springer, 1996.

[103] O. Stursberg,S. Kowalewski, S. Engell: Generating timed discrete models of continuous systems. In Proc. IMACS, Symposium on Mathematical Modelling, pp. 203-209, 1997

[104] L. Tan, J. Kim, I. Lee. Testing and Monitoring Model-based Generated Program. Electr. Notes Theoritical Computer Science, 89(2):128-148, 2003.

[105] P. Thati, G. Rosu. Monitoring Algorithms for Metric Temporal Logic Specifications. Electr. Notes Theor. Comput. Sci., Elsevier, 113: 145-162, 2005.

[106] A. Tiwari and G. Khanna. Series of Abstractions for Hybrid Automata. In Hybrid Systems: Computation and Control, LNCS 2289, pp. 465-478, Springer, 2002.

[107] A. Tiwari and G. Khanna. Nonlinear Systems: Approximating Reach Sets. In Hybrid Systems: Computation and Control, LNCS 2993, pp. 600-614, Springer, 2004.

[108] A. Vachoux, C. Grimm, K. Einwich. Towards Analog and Mixed-Signal SOC Design with SystemC-AMS.In Electronic Design, Test and Applications, IEEE, pp. 97-102, 2004.

[109] Verilog-AMS Language Reference Manual (2004). Available: http://www.accellera.org

[110] VHDL-AMS Language Reference Manual (2004). http://www.eda.org/vhdl-ams/

[111] J. Vlach, K. Singhal. Computer Methods for Circuit Analysis and Design. Kluver, 2003.

[112] D. Walter, S. Little, N. Seegmiller, C. Myers, and T. Yoneda, Symbolic Model Checking of Analog/Mixed-Signal Circuits. In IEEE Asia and South Pacific Design Automation Conference, pp.316-323, 2007

[113] D. Walter, S. Little, C. Myers. Bounded Model Checking of Analog and Mixed-Signal Circuits Using an SMT Solver. In Automated Technology for Verification and Analysis, LNCS 4762, pp. 66-81, Springer, 2007.

[114] S. Wolfram. Mathematica: A System for Doing Mathematics by Computer. Addison Wesley Longman Publishing, USA, 1991.

[115] H. Yazarel and G. J. Pappas. Geometric programming relaxations for linear system reachability. In proc. AACC of American Control, pp. 553-559, 2004

[116] J. Yuan, C. Pixley, A. Aziz. Constraint-Based Verification, Springer, 2006.

[117] C. Yan, M. Greenstreet. Circuit-Level Verification of a High-Speed Toggle, IEEE International Conference on Formal Methods in Computer-Aided Design, pp. 199-206, 2007.

[118] Z. Manna. Mathematical Theory of Computation. Dover, 2003.

Biography

Education

- **Concordia University:** Montreal, Quebec, Canada
  Ph.D candidate, in Electrical Engineering, 01/03-present
- **Concordia University:** Montreal, Quebec, Canada
  M.A.Sc., in Electrical Engineering, 09/00 - 12/02
- **Ain Shams University:** Cairo, Egypt.
  B. Eng., Electronics & Communication Engineering, 09/95 - 09/00

Work Experience

- **Research Assistant:** 09/00-present
  ECE Department, Hardware Verification Group (HVG), Concordia University
- **Teaching Assistant:** 09/00-present
  ECE Department, Concordia University

Publications

- **Journal Publications:**

  [Bio:Jr-01] M.H. Zaki, S. Tahar, and G. Bois: Qualitative Abstraction based Verification for Analog Circuits. Revue des Nouvelles Technologies de l'information, Vol. 4, Issue 7, December 2007, RNTI-SM-1, Edition Cepadues, pp. 147-158.

  [Bio:Jr-02] M.H. Zaki, S. Tahar, and G. Bois: Formal Verification of Analog and Mixed Signal Designs : A Survey. Microelectronics Journal, Elsevier, 2008, In Print.

[Bio:Jr-03] G. Al Sammane, M.H. Zaki, S. Tahar, and G. Bois: A Formal Approach for the Verification of Discrete-Time Analog/Mixed Signal Designs. Transaction on Computer Aided Design. Submitted.

[Bio:Jr-04] M.H. Zaki, W. Denman, S. Tahar and G. Bois. A Formal Verification Methodology for the Analog behaviour of Embedded Systems. AIAA Journal of Aerospace Computing, Information, and Communication. Submitted.

[Bio:Jr-05] M.H. Zaki, G. Al Sammane, S. Tahar and G. Bois. A Bounded Verification Approach for Analog and Mixed-Signal Designs Using Symbolic and Interval based Methods . Formal Methods in System Design, Springer. Submitted

- **Conferences Publications:**

[Bio:Cf-01] W. Denman, M.H. Zaki, S. Tahar A Bond Graph Approach for the Constraint based Verification of Analog Circuits. In Workshop on Formal Verification of Analog Circuits (FAC'08), Princeton, USA, July 14th, 2008.

[Bio:Cf-02] R. Narayanan, N. Abbasi, G. Al Sammane, M.H. Zaki and S. Tahar. A Comparative Study of AMS Circuit Simulation in VHDL-AMS and SystemC-AMS. In International Symposium on Embedded Systems & Critical Applications (ISESCA'08), Tunisia, May 2008.

[Bio:Cf-03] Z.J. Dong, M.H. Zaki, G. Al Sammane, S. Tahar and G. Bois: Checking Properties of PLL Designs using Run-time Verification; Proc. IEEE International Conference on Microelectronics (ICM'07), pp.125-128, Cairo, Egypt, December 2007.

[Bio:Cf-04] G. Al Sammane, M.H. Zaki, Z.J. Dong and S. Tahar: Towards Assertion Based Verification of Analog and Mixed Signal Designs Using PSL; Proc. Languages for Formal Specification and Verification, Forum on Specification & Design Languages (FDL'07), Barcelona, Spain, September 2007.

[Bio:Cf-05] M. Zaki, G. Al Sammane, S. Tahar, and G. Bois: Combining Symbolic Simulation and Interval Arithmetic for the Verification of AMS Designs; Proc. IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD'07), pp.207-215, Austin, Texas, USA, November 2007.

[Bio:Cf-06] M. Zaki, G. Al Sammane, and S. Tahar: Constraint-Based Verification of Delta Sigma Modulators Using Interval Analysis; Proc. IEEE Midwest Symposium on Circuits & Systems (MIDSWEST'06), pp.726-729, Montreal, Quebec, Canada, August 2007.

[Bio:Cf-07] Z.J. Dong, M. Zaki, G. Al Sammane, S. Tahar and G. Bois: Run-Time Verification using the VHDL-AMS Simulation Environment; Proc. IEEE Northeast Workshop on Circuits and Systems (NEWCAS'07), pp.1513-1516, Montreal, Quebec, Canada, August 2007.

[Bio:Cf-08] M. Zaki, S. Tahar, and G. Bois: A Symbolic Approach for the Safety Verification of Continuous Systems; Proc. International Conference on Computational Science (ICCS'07), pp. 93-100, Beijing, China, May 2007.

[Bio:Cf-09] M. Zaki, G. Al Sammane, and S. Tahar: Formal Verification of Analog and Mixed Signal Designs in Mathematica; In: Y. Shi et al. (Eds.), Computational Science (ICCS'07), Lecture Notes in Computer Science 4488, Springer Verlag, 2007, pp. 263-267, Beijing, China, May 2007.

[Bio:Cf-10] G. Al Sammane, M. Zaki, and S. Tahar: A Symbolic Methodology for the Verification of Analog and Mixed Signal Designs; Proc. IEEE/ACM Design Automation and Test in Europe (DATE'07), pp.1-6, Nice, France, April 2007.

[Bio:Cf-11] M. Zaki, S. Tahar, and G. Bois: Abstraction Based Verification of Analog Circuits Using Computer Algebra and Constraint Solving; Proc. International Workshop on Symbolic Methods and Applications to Circuit Design (SMACD'06), Florence, Italy, October 2006.

[Bio:Cf-12] M. Zaki, S. Tahar, and G. Bois: Formal Verification of Analog and Mixed Signal Designs: Survey and Comparison; Proc. IEEE Northeast Workshop on Circuits and Systems (NEWCAS'06), pp.281-284, Gatineau, Quebec, Canada, June 2006.

[Bio:Cf-13] M. Zaki, S. Tahar, and G. Bois: A Practical Approach for Monitoring Analog Circuits; Proc. ACM 16th Great Lakes Symposium on VLSI (GLS-VLSI'06), pp. 330-335, Philadelphia, Pennsylvania, USA, April 2006.

[Bio:Cf-14] M. Zaki, A. Habibi, S. Tahar, and G. Bois: On the Formal Analysis of Analog Systems using Interval Abstraction; Proc. NETCA Workshop on Verification and Theorem Proving for Continuous Systems, Oxford, UK, August 2005.

[Bio:Cf-15] M. Zaki, Y. Mokhtari, and S. Tahar: Model Reduction Tool for Hardware Verification. Proc. IEEE Northeast Workshop on Circuits and Systems (NEWCAS'04), pp. 57-60, Montreal, Quebec, Canada, June 2004.

[Bio:Cf-16] A. Talaat, M. Zaki and S.Tahar: A tool for Converting Finite State Machine to VHDL; Proc. IEEE Canadian Conference on Electrical & Computer Engineering (CCECE'04), pp. 1907-1910, Niagara Falls, Ontario, Canada, May 2004.

[Bio:Cf-17] M. Zaki, Y. Mokhtari, and S. Tahar: A Path Dependency Graph for Verilog Program Analysis; Proc. Northeast Workshop on Circuits and Systems (NEWCAS'03), Montreal, Quebec, Canada, June 2003.

[Bio:Cf-18] M. Zaki and S. Tahar: Syntax Code Analysis and Generation for Verilog; Proc. IEEE Canadian Conference on Electrical & Computer Engineering (CCECE'03), pp. 235-240, Montreal, Quebec, Canada, May 2003.

- **Technical Reports:**

[Bio:Tr-01] M.H. Zaki, S. Tahar, G. Bois: A Survey on Formal Methods for Analog and Mixed Signal Designs, Technical Report, ECE Dept, Concordia University, May 2006.

[Bio:Tr-02] M.H. Zaki, G. Al Sammane, S. Tahar and Guy Bois: A Bounded Model Checking Approach for AMS Designs; Technical Report, Concordia University, Department of Electrical and Computer Engineering, May 2007.

[Bio:Tr-03] M.H. Zaki, S. Tahar and G. Bois: Combining Constraint Solving and Formal Methods for the Verification of Analog Designs; Technical Report, Concordia University, Department of Electrical and Computer Engineering, June 2007.

[Bio:Tr-04] Z. J. Dong, M.H. Zaki, G. Al Sammane, S. Tahar and G. Bois. A Run-Time Verfication Approach for AMS Designs. Technical Report, Department of Electrical and Computer Engineering, Concordia University, July 2007.

[Bio:Tr-05] W. Denman, M. Zaki and S. Tahar. Analog Formal Verification Via Bond Graphs and Constraint Solving. Technical Report, ECE Dept., Concordia University, Montreal, Quebec, Canada, April 2008.