EFFICIENT MODELING AND VERIFICATION OF ANALOG/MIXED-SIGNAL CIRCUITS USING LABELED HYBRID PETRI NETS

by

Scott R. Little

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

December 2008

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Scott R. Little

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

______________	______________________________
	Chair: Chris J. Myers
______________	______________________________
	Ganesh Gopalakrishnan
______________	______________________________
	Priyank Kalla
______________	______________________________
	John Regehr
______________	______________________________
	Ken Stevens

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of __________Scott R. Little__________ in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date	Chris J. Myers Chair, Supervisory Committee

Approved for the Major Department

Martin Berzins
Chair/Dean

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

Analog circuit design is traditionally done by expert designers in an ad hoc manner heavily dependent on simulation. This methodology has worked successfully for many years, but process variation and design complexity are prompting designers to explore new techniques. Formal methods are being used successfully to aid in the complex validation problem for digital circuits. This dissertation presents formal methods for *analog and mixed-signal* (AMS) circuits.

This dissertation describes the development of a formal model, *labeled hybrid Petri nets* (LHPNs), appropriate for the modeling and verification of AMS circuits. An LHPN is a Petri net variant capable of modeling both continuous and discrete quantities. Creating an LHPN model of an AMS circuit by hand is a complicated and error prone exercise that requires expert knowledge. This is unacceptable for practical adoption of the LHPN model and its associated analysis methods. For this reason, this dissertation introduces an automatic LHPN model generation method. The method uses a set of simulation traces and a desired system property to generate an LHPN modeling the behavior of the simulation traces. The model generator can also be used to generate abstract Verilog-AMS or VHDL-AMS models suitable for use in system-level simulations.

Formal verification of a property over the entire state space of an LHPN model is complicated by the infinite state of the model. For this reason, the infinite states of the model are grouped into potentially finite groups of equivalent states for verification. *Difference bound matrices* (DBMs), a restricted form of convex polygons, are used to represent these equivalent classes of infinite states. Reachability analysis using DBMs is very efficient at the cost of exactness. This dissertation presents algorithms for conservative state space analysis and verification of LHPNs.

Finally, these methods are demonstrated on several case studies of AMS circuits from both academia and industry. The formal verification methods demonstrate the ability to find bugs missed by standard simulations. The abstract modeling methods show the promise of using automatically generated abstract models by demonstrating up to 40x speedup for some examples.

To good light and the landscape...

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

I do not think I ever really expected to be at this juncture. Somehow it has happened. It certainly would not have happened without the support and guidance of several people to whom I owe a great deal.

First, I would like to thank my advisor, Chris Myers. Chris took me on as a pretty clueless freshman. He has been patient enough to mentor me for more than eight years. I have grown tremendously through his invaluable guidance and teaching. I have learned many things from Chris in regard to research, academia, and life in general.

Next, I need to thank my ever patient and encouraging wife Monica. She has lived through the triumph and pain of my undergraduate and graduate education. She managed to hold down a “real” job while I “worked” on my Ph.D. I am not sure what my son Milo will remember about this whole experience. Probably something about Dad randomly going up to school to do some work. I am quite sure that my daughter Holly will remember nothing of the experience. That may be for the best.

I have enjoyed spending time in the “lab” due to friendships formed with many different labmates. I am not going to name everyone, but I would like to single out Eric Mercer and David Walter. Eric spent a significant amount of his time mentoring me when I first starting working in the lab. He helped keep me moving along and prevented me from getting too frustrated. David and I worked together on AMS verification. I really appreciate David’s willingness to listen to and sympathize with my numerous rants. I also appreciate the trailblazing he did which has made my experience much, much easier.

I would like to thank several others at the University of Utah who have contributed to this work and my education in general. Nick Seegmiller and Kevin Jones contributed important pieces of this work as undergraduates. Nick wrote the VHDL-AMS compiler while Kevin created the initial ideas and infrastructure for the model generator. The friendship of labmates Eric Peskin, Kip Killpack, Vijay Durairaj, and Sivaram Gopalkrishnan made life in the lab just that much more enjoyable. I would like to thank my committee composed of Ganesh Gopalakrishnan, Priyank Kalla, John Regehr, and Ken Stevens for their help and support. I would like to thank Erik Brunvand, Al Davis, Dave

Hanscom, and Arn Stolp for their great teaching and advice. They helped me get started and stay interested in academia.

I would like to thank Tomohiro Yoneda for his tremendous hospitality and well thought out feedback. My trips to Tokyo have very pleasant and productive thanks in large part to Tomohiro.

I need to thank my parents, Ray and Ruth, who have been amazingly supportive of something that they have not totally understood. This graduate school thing is a new and somewhat perplexing endeavor for them. Regardless of the doubts, they have given their support, which I appreciate. I also want to thank my grandparents, Berwyn and Virginia. I appreciate their unconditional support, encouragement, and extra bedroom. I particularly appreciate my grandfather who helped me get a job at L-3 Communications and opened my eyes to the beauty of engineering.

My internship experiences have been an invaluable part of my research. They improved and dramatically changed my understanding of the AMS verification problem. I would like to thank Kevin Jones at Rambus for giving me the opportunity to spend a summer at Rambus and in the Bay Area. It is a summer that I will not soon forget. I appreciate Kevin's patience as he taught me how to do research in a few short months. I would also like to thank my coworkers at Rambus: Tom Sheffler, Katy Mossawir, Jaeha Kim, and Brian Leibowitz. I would like to thank Alper Sen for the part he played in getting me an internship and later a job at Freescale Semiconductor. The internship put me on the right track and in the right frame of mind to wrap up my work at the University of Utah.

Semiconductor Research Corporation (SRC) has been an invaluable asset to me during my time at the University of Utah. They have funded my research, but more importantly, they have provided me opportunities to present my research to industry. These opportunities have lead to internships, useful feedback, and valuable networking opportunities. I would like to thank Bob Kurshan for the mentoring he gave me as my industrial liaison in the SRC Fellowship program. His feedback has been very valuable.

Finally, I would like to thank Jorge Cham for writing PhD Comics. His strips have helped me lighten up, procrastinate, and realize that I am having the typical graduate student experience...

CHAPTER 1

INTRODUCTION

Electronic computing devices are becoming increasingly complex and pervasive in today's society. These devices are used in a wide range of applications such as digital cameras, mobile phones, automobiles, airplanes, satellites, medical monitoring equipment, etc. Some of these devices are more safety critical than others, but even nonsafety critical devices must function properly a high percentage of the time to be competitive in an increasingly crowded marketplace. These devices are largely composed of digital circuits. Therefore, significant effort and progress has been made in the modeling and verification of digital circuits [50, 49, 31, 32, 40, 95, 122]. Although analog circuits do not make up a large portion of these systems, they play an important role when interfacing with the inherently analog environment. As a result, it is critical that the analog circuits and the circuits interfacing the analog circuits to the digital circuits, *mixed-signal circuits*, function properly. Therefore, this dissertation focuses on the modeling and verification of *analog and mixed-signal* (AMS) circuits.

1.1 AMS Circuit Design and Verification Methodology

During the past decade, AMS design has undergone dramatic changes fueled largely by an increase in the complexity of the circuits and the variability of the semiconductor fabrication processes. Over the years, AMS designers have developed a design methodology based on a high level of designer skill, creativity, and experience. This design methodology relies heavily on simulation to verify the correctness of the circuit. As the complexity and variability increase, this simulation based methodology becomes impractical.

The increased circuit complexity comes from two major sources, algorithmic complexity and modes of operation [33]. A powerful example that illustrates the increase in algorithmic complexity is the analog to digital converter (ADC). Initially, ADCs found on integrated circuits were flash ADCs. Flash ADCs do an analog to digital conversion in a single ADC clock cycle, but the drawback of flash ADCs is that they require one comparator per bit of resolution. To help mitigate the area cost of flash ADCs, dual-slope ADCs that require less hardware but require three cycles per conversion were developed. Successive approximation ADCs followed. They do a conversion requiring the same number of cycles as the bits in the output but do not require any extra hardware for the extra bits of resolution. Currently, the state of the art ADC is a $\Delta\Sigma$ ADC [29, 79, 22, 19] that uses complex digital signal processing algorithms and may require hundreds or thousands of cycles to do a single ADC conversion. This increase in algorithmic complexity results in an increase in simulation time which stands in conflict to the reduced time to market for today's designs.

The modes of operation for AMS circuits are increasing because today's designs need to operate over a wide range of conditions and protocols to be compatible with the large number of applications where they may be used. This is due in part to the cost of a large analog design. If a company is going to justify the cost for a large analog design, it is necessary that the design operate in many potential products. For instance, a mobile phone receiver must support a number of communication protocols, operate over a wide range of operating conditions (temperature, humidity, etc.), and be as power efficient as possible while remaining responsive to the user. Each of the growing number of modes needs to be verified for proper operation. Power down modes can be particularly problematic as they are often similar to initialization sequences which are notoriously long and difficult to verify using standard simulation techniques.

Process variation is increasing the difficulty of AMS circuit design and verification. In the past, AMS designers only needed to ensure that their designs operated properly for nominal conditions and a handful of global variations. With the increased variability of modern semiconductor processes, designers must ensure that the design works over a growing range of process variations and environmental conditions. There are two primary methods of characterizing a design over a range of variation, corner simulation and Monte Carlo simulation. Corner or PVT (Process, Voltage, and Temperature) simulations are intended to characterize the design for global variation. Ideally, simulations for all different combinations of extreme variations for process, voltage, and temperature are performed. Traditionally, process corners are run for fast and slow NMOS and PMOS devices. For instance, if simulations are run for fast and slow process corners for both NMOS and PMOS devices, temperature extremes, and voltage extremes for a single supply voltage this would require 2^4 or 16 simulations. Doing sixteen simulations is not unreasonable, but the number of simulations required is exponentially related to the number of sources of variation. In modern processes, the number of sources of variation is increasing. The sources of process variation are increasing as designs are simulated with variations for resistances, capacitances, etc. as well as the number of voltage sources. This often results in the designer running only a subset of the variation space because it would take too long to run all possible corners.

Monte Carlo simulation [110] is used to help designers understand the effects of local parameter variations. For instance, it is common for designers to run Monte Carlo simulations for a range of threshold voltages particularly when the circuit involves matched pairs of transistors. The number of simulations required for adequate confidence results using Monte Carlo simulation is situational but typically on the order of hundreds to thousands of simulations. The number of Monte Carlo simulations designers are using is also increasing as the sources for local variation increase.

These complexity and variability issues are compounded by the ad hoc nature of today's analog verification methodology. Figure 1.1 illustrates a typical analog design methodology. The system-level specification is set by the system architects based on customer requirements, industry standards, architectural explorations, etc. Individual subsystem specifications and even topologies may be provided for critical subsystems by the architects. After the architectural decisions have been made, the design team lead assigns each designer a block of the system to design and verify. Each designer derives the block-level specification from the provided system-level specification for his individual blocks. The designer then begins the process of designing a circuit to meet the specification for the nominal case. This design is done using a set of simulations deemed adequate to characterize the system by the designer. Once the design is satisfactorily operating at the nominal case, the designer begins the iterative optimization process. Optimization characterizes the design by exploring the operation over ranges of both global and local variation in an attempt to center the design within the variation space. Once the designer is satisfied with the circuit optimization, he declares the design complete.

This process has worked well for a number of years, but it is beginning to show signs of inadequacy. As the level of complexity grows in ways described previously, this largely ad hoc process is not able to cope with the decreasing time to market and the increasing cost of *respins*, redesigns that require a change to the mask set used in semiconductor

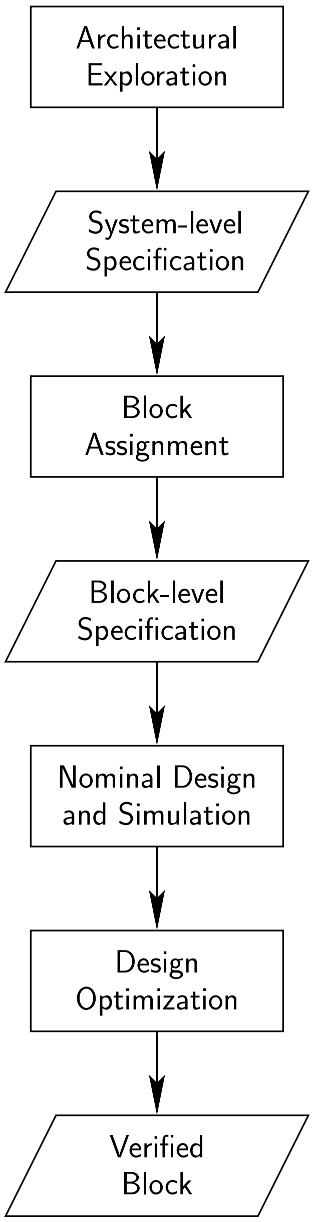

Architectural Exploration
System-level Specification
Block Assignment
Block-level Specification
Nominal Design and Simulation
Design Optimization
Verified Block

Figure 1.1. A block diagram describing the analog design process.

fabrication. In addition to needing improved simulators or simulation techniques, the AMS design process is in need of design automation particularly in the verification arena. It is possible for designers to create an automated suite of simulations for a given circuit although each simulation still needs to be checked individually by a designer to verify the correctness of the result. The number and type of simulations used to verify a given circuit is not regulated by any quantitative measures. It is difficult to communicate the verification effort and completeness to managers or other team members. All of these issues often result in costly design respins. For cutting edge analog circuits, it is not uncommon to see five to six respins which is incredibly expensive and makes many complex designs unprofitable. Automated verification tools bring the promise of allowing the designer to specify the design constraints in a standard format then letting a tool run the needed simulations and automatically check that the design constraints are satisfied, provide quantitative measures of verification quality understandable by team leaders and other designers, and decrease the number of respins required. The tools are not meant to replace the designer but instead automate standard parts of the design process enabling the designer to spend more time doing actual design. Moving forward, the two primary challenges for AMS verification are creating abstract models for AMS circuits and developing efficient, automated verification methodologies. These new solutions cannot come quickly enough as illustrated by a flamboyant quote from Sandipan Bhanot, CEO of Knowlent, “If the digital designers did verification the way analog designers do verification, no chip would ever tape out” [148].

1.2 AMS Circuit Modeling

Having access to models at many levels of abstraction is an important component of an efficient verification strategy. Given several levels of abstraction, the verification engineer can verify properties at the appropriate level of abstraction thus increasing the efficiency of verification. Digital systems typically have RTL-level, switch-level, transistor-level, and layout-level models available. Increasingly, digital designs also have word-level and transaction-level models available to verification engineers. In the digital domain, these models also have the advantage that when moving from one level of abstraction to the next, the model is often automatically generated and provably equivalent to the model in the next level of abstraction. In contrast, AMS circuits have transistor-level and layout-level models available. There are instances when an RTL-level model is available to the designer, but the RTL-level model is not automatically generated nor provably equivalent to the transistor-level model. AMS RTL-level models are manually created by designers who are expert in both AMS design and AMS *hardware description language* (HDL) creation. It should be noted that within the AMS modeling community, the term RTL-level model is not used. The term *macromodel* is used to mean a simplified model that captures only the essential behavior—but not necessarily the implementation—of the circuit for the given application.

The AMS modeling community understands these issues and has taken steps to improve the situation. The creation of AMS extensions to common digital HDLs in the form of Verilog-AMS [2] and VHDL-AMS [85] is a positive step. However, much of the work done by the AMS modeling community is still tied to SPICE-accurate abstract modeling. The SPICE simulator uses device-level transistor models (e.g., BSIM3) to simulate the behavior of individual transistors with high accuracy to the physical world. Significant effort is involved in creating accurate device models. Designers depend on and are accustomed to the accuracy of these models. As a result, the AMS modeling community strives to produce abstract models that accurately mimic the device-level model behavior. While this level of accuracy is comfortable for the designers, it is not always the best level of abstraction for verification. For instance, RTL-level models cannot be used to verify timing sensitive properties, but they can be used to verify much of the general functionality. Our work proposes an efficient method to automatically generate abstract AMS circuit models. These models may not accurately capture device-level behavior but do capture system-level behaviors.

1.2.1 Linear Systems

Macromodel creation for linear systems is a well understood discipline with a strong foundation [130, 123, 13, 129]. The majority of macromodeling methods for linear systems involve *model order reduction* (MOR) where the dimensionality of the state space or order of the model is reduced while accurately maintaining the important system dynamics. These methods have been successful because the MOR methods employed substantially reduce the order of the system. While the methods have been successful and are used to analyze systems such as networks of interconnect, there are still issues to be resolved namely in regard to efficiency, stability, and extensions for parametrized models [45, 147].

1.2.2 Nonlinear Systems

Unfortunately, any system containing a transistor is nonlinear, and creating macromodels for arbitrary nonlinear systems is far from a solved problem. There are promising solutions for weakly nonlinear system [35, 99, 128] and specific circuits such as PLLs and oscillators [98, 159] although these solutions do not work with a standard SPICE simulator. Investigations into the general macromodeling problem for nonlinear circuits have encountered scalability, efficiency, and global accuracy problems [136, 135, 153, 154]. Specifically, the MOR methods applied to nonlinear circuits do not reduce the models as strongly as when these methods are applied to linear circuits. Furthermore, for nonlinear circuits, there is no guarantee that a reduced order model will require less simulation time. Many of the MOR methods for nonlinear systems involve characterizing the circuit for parts of the operation space [153, 135]. If the simulation using the reduced order model deviates too far from the characterized operation space, the model accuracy suffers dramatically. Steps have been taken to address this problem [154], but model generation efficiency remains an issue due to the number of simulations required to characterize the circuit.

1.3 AMS Circuit Verification

Traditional verification of AMS circuits is intimately connected to the design of AMS circuits. In contrast, digital designs have distinct design and verification phases where different engineers perform design and verification. AMS verification needs to move in this direction, but tool support is needed first. The verification community has produced a number of interesting solutions to aid in automating the AMS workflow [146, 164], but there is still significant work to be done. The remainder of this section discusses several formal and semiformal approaches to AMS circuit verification.

Formal methods have become essential to verify a number of scenarios that arise in digital design. These techniques are not directly applicable to AMS circuits due to the need to represent continuously varying quantities and high dimension nonlinear dynamics. AMS circuits are also very sensitive to environmental fluctuations which do not affect digital circuits as dramatically due to the digital abstraction. It is these challenges that have prompted the recent efforts by the AMS verification community.

1.3.1 Automated Theorem Proving

Automated theorem provers are software systems developed to automatically or more often with the aid of a user, devise deductive proofs for mathematical theorems. These tools are notoriously difficult to operate by the novice user, but they are used in industry to verify specific circuit types such as floating point arithmetic units. Due to the ability of these tools to reason about linear arithmetic on real numbers, there have been several attempts to use these tools to verify properties of AMS designs.

In [60], Gosh et al. use the PVS proof checker to verify that synthesized circuits conform to a user provided behavioral description of the DC and small signal behavior of the circuit. This technique can be applied to designs done using traditional analog design methodologies but requires that the designer provide a piecewise linear behavioral description of the circuit as well as a structural specification. This paper presents results for three different designs containing small numbers of op amps, resistors, and transistors.

Hanna has explored modeling the nonidealities of digital circuits without the traditional binary abstraction [72]. The specification of the correct analog-like response of the circuit to an input sequence is conservatively encapsulated in a rectilinear region of space. This specification is verified against the implementation using proof techniques. Hanna extends this work in [73] where the analog nature of the circuits can be specified using a combination of piecewise linear or rectilinear specifications. This new work also moves away from the theorem proving approach and proposes the use of constraint satisfaction techniques to improve efficiency and automation.

In [4], Al-Sammane et al. propose a symbolic verification methodology for AMS circuits that given a description of the circuit and a set of properties can extract a set of recurrence equations. The computer algebra system Mathematica is used to prove the properties about the recurrence relations using an induction verification strategy. In this work, the AMS circuit is described using differential algebraic equations for the analog parts of the circuit and an RTL-level description of the digital parts of the circuit. This methodology is used to prove the stability of a third order $\Delta\Sigma$ modulator.

A method for verifying custom SRAM and flash memories using the ACL2 theorem prover [92] is described by Ray and Bhadra in [133]. The authors develop behavioral descriptions in the form of finite state machines (FSMs) for generic memory components such as the bitcell and sense amp. These component FSMs are composed to form an entire memory system. ACL2 is used to prove properties about the entire memory system.

1.3.2 Model Checking

Model checking [40] is the process of automatically determining if a model conforms to a given property. This is done via state space exploration or reachability analysis. Although model checking has been reasonably successful for digital circuits [41, 93, 131], there has been very little work on model checking for AMS circuits until recently. One of the major challenges in model checking AMS circuits is that continuous values such as voltages and currents must be tracked accurately complicating an already expensive state space exploration process. As a result, the primary difference in the model checking work for AMS circuits is the way that the different methods represent the state space. A secondary difference between the methods is the model used. Any model used for AMS verification must be able to model both continuous and discrete dynamics as well as the interactions between the two. The types of continuous dynamics allowed vary from method to method due in large part to the types of continuous dynamics supported by the state space exploration algorithms. An expanded discussion of model differences can be found in Section 2.2.

One of the first works to apply model checking to analog circuit models is the work by Kurshan and McMillan [96]. Finite state models are abstracted from analog circuit models using homomorphic (behavior-preserving) transformations. Given the finite state model of the analog circuit, standard finite state model checking techniques can be used to prove properties about the analog circuit of interest. The complexity of this method comes in reducing the continuous models to the finite state models. The reduction method involves dividing the state space into boxes and then integrating the nonlinear circuit model for fixed time steps to determine the possible transitions between the boxes. The applicability of this work to analog circuits is demonstrated by verifying a Seitz arbiter.

The work by Greenstreet in [65, 66, 67, 160] improves upon the work in [96]. In [65], a method to map continuous trajectories onto discrete behaviors using topological properties of the continuous model is presented. This method is superior to the homomorphic reductions used by [96] because it avoids the potentially negative interactions between box size and step size that can reduce model quality. The work in [66, 67, 160] incrementally improves upon the the state space representation method eventually settling on the *projectagon* representation presented in [160]. The verification of a high speed toggle circuit requiring seven dimensions is verified which demonstrates the efficiency and scalability of the projectagon state space representation.

Using a similar idea to Kurshan, McMillan, and Greenstreet, the work by Hartong, Hedrich, and Barke creates a Boolean abstraction for the system and then uses standard digital verification methods to do the analysis [74, 75]. This Boolean abstraction is created by automatically partitioning the continuous state space into boxes and treating each box as a discrete state. This partitioning method uses heuristics to create more boxes in the nonlinear regions to capture the correct behavior while reducing the number of boxes in linear regions to improve runtime. Each box is encoded using a Boolean encoding to create the Boolean abstraction used for model checking. The transitions between the boxes are found by selecting points within the box and using simulation to determine which boxes are reachable from the selected points. Properties are specified using basic CTL with additional greater than and less than operators. This approach is implemented in the *Amcheck* tool [76] and used to verify a tunnel diode oscillator circuit. Hendricx and Claesen have also applied Boolean based approaches to AMS verification with similar success [80].

Although Krogh, Chutinan, Silva et al. designed CheckMate to verify hybrid systems [42, 145, 144, 37], it has been applied to verify AMS circuits [71]. CheckMate follows an approach similar to [74, 75], but uses flowpipe approximations, which are sequences of polyhedra that follow the natural contour of the vector field [39], to create a sound abstraction of the continuous dynamics. Using the flowpipes method, the state space is partitioned dynamically as it is explored. In this way, only reachable portions of the state space are partitioned. The transitions between the partitions are constructed from the flows. The CheckMate tool is implemented within MATLAB using `Simulink` and `Stateflow` blocks to specify the design and properties. CheckMate properties are specified using ACTL or by labeling the appropriate states in the finite abstraction as *reach* or *avoid* states. The tool has been used to verify a tunnel diode oscillator and a $\Delta\Sigma$ ADC.

Frehse's `PHAVer` model checker extends the work done by CheckMate. It analyzes linear hybrid automata models of AMS circuits using convex polyhedra to represent the continuous state space [55, 57, 58]. To improve the approximations of the continuous trajectories, `PHAVer` uses a combination of forward and backward reachability [56]. While these polyhedra can still become quite complex, one unique feature of `PHAVer` is that it allows for performance to be tuned at the expense of a conservative state space. `PHAVer` demonstrates the effectiveness of this approach on several benchmark examples including the verification of the amplitude and phase jitter of a tunnel diode oscillator and a switched buffer network.

The `d/dt` tool designed by Asarin, Dang, and Maler was also originally developed to verify hybrid systems [16]. Recent work [43] extends the reachability analyses for linear [14] and nonlinear [15] ordinary differential equations to differential algebraic equations using methods similar to those in [96, 75]. The reachability computation employs convex polyhedra computed using *face lifting* [44]. This method uses optimal control instead of reachability analysis to verify bounded safety properties for both a low pass filter and $\Delta\Sigma$ ADC. Recent work by Girard extends this tool to work with *zonotopes* for more efficient reachability analysis [61].

Zaki et al. present a refined approximation for the reachability calculation by using Taylor approximations over interval domains [163]. They use this technique along with symbolic analysis to do bounded model checking of safety and liveness properties.

Walter et al. have developed *labeled hybrid Petri net* (LHPN) analysis tools using *binary decision diagrams* (BDDs) [156] and an *satisfiability modulo theories* (SMT) solver [155]. The LHPNs are translated to a symbolic model that is used to verify safety properties specified in TCTL using a BDD engine or a bounded SMT engine. The BDD algorithm is theoretically exact, but due to the high memory requirement of adding transitivity constraints the algorithm can be made conservative and more efficient by reducing the number of transitivity constraints added. The SMT based engine uses bounded model checking to verify properties on bounded length traces. The algorithm is exact for a given trace length.

1.3.3 Equivalence Checking

Equivalence checking is the process of exhaustively proving that two different system models are equivalent. In most cases, the two system models are at different levels of abstraction. Equivalence checking has seen widespread adoption with digital systems due to its efficiency and ease of use. Again, the digital abstraction dramatically decreases the level of complexity for the method. In digital designs, proving equivalence can be as simple as proving that two systems implement the same Boolean function. The meaning of equivalence for two analog circuits is less clear.

In [24], Balivada et al. describe an equivalence checking approach that involves comparing the transfer function of the circuit implementation against the transfer function of the circuit specification for linear circuits. The transfer functions are extracted and then transformed into the discrete Z-domain where they can be represented in terms of digital adders, multipliers, and delay elements. The transient behavior of these two digital descriptions are compared using BDDs and considered equivalent if they produce the same results within a given tolerance. The key to this method is the transfer to the Z-domain where it is critical to preserve the differences between the two specifications. This work is extended by Seshadri et al. in [142] to verify the conformance between the two transfer functions which avoids the problematic transfer to the Z-domain.

An approach based on sampling the state spaces and computing the differences in the vector fields is presented by Hedrich and Barke in [77]. Direct comparison of the vector fields is usually not possible, so nonlinear transformations are applied to the sample state spaces to enable this comparison. The difference between the two designs is presented as an explicit error measure. Finding correct transforms is nontrivial, so heuristics are presented to select the transforms. Unfortunately, poor transforms affect the soundness of the methodology.

Using interval arithmetic, Hedrich and Barke present a method to prove that linear circuits satisfy a specification in a given frequency range over a range of parameters [78]. The description of the linear analog circuits can be a transfer function or extracted from a netlist using symbolic analysis methods. Equivalence checking is done by testing the inclusion of the value sets of the transfer functions for the specification and the implementation. To maintain the soundness of the verification, an over-approximation is selected for the implementation and an under-approximation is selected for the specification.

An equivalence checking approach to verify VHDL-AMS designs is presented by Salem in [141]. This methodology involves using traditional digital equivalence checking, a set of rewrite rules, and simulations to check the equivalence of simple VHDL-AMS specifications. The designs are partitioned into digital and analog components. The digital components are verified using traditional digital equivalence checking. The analog components are simplified using rewrite rules and equivalence checked using pattern matching. To enhance the methodology, comparators are added to the outputs of the system and verified for equivalence during simulation runs.

1.3.4 Lightweight Verification

The methods described previously in this section work well on small or heavily abstracted examples and have shown some promise to work on larger circuits. One challenge for these methods is the significant effort required to create an appropriate abstract formal model for each circuit of interest. These methods also suffer from high computation costs when analyzing the model. The more accurately the method explores the state space of the system, the more computationally intensive it is. In response to these challenges, there has been recent work in verifying formal properties within the framework of simulation.

Dastidar et al. generate a finite state machine (FSM) from a systematic set of simulation traces [46]. Their FSM includes currents, voltages, and time as state variables to generate an acyclic FSM. The state space of the system is divided into symmetric state divisions. After each delta time step, the current state of the simulator is determined and rounded to the center of the appropriate state division. The simulator is then started from this point and run for the next delta time step. This process continues until the global time reaches a user specified maximum. Properties are verified on the FSM model. These properties are specified in Ana CTL, a CTL-like temporal logic with specific extensions for specifying properties of analog circuits.

In [64, 54], Girard, Fainekos, and Pappas present methods to verify safety properties [64] and LTL specifications [54] for linear systems using a finite set of simulation traces. This is done by sampling the set of initial states then running a finite length simulation for each sample. Using bisimulation theory [62, 63], the robustness of the set of simulation trajectories with respect to the property can be bounded. If the robustness measure is high enough, then the property can be proven. If the robustness measure is not high enough, the method must refine the sampling of the initial states. Bounds for the maximum number of required simulations can be computed.

Drawing inspiration from [91], Donzé and Maler present a method to verify safety properties of dynamical systems using a finite number of simulations [53]. The algorithm uses a set of initial conditions and a set of bad states. The algorithm selects an appropriate sample point within the set of initial conditions and performs a transient simulation for a bounded time period. Based on the sensitivity analysis of the trace, the system can be declared safe or may require further simulations. The algorithm is guaranteed to terminate using a finite number of simulations.

The work presented by Nahhal and Dang uses coverage measures for hybrid systems to guide test generation based on the rapidly-exploring random tree algorithm [120, 119]. The goal of this work is to verify the correctness of high percentage of the reachable state space via simulations generated by the algorithm. Quantification of the exploration completeness is done using coverage metrics based on the star discrepancy measure from statistics.

The work by Nickovic, Maler, and Pnueli presents the Analog Monitoring Tool (Amt) and its underlying technology [106, 121, 107]. Amt checks STL/PSL specifications on simulation traces. The STL/PSL specification language is a combination of STL [105] and PSL [1] with extensions to support checking analog properties. Amt has been used in case studies to verify properties of flash memory [121] and DDR2 DRAM [88] with varying levels of success.

In [165], Zaki et al. propose a run-time verification methodology based on monitoring the behavior of analog circuits using interval analysis. Given the system description, its specification described by nonlinear differential equations, and timed CTL formulas, the authors build a timed automata monitor which can detect bad behavior within a specified period of the interval arithmetic simulations.

Al-Sammane et al. use PSL properties to monitor AMS designs [3]. The PSL properties are used to generate monitors for discrete-time designs. This limits the applicability of the method although the method does use unmodified PSL.

1.4 Contributions

The research in this dissertation describes improved tools and methodology for the verification of AMS circuits. There are five contributions: a new hybrid Petri net model, *labeled hybrid Petri nets* (LHPNs), capable of modeling AMS circuits; an automated abstract model generation technique for LHPNs, VHDL-AMS, and Verilog-AMS; a novel method called *warping* which enables *difference bound matrices* (DBMs) to represent LHPNs; model checking algorithms for LHPNs using DBMs; and application of a new *simulation aided verification* (SAV) methodology to academic and industrial benchmarks.

The first contribution is the development of a new hybrid Petri net model, LHPNs, capable of modeling AMS circuits [101]. Designers use several types of models for verification, but none of them are amenable to formal methods and easily generated from common representations used by AMS designers. The development of LHPNs provides a model expressive enough to model AMS circuits at several levels of abstraction yet simple enough to be analyzable by formal verification tools. The introduction of a VHDL-AMS to LHPN compiler that translates behavioral VHDL-AMS models to LHPNs creates a path to obtain formal models from a modeling formalism used by designers. Transformations from LHPNs with ranges of rates to LHPNs with constant rates enables the use of LHPN models by a wider range of analysis engines.

The second contribution is an automated abstract model generation technique for LHPNs, VHDL-AMS, and Verilog-AMS [103, 102, 100]. AMS designers do not traditionally use models amenable to formal analysis, so obtaining formal models for use by formal verification tools is a challenge. An automatic model generation methodology generates the needed formal models in the form of LHPNs from a set of simulation traces. This automatic model generation technique leverages previously executed simulations to quickly provide up-to-date formal models for verification. Abstract AMS models are also needed for use by designers in system-level simulations. Using the same model generation methodology, additional algorithms generate both VHDL-AMS and Verilog-AMS system-level models for use in system-level verification. These models do not achieve SPICE-level accuracy, but they are much more efficient. This efficiency enables system-level verification not possible using previous abstract models. Coverage metrics based on the value of each simulation quantify the quality of the resultant model.

The third contribution is a method called warping which enables DBMs, a restricted form of polygons developed for analysis of timed systems, to represent the state space of the LHPN model. The difficulty of representing LHPNs using DBMs is that DBMs do not support variables changing at nonintegral rates. Warping employs variable substitution followed by conservative encapsulation of the polygon produced by the variable substitution within a polygon that can be represented as a DBM.

The fourth contribution is a method for model checking LHPNs using DBMs [104, 101]. Due to the complexity of doing state space analysis of systems containing both continuous and discrete dynamics it is critical to have an efficient state space representation. Very efficient algorithms for state space exploration and model checking of LHPNs enable the analysis of system-level descriptions. To achieve efficiency, the algorithms are conservative which may result in false negative verification results. An error trace generation methodology enables designers to quickly examine verification failures for validity.

The fifth contribution is the application of the verification methodology to several academic and industrial benchmarks. These benchmarks demonstrate the applicability of the previously described methodology to AMS circuits. The academic benchmarks include several classical examples from hybrid systems theory as well as a tunnel diode oscillator and switched capacitor integrator example. Finally, verification results obtained for two industrial examples, a PLL phased detector and a CMOS ring oscillator with feedforward inverters, show the promise of this methodology.

1.5 Dissertation Overview

The remainder of this dissertation is divided into five chapters describing the contributions of this dissertation. Figure 1.2 is a block diagram for the *LHPN Embedded/Mixed-signal Analyzer* (`LEMA`) tool which implements the algorithms presented in this dissertation. The chapters describe the different components of `LEMA` and how they interact to form a SAV verification methodology for AMS circuits.

Chapter 2 describes the LHPN model. Several modeling formalisms for hybrid systems and AMS circuits are discussed with the aid of the switched capacitor integrator circuit example. This discussion is followed by a description of the LHPN syntax and semantics. A brief discussion of translation methods to LHPNs from VHDL-AMS follows. This chapter concludes with a discussion of LHPN transformations that enable the LHPN model to be analyzed by state space exploration engines supporting only constant rate changes. This description includes a discussion of limitations introduced by the transformation.

Chapter 3 describes the Model Generator block in Figure 1.2. Obtaining abstract

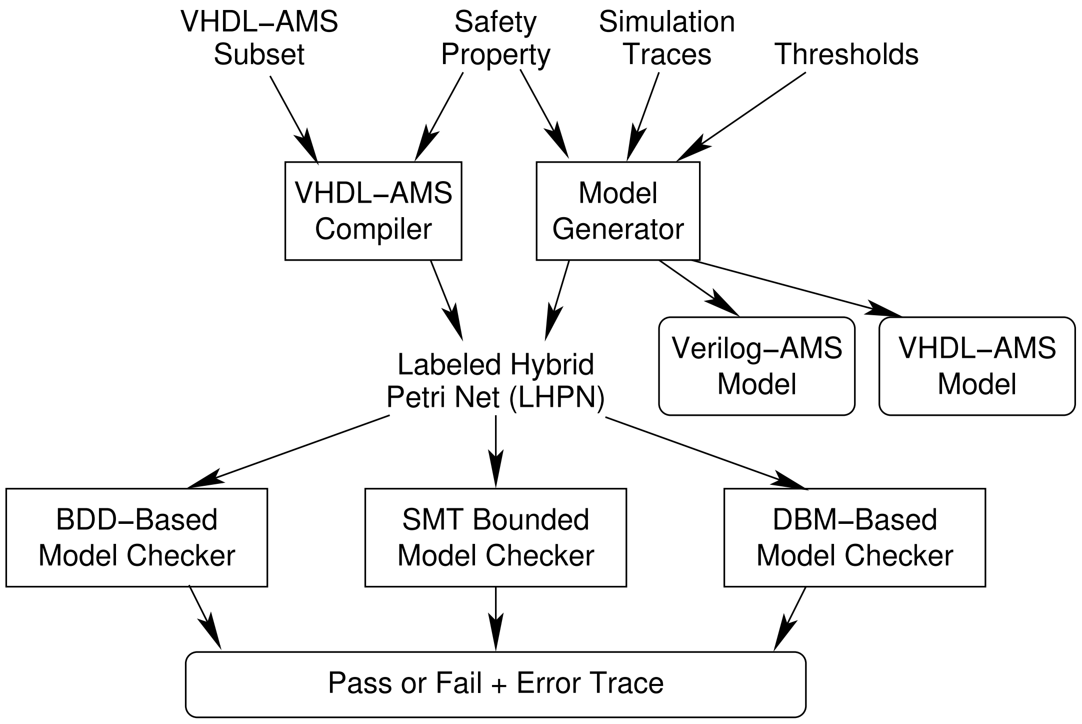


Figure 1.2. A block diagram describing `LEMA`.

models of AMS circuits for use in both formal verification and system-level simulation is critical to improving AMS verification. This chapter describes an automatic model generation method that uses simulation traces, thresholds on the signal levels of the system variables, and a safety property to generate Verilog-AMS, VHDL-AMS, and LHPN models of the circuit. The Verilog-AMS and VHDL-AMS models are intended to be used in standard simulators to perform highly efficient system-level simulation. The LHPN models are for use in formal verification. This chapter concludes with a description of the coverage metrics developed to aid in model quality quantification during the model generation process. These coverage metrics provide our SAV methodology with a quantitative measure of the value of each additional simulation to the model quality.

Chapter 4 introduces DBMs as a state space representation for LHPN reachability. A method called warping is presented that enables the use of DBMs to represent states of LHPNs that contain nonintegral rates.

Chapter 5 describes an efficient DBM-based state space exploration algorithm for LHPNs. DBMs have been used previously for timed circuit verification. This chapter explains extensions to timed circuit algorithms enabling the analysis of LHPNs where variables can change at nonintegral rates. This chapter concludes with a discussion of a method to generate error traces when a counterexample is found.

Chapter 6 discusses the performance of the SAV methodology presented in this dissertation on several industrial and academic examples. This chapter begins by examining the modeling and verification of several classical examples in hybrid systems theory. Next, two examples developed in academia are analyzed, a tunnel diode oscillator and a switched capacitor integrator. This chapter concludes with the analysis of two industrial examples, a PLL phase detector and a CMOS ring oscillator with feedforward inverters. These results highlight the progress that has been made as well as directions for future work.

Chapter 7 summarizes the contributions of this work. It discusses how the SAV methodology can be successfully integrated into an industrial AMS workflow. The dissertation concludes with a description of potential future work in AMS verification. There are several directions that can be undertaken with regard to model generation and abstraction, coverage metrics, an AMS property specification language, AMS circuit monitors, stability verification, improved performance of the state space analysis algorithms, and embedded software verification.

CHAPTER 2

LABELED HYBRID PETRI NETS

Traditionally AMS circuit designers have used SPICE-based models for design. SPICE models are based on differential equations which can accurately represent the continuous behavior of AMS circuits. While differential equations are amenable to simulation, they are not directly analyzable using state space exploration techniques. For this reason, a different modeling formalism is needed to represent AMS circuits when used in conjunction with formal verification tools.

This chapter introduces a switched capacitor integrator circuit that serves as a motivating example throughout this dissertation. In this chapter, the switched capacitor integrator circuit is used to compare and contrast several hybrid systems modeling formalisms. This chapter also introduces a new modeling formalism, *Labeled Hybrid Petri Nets* (LHPNs), which can be generated from VHDL-AMS descriptions using a compiler. An LHPN transformation method to enable analysis of LHPNs by state space exploration methods that only support constant rates concludes the chapter.

2.1 Motivating Example

To concretely illustrate concepts throughout this dissertation, the switched capacitor integrator circuit shown in Figure 2.1 is used as a motivating example. Switched capacitor integrator circuits function as a component in many AMS circuits such as ADCs and DACs. Although only a small piece of these complex circuits, the switched capacitor integrator proves to be a useful example illustrating the type of problems that can be present in AMS circuit designs. Discrete-time integrators typically utilize switched capacitor circuits to accumulate charge. Capacitor mismatch can cause gain errors in the integrators. Also, the CMOS switch elements in switched capacitor circuits inject charge when they transition from closed to open. This charge injection is difficult to control with any precision, and its voltage-dependent nature leads to circuits that have a weak signal-dependent behavior. This can cause integrators to have slightly different

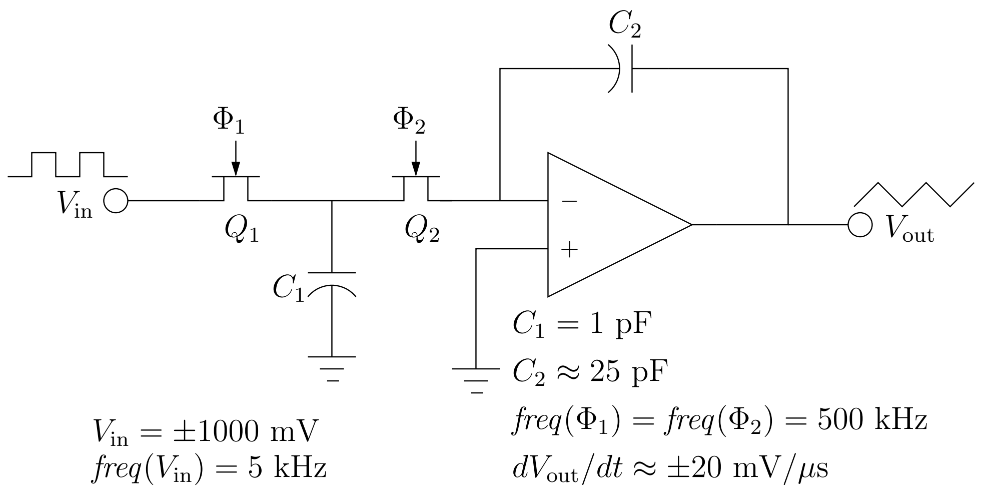

C_2
Φ_1
Φ_2
V_{in}
Q_1
Q_2
C_1
V_{out}
$C_1 = 1$ pF
$C_2 \approx 25$ pF
$freq(\Phi_1) = freq(\Phi_2) = 500$ kHz
$dV_{\text{out}}/dt \approx \pm 20$ mV/μs
$V_{\text{in}} = \pm 1000$ mV
$freq(V_{\text{in}}) = 5$ kHz

Figure 2.1. A schematic diagram of a switched capacitor integrator circuit.

gains depending on their current state and input value. Circuits using integrators run the risk of the integrator saturating near one of the power supply rails. It is essential to ensure that this never happens during operation under any possible permutation of component variations. Therefore, the verification property for this circuit is whether or not the output voltage, V_{out}, can rise above 2000 mV or fall below -2000 mV.

The input to the switched capacitor integrator circuit is a 5 kHz square wave with a low value of -1000 mV and a high value of 1000 mV. The circuit integrates this input square wave to produce a triangle wave at the output. For the given parameters, the output rate of change (slew rate) is ± 20 mV/μs. The op amp in the circuit is connected in an inverting configuration such that when V_{in} is positive, V_{out} is decreasing, and when V_{in} is negative, V_{out} is increasing. Transistors Q_1 and Q_2 in conjunction with capacitor C_1 form the switched capacitor part of the circuit. This switched capacitor component acts like a resistor. In fact, the behavioral models in this chapter make this simplification and replace the switched capacitor with a resistor. For this example, let us assume that due to noise and uncertainty in model parameters that the output slew rate has a variance of ± 10 percent (i.e., $\pm$(18 to 22) mV/μs). This circuit, therefore, must be verified for all values in this range [116].

To investigate the potential for failure using simulation, the designer would characterize the behavior of the circuit using the process described in Section 1.1. This characterization begins by running a simulation under nominal conditions. The simulation results under nominal conditions for the switched capacitor integrator circuit do not indicate a potential for failure as shown by the simulation trace in Figure 2.2. The characterizing simulations should also explore the global extremes of the variation or corner cases. A simulation for the circuit with a slew rate of ±22 mV/μs is shown in Figure 2.3, and a simulation with a slew rate of ±18 mV/μs is shown in Figure 2.4. Neither of these simulations shows the potential for error. An experienced analog designer would recognize the potential for charge injection and run random simulations to characterize the amount of charge injection. A random simulation for the switched capacitor integrator where the rate of change for V_{out} is varied randomly between ±18 to 22 mV/μs is shown in Figure 2.5. Although the waveform shows some fluctuation in the value of V_{out}, the results do not show a failure. A very specific simulation such as the one shown in Figure 2.6 is required to show the failure via simulation. In the failing simulation, the rate of change for V_{out} always increases faster than it decreases causing charge to build up on capacitor C_2, and the integrator to saturate.

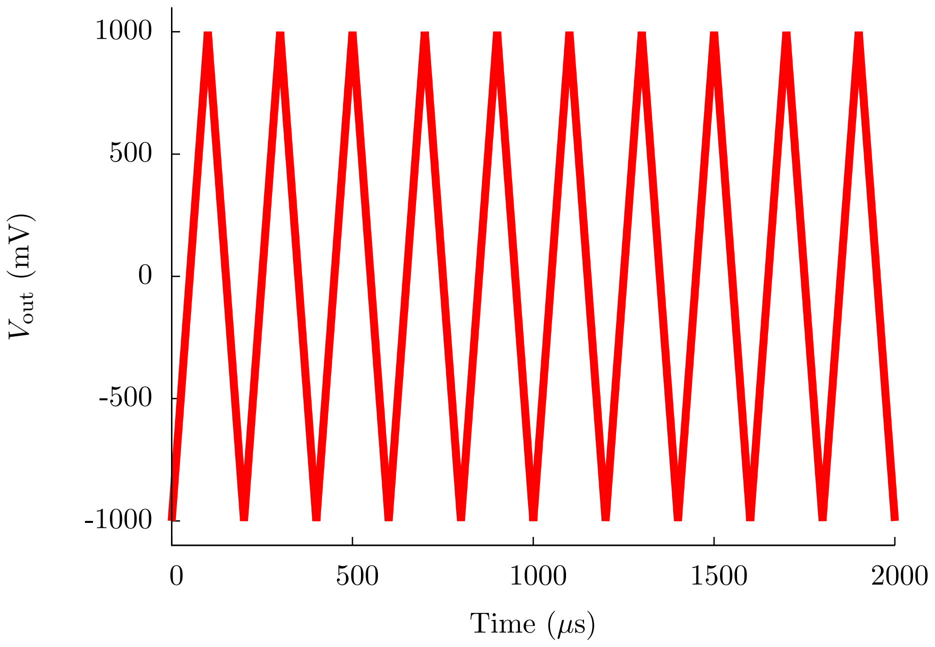


Figure 2.2. A simulation of the switched capacitor integrator circuit under nominal conditions, an output slew rate of ±20 mV/μs.

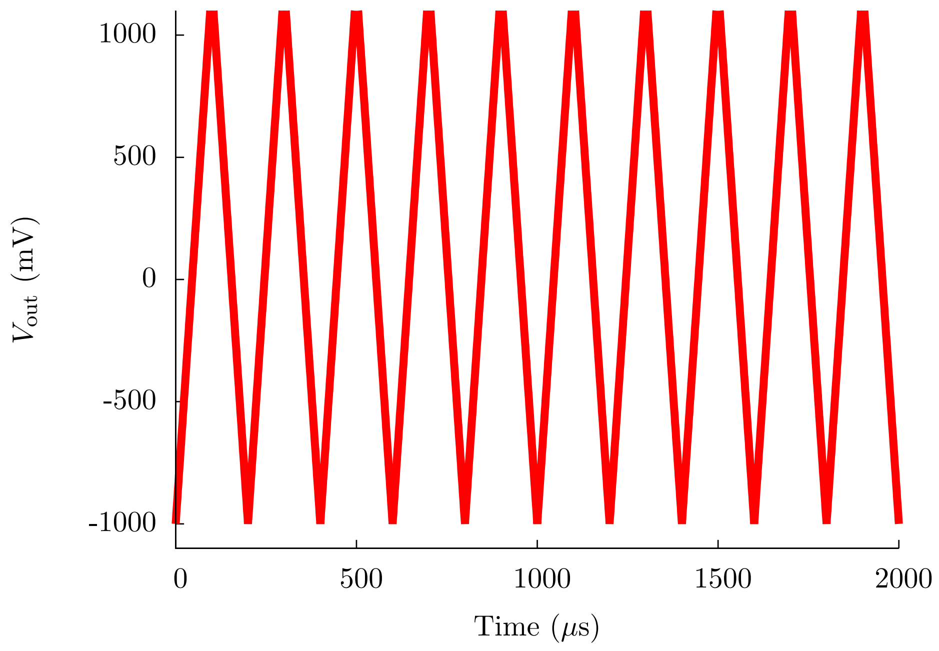


Figure 2.3. A simulation of the switched capacitor integrator circuit with an output slew rate of ±22 mV/μs.

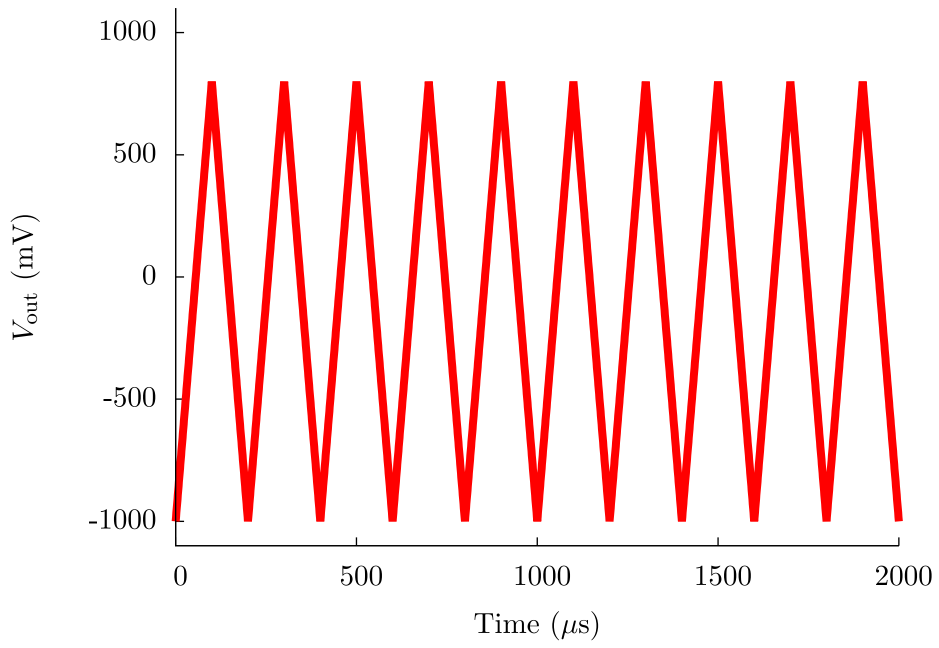


Figure 2.4. A simulation of the switched capacitor integrator circuit with an output slew rate of ±18 mV/μs.

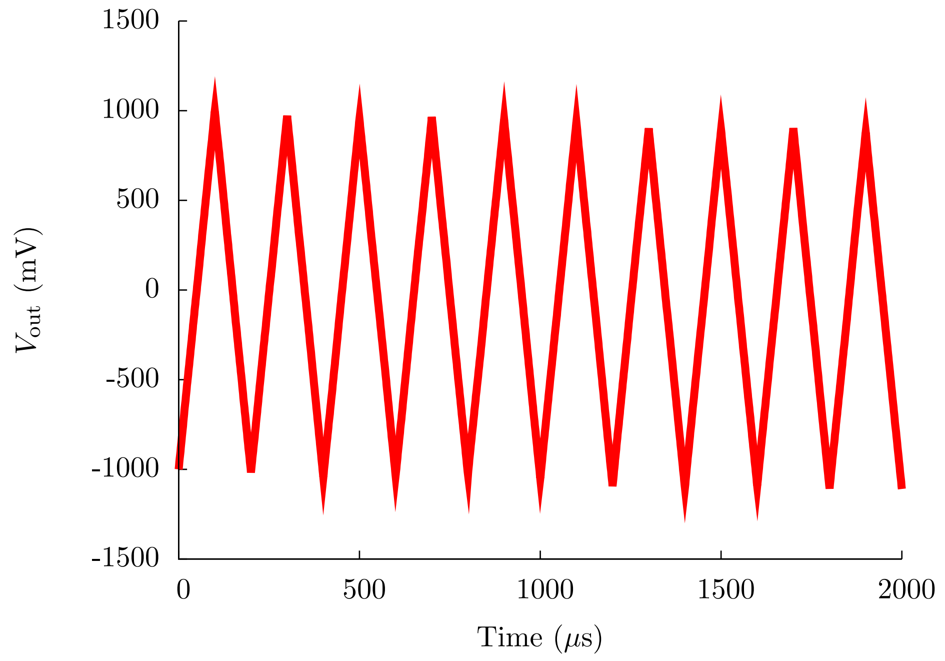


Figure 2.5. A random simulation of the switched capacitor integrator circuit with a variable output slew rate.

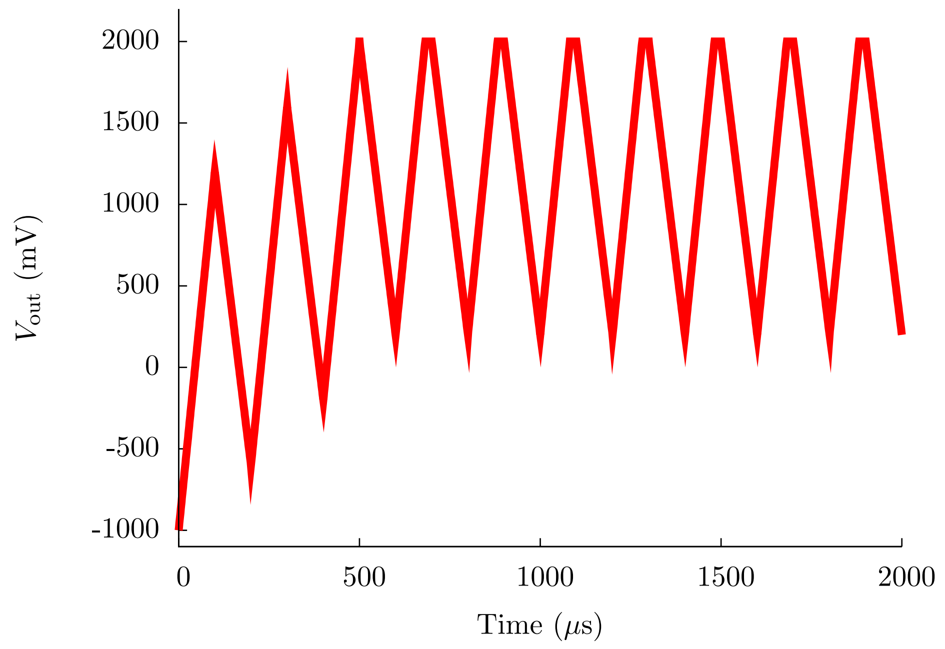


Figure 2.6. A worst case simulation of the switched capacitor integrator circuit showing saturation of the op amp.

2.2 Related Work

AMS circuits contain both digital and analog components. The digital components lend themselves to being modeled as a sequence of discrete events while the analog components are modeled as continuous flows. The class of systems that involve interactions between continuous and discrete dynamics are called hybrid systems. Several hybrid system models have been proposed and provide inspiration for the LHPN model presented in this chapter.

2.2.1 Hybrid Systems Modeling Languages

As hybrid systems (control systems, embedded systems, AMS circuits, etc.) become more common and complex, specialized modeling languages have been developed to support the design and verification of hybrid systems. Hybrid systems have the unique feature that continuous and discrete dynamics must interact. These modeling languages support the unique nature of this modeling and analysis problem.

CHARON [9] is a hybrid systems modeling language that supports hierarchical specifications and modular simulation. CHARON is designed to support state-of-the-art modeling concepts such as encapsulation, reuse, preemption, and hierarchy. The goal of the language is to be rich enough to support high-level modeling yet formal enough to support analysis. Although the language is full featured, it has not been adopted by industry.

The Mathworks tools `Simulink` and `Stateflow` are commonly used in industry for high-level modeling, simulation, and verification of hybrid systems. `Simulink` provides a graphical environment to model and design systems using a combination of library elements and custom blocks. `Stateflow` adds the ability to create complex state machines and flow charts that can be used in the `Simulink` simulation environment. While very popular, most models produced by `Simulink` are not well suited for formal analysis of hybrid systems although work has been done using `Simulink` as a frontend for the formal verification tool CHECKMATE [37].

VHDL is a commonly used digital *hardware description language* (HDL) [86]. Recognizing the need for modeling AMS circuits, extensions for AMS modeling were added to VHDL resulting in the VHDL-AMS language [85]. To enable the modeling of AMS circuits, VHDL-AMS adds the **quantity** annotation which is used to specify continuously varying variables represented by real numbers, and **break** statements are used to initialize these real quantities. Real quantities are assigned a rate of change using the **’dot** operator within simultaneous **if-use** and **case-use** statements. Conditions on real quantities are specified using **'above**. The rates of real quantities are updated using simultaneous statements such as the **if-use** and **case-use**. Figure 2.7 shows a VHDL-AMS description for the abstract behavior of the switched capacitor integrator. In our example, the Boolean signal `Vin` represents the input voltage while the continuous quantity `Vout` represents the output voltage. `Vout` is initialized to -1000 mV using a **break** statement. The Boolean signal `Vin` determines the rate of `Vout` using the **if-use** statements in our example. When `Vin` is 0, `Vout` increases at a rate between 18 and 22 mV/μs, and when `Vin` is 1, `Vout` decreases at a rate between -22 and -18 mV/μs. This is accomplished using the **span** procedure, defined in Section 2.4 as an extension to the `nondeterminism` package from [114]. This procedure accepts two real values and returns a random value within that range. In our example, the **process** statement controls `Vin`, and it uses the **assign** procedure from the `handshake` package [114]. This procedure performs an assignment to a signal and at some random time within a bounded range specified by its parameters and waits until the assignment has been performed before returning. The **assert** statement is used to specify the properties to verify. These properties can be arbitrary Boolean equations on the signal values and **'above** tests of continuous quantities. In our example, the **assert** statement checks if `Vout` falls below -2000 mV or goes above 2000 mV using the **'above** construct. Although **assert** statements only allow for the specification of time-domain safety properties, it should be noted that this includes bounded response time properties. Such properties can be specified by introducing a clock (a continuous variable that increases at rate 1), and asserting that some state is reached before the clock exceeds some value.

Verilog is another common digital HDL that has extensions for AMS circuits in the form of Verilog-AMS [84, 2]. Figure 2.8 is a Verilog-AMS description of the switched capacitor integrator circuit. Verilog-AMS adds a **real** data type to model continuous values. In the description of the switched capacitor integrator, both `Vin` and `Vout` are represented using real variables. `Vin` is an input to the system. `Vout` is calculated by the code and is an output of the system. The rate of `Vout` is also represented using a real variable, `Vout_rate`. Initialization of the real variables is done in an **initial_step** block which initializes `Vout` to -1 V and `Vout_rate` to 0.02 V/μs. In Verilog-AMS there are two types of conditionals for real quantities, **cross** and **if**. The language definition of the **cross** statement requires the simulator to adjust its time step to determine the exact crossing point of the conditional while the **if** statement has no such requirement.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.nondeterminism.all;
use work.handshake.all;
entity integrator is
end integrator;
architecture switchCap of integrator is
   quantity Vout:real;
   signal Vin:std_logic:='0';
begin
   break Vout=> -1000.0; --Initial condition
   if Vin='0' use
     Vout'dot == span(18.0,22.0);
   elsif Vin='1' use
     Vout'dot == span(-22.0,-18.0);
   end use;
   process begin
     assign(Vin,'1',100,100);
     assign(Vin,'0',100,100);
   end process;
  assert (Vout'above(-2000.0) and
     not Vout'above(2000.0))
    report "Error: The output voltage saturated."
    severity failure;
end switchCap;
```

Figure 2.7. A behavioral VHDL-AMS description of the switched capacitor integrator circuit.

The **cross** statement also specifies the direction of crossing. A '1' represents crossing from below the threshold, and a '$-$1' specifies a crossing from above the threshold. A '0' executes the block whenever the threshold is crossed regardless of the direction. The **timer** construct can be used to update the value of a continuous variable at a given time step. Real variables used outside the system must be output to the external system using the **transition** construct. The speed at which this transition occurs can be specified. This description uses these constructs to create a behavioral description of the switched capacitor integrator.

2.2.2 Automata Based Models

Timed automata [8, 5] are a mature modeling formalism with a strong theoretical background and adoption in niche industrial settings. There are several mature tools

```
`include "disciplines.h"
module swCap(Vin_io,Vout_io);
  inout Vout_io;
  electrical Vout_io;
  real Vin_var, Vout_var, Vout_rate;
  analog begin
    @(initial_step) begin
      Vin_var = -1.00;
      Vout_var = -1.00;
      Vout_rate = 0.020;
    end
    @(cross(Vin_var-0.0,-1)) begin
      Vout_rate = 0.020;
    end
    @(cross(Vin_var-0.0,1)) begin
      Vout_rate = -0.020;
    end
    @(timer(0.0,100e-06)) begin
      if (Vin_var > 0)
        Vin_var = -1.0;
      else
        Vin_var = 1.0;
    end
    @(timer(0.0,1e-06)) begin
      Vout_var = Vout_var + Vout_rate;
    end
    V(Vout_io) <+ transition(Vout_var,1p,1p,1p);
  end
endmodule
```

Figure 2.8. A behavioral Verilog-AMS of the the switched capacitor integrator circuit.

available for verifying systems modeled by timed automata [125, 162, 12]. While it is possible to model some hybrid systems using timed automata or extensions of timed automata [146], the model is obviously limited as only a single continuous quantity (time) is available. The model works well for real-time systems and a restricted class of hybrid systems but is not powerful enough to model hybrid systems in general.

Hybrid automata are a generalization of timed automata which allow each variable to have its own rate of change [7, 6]. The rate can be specified by any mathematical equation and may depend on the value or rate of other variables. This generality allows hybrid automata to accurately model even nonlinear hybrid systems. Due to this generality, hybrid automata are a popular model and are supported by tools such as HyTech [81, 10, 6], CHECKMATE [42, 145, 144, 37], `red` [158], `d/dt` [16], and `PHAVer` [55, 56, 57, 58]. However, the generality is a challenge because analyzing a very general model is computationally expensive. In fact, most tools only support a subset of the features of hybrid automata.

A *linear hybrid automata* (LHA) model of the switched capacitor integrator circuit is shown in Figure 2.9. This example helps illustrate the syntax and semantics of a restricted subset of hybrid automata, LHA. A formal definition of LHA can be found in [10]. LHA have two types of *data variables* $X = \{x_1, x_2, \ldots, x_n\}$ and $\dot{X} = \{\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n\}$ where $\dot{x}_i$ refers to the first derivative of x_i with respect to time. In Figure 2.9, $X = \{V_{out}, clk\}$; $\dot{X} = \{\dot{V}_{out}, \dot{clk}\}$; and the initial values for V_{out} and clk are -1000 mV and 0 μs, respectively. The vertices in an LHA graph are called *control locations*. Figure 2.9 has two control locations labeled **low** and **high**, and the location **low** is initially active. LHA have *transitions* to move between control locations. In Figure 2.9, there are two transitions: $t_1 = (\textbf{low}, \textbf{high})$ and $t_2 = (\textbf{high}, \textbf{low})$. LHA have invariants over the data variables in the system. If the automaton is in location v, $inv(v)$ may force a transition to occur by preventing time from progressing beyond a point in which $inv(v)$ is true. An example of an invariant in Figure 2.9 is $0\ \mu\text{s} \leq clk \leq 100\ \mu\text{s}$ in location **low**. This invariant has the effect of preventing clk from exceeding the value of 100 μs by stopping time or forcing a transition to be taken. Each variable in each control location v may have a range of rates of the form $\dot{x}_k := [l_k, u_k]$ which assigns an inclusive range of rational values between l_k and u_k to $\dot{x}_k$. When assigning a single rate, the abbreviated form $\dot{x}_k := a_k$ is used. For example, location **low** in Figure 2.9 has ranges of rates of $\dot{V}_{out} := [18, 22]$ mV/μs and $\dot{clk} = 1$. Each transition may have an action that is executed when the transition is taken. The action is a guarded command $act(t) = (guard(t) \rightarrow assign(t))$ where $guard(t)$ is a Boolean formula of predicates on the data variables and $assign(t)$ is a set of data variable assignments of the form $x_k := [l_k, u_k]$ which assigns an inclusive range of rational values between l_k and u_k to x_k. When assigning a single value, the abbreviated form $x_k := a_k$ is used. In Figure 2.9, the action for the transition between **low** and **high** is $clk = 100\ \mu\text{s} \rightarrow clk := 0\ \mu\text{s}$ where $clk = 100\ \mu\text{s}$ is the guard and $clk := 0\ \mu\text{s}$ is the assignment.

Formal semantics for hybrid automata are given in [10]. Intuitively, transitions in LHA are controlled by a combination of guards and invariants. While in a location, the data variables change at their specified rate as long as the invariant is satisfied. If progress would violate the invariant, time progression is halted. In Figure 2.9, beginning

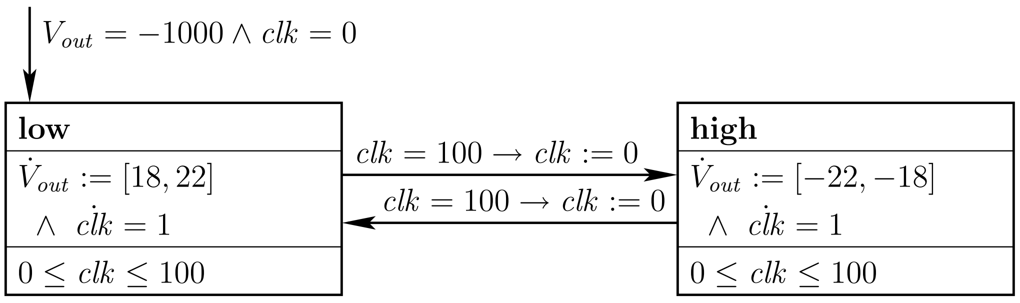

$V_{out} = -1000 \wedge clk = 0$
low
$\dot{V}_{out} := [18, 22]$
$\wedge\ \dot{clk} = 1$
$0 \leq clk \leq 100$
$clk = 100 \rightarrow clk := 0$
$clk = 100 \rightarrow clk := 0$
high
$\dot{V}_{out} := [-22, -18]$
$\wedge\ \dot{clk} = 1$
$0 \leq clk \leq 100$

Figure 2.9. A hybrid automata model of the switched capacitor integrator circuit.

in location **low** with V_{out} equal to -1000 mV and *clk* equal to 0 μs, V_{out} is increasing at a rate between 18 and 22 $mV/\mu s$ for 100 μs. Once *clk* equals 100 μs, the enabling condition on the transition between **low** and **high** becomes enabled and the transition occurs before *clk* increases beyond 100 μs which would violate the invariant. While in location **high**, V_{out} decreases at a rate been 18 and 22 $mV/\mu s$ until *clk* equals 100 μs. At this point, a transition into the **low** state occurs and the process repeats.

2.2.3 Petri Net Based Models

Time/Timed Petri nets (TPNs) [109] are Petri net based formalisms used to model timed circuits and systems. TPNs extend traditional *Petri nets* (PNs) [124] by adding timers to the transitions/places of the net. These formalisms are well researched and several tools have been developed to support analysis of TPNs [161, 115, 28]. Similar to timed automata, TPNs do not have the modeling power necessary to model hybrid systems but have provided inspiration for the development of *hybrid Petri net* (HPN) models.

While PNs require a discrete number of tokens be removed upon firing a transition, *continuous Petri nets* (CPNs) [47] allow a real valued number of tokens to be removed upon a transition firing. The notion of CPNs is extended to *timed continuous Petri nets* (TCPNs) which remove a real valued number of tokens after a delay. The TCPN formulation results in a model where tokens are removed at a given rate of change which is an important part of an HPN model.

The *hybrid net condition/event system* (HNCES) model [34] extends previously developed PN models. This model takes TPNs and CPNs and allows them to interact via condition and event signals. This is done by adding input and output conditions and events to a TPN or CPN. For example, a CPN may have a condition that triggers an output event when a continuous state reaches a given threshold. The conditions and events of the TPNs and CPNs are connected to form an HNCES. These connections between the discrete and continuous dynamics allow HNCES to model hybrid systems.

An HPN model using both PNs and CPNs is presented in [48]. This formulation allows transitions to be enabled or disabled by incoming arcs from both PNs and CPNs. The discrete part of the net influences the continuous part by enabling or disabling a continuous transition (flow) based upon its marking. Continuous quantities may also be used to enable or disable discrete transition firings. When the value of a continuous place reaches a specific value it enables the firing of a discrete transition. In this model, continuous markings may be converted into discrete markings and vise versa.

Another set of HPN models are based on stochastic Petri nets [36]. The two models, *fluid stochastic Petri nets* (FSPNs) [83] and *first-order hybrid Petri nets* (FOHPNs) [23], are very similar. Both models allow continuous and discrete places to be connected via transitions. The main difference between FSPNs and FOHPNs is that the rates in FSPNs are piecewise constant functions defined by the marking of the entire net, but the rates in FOHPNs are specified by a vector of constants.

During the evolution of our HPN model, we created *timed hybrid Petri nets* (THPNs) which are similar to HNCES. THPNs are described briefly using a THPN model of the switched capacitor integrator in Figure 2.10. For a complete description of THPNs see [104]. Similar to the HPN model in [48], THPNs are composed of a discrete PN and a CPN. Discrete places are depicted as circles, discrete transitions as solid boxes, and markings as solid circles (see discrete place pI and discrete transition $DecL$ in Figure 2.10 where pI is initially marked). When a discrete transition fires, it removes one token from each of the incoming places and adds one token to each of the outgoing places. Arcs from a discrete place to a discrete transition are also annotated with a bounded delay assignment (arc $(rP1, IncU)$ in Figure 2.10 has a bounded delay assignment of $[0, 100]$). Once the discrete transition becomes enabled, it fires between the lower and upper delay values. Graphically, continuous places are depicted as double circles, continuous transitions are depicted as empty boxes, and each continuous transition is annotated with its flow rate. For example, continuous place V_{out} is initially -1000 mV and continuous transition iL has a rate of 18 mV/μs in Figure 2.10. Continuous transitions fire continuously while they contain tokens. The flow rate of a continuous place is determined by adding the rates of all enabled incoming transitions and subtracting the rates of all enabled outgoing transitions.

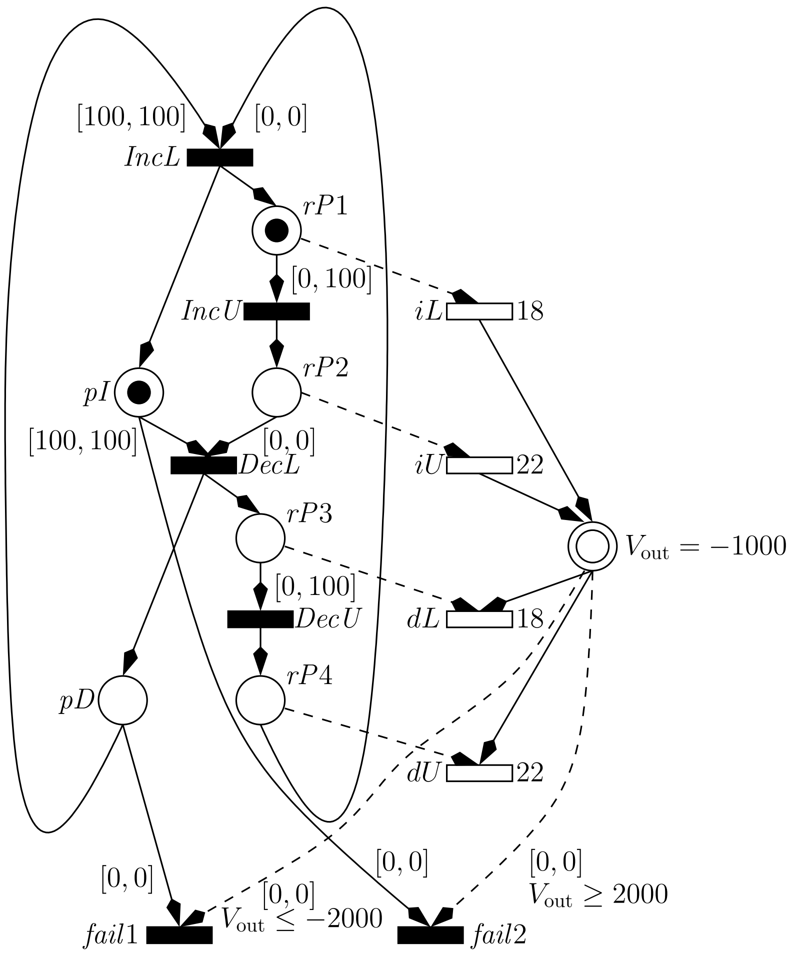


Figure 2.10. A THPN model of the switched capacitor integrator circuit.

The discrete PN and the CPN communicate via arcs denoted as dashed lines. An arc from a continuous place to a discrete transition is annotated with an enabling condition and a bounded delay assignment as demonstrated by (V_{out}, *fail*1) in Figure 2.10. The discrete transition fires between the lower bound and the upper bound of the delay after the enabling condition becomes true. An arc from a discrete transition to a continuous place enables the continuous transition when the discrete place is marked. For example, arc ($rP1$, iL), is initially enabled. Arcs from discrete transitions to continuous places add a single token to the continuous place when the discrete arc fires. Arcs from continuous transitions to discrete places are not allowed.

The semantics of THPNs are illustrated using the THPN for the integrator shown in Figure 2.10. In the initial state, V_{out} increases with a rate of 18 mV/μs. After a period of 0 to 100 μs, *IncU* fires which causes V_{out} to begin to increase with a rate of 22 mV/μs.

After 100 μs of V_{out} increasing, the transition *DecL* fires resulting in V_{out} beginning to decrease at a rate of 18 mV/μs which it does for 0 to 100 μs. When *DecU* fires, V_{out} begins to decrease with a rate of 22 mV/μs. Finally, after 100 μs of decrease, *IncL* fires returning the LHPN to its initial discrete marking. *fail1* and *fail2* are transitions used to remove tokens from places pI or pD when V_{out} exceeds 2000 mV or goes below -2000 mV. The firing of a *fail* transition causes the model to deadlock.

2.3 Labeled Hybrid Petri Nets: Syntax

This chapter presents a new HPN model, LHPNs, developed to represent AMS circuits and inspired by features in both hybrid Petri nets [48] and hybrid automata [7, 6]. While the hybrid automata model is probably the most common model used in hybrid system modeling and verification, we found it inadequate for our purposes. The creation of hybrid automata models is cumbersome and nonintuitive due to the guard/invariant interactions required to move from one control location to another. More importantly, it is difficult to automatically generate hybrid automata models from AMS-HDL code and simulation traces. HPNs and THPNs are also powerful enough to model AMS circuits and formal enough to be used for formal verification but are cumbersome to generate from AMS-HDL code and simulation traces. LHPNs are a model that is powerful enough to model AMS circuits, formal enough to be used for formal verification, and amenable to automatic model generation/compilation.

An LHPN is a tuple $N = \langle P, T, B, V, F, L, M_0, S_0, Q_0, R_0 \rangle$:

- P is a finite set of places;
- T is a finite set of transitions;
- B is a finite set of Boolean signals;
- V is a finite set of continuous variables;
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation;
- L is a tuple of labels defined below;
- $M_0 \subseteq P$ is the set of initially marked places;
- S_0 is the set of initial Boolean signal values;
- Q_0 is the set of initial ranges of values for each continuous variable; and
- R_0 is the set of initial ranges of rates for each continuous variable.

A key component of LHPNs are the labels. The labels permitted in LHPNs are represented using a tuple $L = \langle En, D, BA, VA, RA\rangle$:

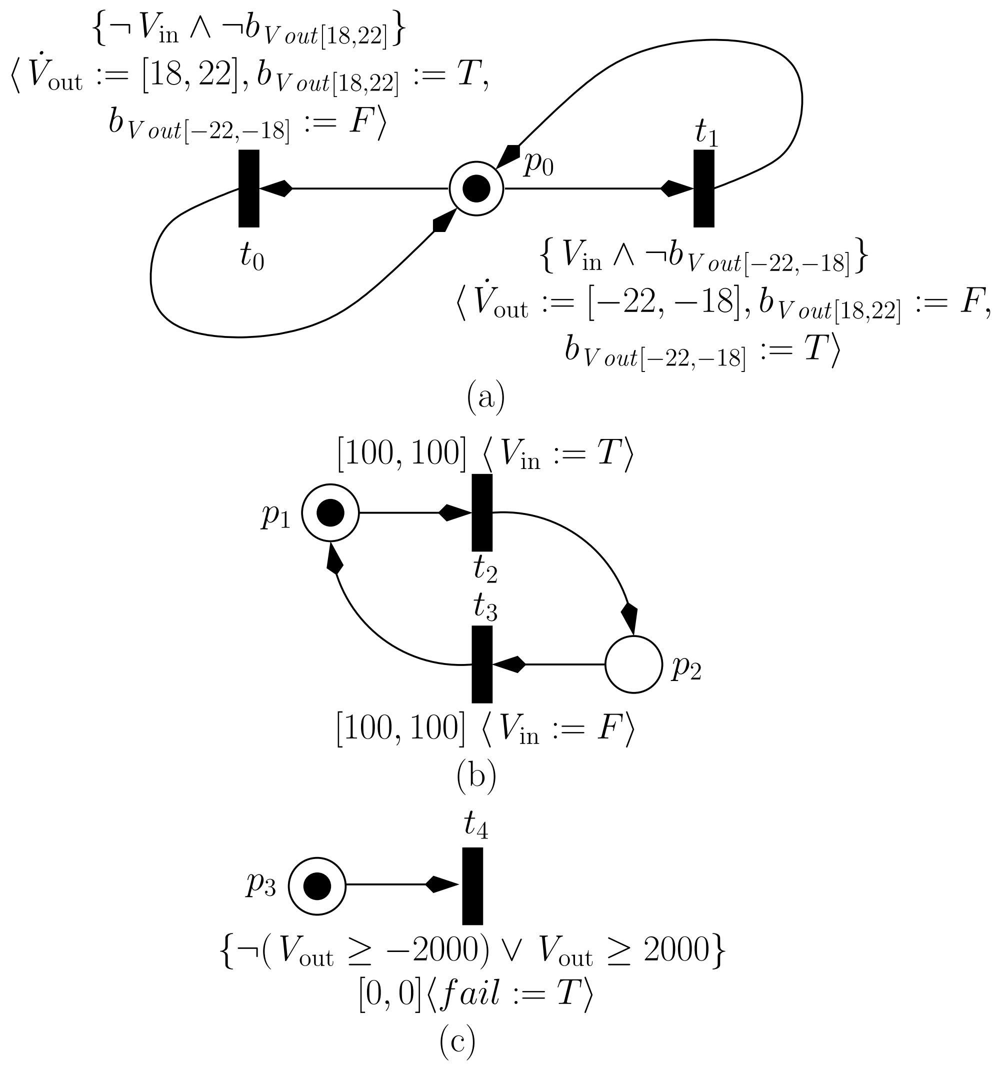
- $En : T \rightarrow \mathcal{P}$ labels each transition $t \in T$ with an enabling condition;
- $D : T \rightarrow |\mathbb{Q}| \times (|\mathbb{Q}| \cup \{\infty\})$ labels each transition $t \in T$ with a lower and upper bound $[d_{\mathrm{l}}, d_{\mathrm{u}}]$ on the delay for t to fire;
- $BA : T \rightarrow 2^{(B\times\{0,1\})}$ labels each transition $t \in T$ with Boolean signal assignments made when t fires;
- $VA : T \rightarrow 2^{(V\times\mathbb{Q}\times\mathbb{Q})}$ labels each transition $t \in T$ with continuous variable assignment ranges, consisting of a lower and upper bound $[a_l, a_u]$, that are made when t fires;
- $RA : T \rightarrow 2^{(V\times\mathbb{Q}\times\mathbb{Q})}$ labels each transition $t \in T$ with range of rates, consisting of a lower and upper bound $[r_l, r_u]$, that are assigned when t fires.

The enabling condition is defined using a restricted set of *hybrid separation logic* (HSL) formulas from the set $\mathcal{P}$ which are a Boolean combination of Boolean signals and separation predicates (inequalities relating continuous variables to constants). These formulas satisfy the following grammar:

$$\phi \quad ::= \quad \mathbf{true} \mid \mathbf{false} \mid b_i \mid \neg\phi \mid \phi \wedge \phi \mid v_i \geq k_i$$

where b_i is a Boolean signal, v_i is a continuous variable, and k_i is a rational constant in $\mathbb{Q}$. Since nonstrict inequalities are not supported by the DBM-based state space exploration analyzer presented in Chapter 4, the negation of $\geq$ inequalities represent $\leq$ inequalities.

Figure 2.11 shows an LHPN model for the switched capacitor integrator. The LHPN controlling the rate of change for V_{out} is shown in Figure 2.11a. This LHPN represents the behavior that relates the value of the input to the rate of change of the output. The LHPN in Figure 2.11a has a single place and two transitions. The LHPN is composed of a variable representing the digital value of V_{in}; a rate variable for V_{out}, $\dot{V}_{\mathrm{out}}$; and two rate Boolean signals representing the current rate of V_{out}, $b_{Vout[18,22]}$ and $b_{Vout[-22,-18]}$. The enabling condition on transition t_0 becomes true when V_{in} is **false**, and $b_{Vout[18,22]}$ is **false**. The assignment on the transition assigns the rate of V_{out} to $[18, 22]$ as well as the corresponding rate Boolean signals. As a result, transition t_0 changes the rate of V_{out} to $[18, 22]$ when V_{in} is low and the rate is not already set to $[18, 22]$. Transition t_1 is similar except it changes the rate of V_{out} to $[-22, -18]$ when V_{in} becomes **true**.



$\{\neg V_{\text{in}} \wedge \neg b_{Vout[18,22]}\}$
$\langle \dot{V}_{\text{out}} := [18,22], b_{Vout[18,22]} := T,$
$b_{Vout[-22,-18]} := F\rangle$
t_1
p_0
t_0
$\{V_{\text{in}} \wedge \neg b_{Vout[-22,-18]}\}$
$\langle \dot{V}_{\text{out}} := [-22,-18], b_{Vout[18,22]} := F,$
$b_{Vout[-22,-18]} := T\rangle$
(a)
$[100,100]\ \langle V_{\text{in}} := T\rangle$
p_1
t_2
t_3
p_2
$[100,100]\ \langle V_{\text{in}} := F\rangle$
(b)
t_4
p_3
$\{\neg(V_{\text{out}} \geq -2000) \vee V_{\text{out}} \geq 2000\}$
$[0,0]\langle fail := T\rangle$
(c)

$Q_0 = \{V_{\text{out}} = -1000\}$
$R_0 = \{\dot{V}_{\text{out}} = [18,22]\}$
$S_0 = \{\neg V_{\text{in}}, \neg fail, b_{Vout[18,22]}, \neg b_{Vout[-22,-18]}\}$

(d)

Figure 2.11. An LHPN model of the switched capacitor integrator circuit. (a) The LHPN controlling the rate of change for variable V_{out}. (b) The LHPN changing the Boolean signal V_{in}. (c) The LHPN checking for saturation. (d) The initial values for V_{out}, its rate of change, and the Boolean signals.

The input waveform for the switched capacitor integrator is a 5 kHz square wave. This square wave input is represented using the LHPN in Figure 2.11b. In this LHPN, delay bounds are used in conjunction with Boolean signal assignments to model the periodic square wave input. Transition t_2 waits for exactly 100 μs then sets the Boolean value of V_{in} to **true**. Transition t_3 also waits for exactly 100 μs then sets the Boolean value of V_{in} to **false**.

The LHPN representing the property to be verified is shown in Figure 2.11c. This LHPN is enabled by the complement of the property. In this case, it becomes enabled when V_{out} falls below -2000 mV or exceeds a value of 2000 mV. When this enabling condition is **true**, transition t_4 fires and sets the Boolean signal *fail* to **true** indicating a failure of the system. The final component to the LHPN model is the initial conditions found in Figure 2.11d. These indicate that the initial value, Q_0, of `Vout` is -1000, the initial rate, R_0, of `Vout` is $[18, 22]$, and the initial Boolean values, S_0, are **false** for `Vin`, *fail*, and $b_{Vout[-22,-18]}$ and **true** for $b_{Vout[18,22]}$.

2.4 Labeled Hybrid Petri Nets: VHDL-AMS Compiler

The LHPN model is designed to be readily generated from VHDL-AMS behavioral descriptions of AMS circuits. Using a subset of VHDL-AMS, AMS designers can create abstract behavioral descriptions of AMS circuits which can be formally verified using `LEMA`'s VHDL-AMS to LHPN compiler and one of our LHPN state space exploration engines. The VHDL-AMS to LHPN compiler uses a method similar to the one in [166]. Each discrete value is modeled using a **signal** of type **std_logic** and each continuous value is modeled using a **quantity** of type **real**. The initialization of discrete variables is done in the declaration while the initialization of continuous variables uses **break** statements. The assignments to discrete quantities are specified within **process** statements without sensitivity lists. A **process** statement can include **wait**, signal assignment, **if-then**, **case**, and **while-loop** statements. Signal assignment statements use the **assign** procedure defined in the the `handshake` package [114]. The **assign** procedure performs an assignment to a signal at some random time within a bounded range specified by its parameters and waits until the assignment has been performed before returning. The assignments of rates to real quantities uses the **'dot** notation within simultaneous **if-use** and **case-use** statements. The **span** procedure, defined in Figure 2.12 as an extension to the `nondeterminism` package from [114], assigns a range of rates to a continuous variable by taking two real values and returning a random value within that range. To test the current value of discrete quantities, **if-use** statements and **guard** procedures are used. The **guard** procedure, defined in the `handshake` package, checks the value of a Boolean signal. If the Boolean signal is already the specified value, the procedure returns. Otherwise, the **guard** procedure waits for the Boolean signal to become the specified value before returning. There are two variants of the **guard** procedure, **guard_or** and **guard_and**. The **guard_or** procedure takes a set of signals and values, and returns when one of the signals match the specified value. The **guard_and** procedure takes a set of signals and values, and returns when all of the signals match their specified values. The **'above** statement is used to test the value of real quantities.

Properties are specified using **assert** statements. These properties can be arbitrary Boolean equations on the signal values and **'above** tests of continuous quantities. Although **assert** statements only allow for the specification of time-domain safety properties, it should be noted that this includes bounded response time properties. Such properties can be specified by introducing a clock (a continuous variable that increases at rate 1), and asserting that some state is reached before the clock exceeds some value.

The LHPN in Figure 2.11 is automatically generated from the VHDL-AMS model of the switched capacitor integrator circuit in Figure 2.7. The **break** statement sets the initial value of `Vout` to -1000 mV as specified in Q_0 of Figure 2.11d. The declaration of `Vin` sets its initial value to **false** as specified in S_0 of Figure 2.11d. The **if-use** statement

```
impure function span(constant lower :  in real;
                     constant upper :  in real)
  return real is
  variable result :  real;
  variable tmp_real :  real;
  variable seed1 :  integer := 844396720;
  variable seed2 :  integer := 821616997;
begin
  uniform(seed1, seed2, tmp_real);
  result := (tmp_real * (upper - lower) + lower);
  return result;
end span;
```

Figure 2.12. VHDL-AMS code for the **span** procedure.

compiles into the LHPN in Figure 2.11a. The **if** clauses compile into enabling conditions. For example, t_0 is enabled by $\neg$`Vin` which corresponds to the **if** statement, and t_1 is enabled by `Vin` which corresponds to the **elsif** statement. The rate assignment of `Vout` is accomplished using rate assignments, on transitions t_0 and t_1. Transition t_0 assigns a rate of $[18, 22]$ which corresponds to the rate assigned when `Vin` is **false**, and transition t_1 assigns a rate of $[-22, -18]$ which corresponds to the rate assigned when `Vin` is **true**. The rate Boolean signals, $b_{Vout[18,22]}$ and $b_{Vout[-22,-18]}$, are added by the compiler to prevent vacuous firings of transitions t_0 and t_1. One rate Boolean signal is required for each rate assignment in the corresponding **if-use** statement.

The **process** statement compiles into the LHPN in Figure 2.11b. The timing bounds in the **assign** statement are compiled into a delay bound on the transitions, $[100, 100]$ on t_2 for example. The assignment in the **assign** statement translates into a Boolean signal assignment such as $\langle V_{\text{in}} := T \rangle$ on t_2. The assignments in the **process** statement are executed in a perpetual loop, so the corresponding LHPN is also a loop.

The **assert** statement compiles into the LHPN shown in Figure 2.11c. The transition t_4 becomes **true** when the complement of the assert statement is satisfied. When transition t_4 is enabled, it fires immediately, and sets the Boolean signal `fail` to **true** indicating a failure.

The items in the initial state not specified in declarations are derived from the initial state of the LHPN using the specified initial values. For instance, when considering the **if-use** statement if `Vin` is **false** the rate of change for `Vout` is $[18, 22]$. This is reflected in the initial state, R_0, where $V_{\text{out}} = [18, 22]$. Based on the initial rate assignment, the compiler derives the initial values for the rate Boolean signals.

2.5 Labeled Hybrid Petri Nets: Semantics

The state of an LHPN is defined using a 7-tuple of the form $\lambda = \langle M, S, Q, R, RR, I, C \rangle$ where:

- $M \subseteq P$ is the set of marked places;
- $S : B \rightarrow \{0, 1\}$ is the value of each Boolean signal;
- $Q : V \rightarrow \mathbb{Q}$ is the value of each continuous variable;
- $R : V \rightarrow \mathbb{Q}$ is the rate of each continuous variable;
- $RR : V \rightarrow \mathbb{Q} \times \mathbb{Q}$ is the current range of possible rates for each continuous variable;

- $I : v_i \geq k_i \rightarrow \{0,1\}$ is the value of each inequality;
- $C : T \rightarrow \mathbb{Q}$ is the value of each transition clock.

The current state of an LHPN can change via a transition firing or time advancement.

Every transition $t \in T$ has a *preset* denoted by $\bullet t = \{p \mid (p,t) \in F\}$ and a *postset* denoted by $t\bullet = \{p \mid (t,p) \in F\}$. A transition $t \in T$ is enabled when all the places in its preset are marked (i.e., $\bullet t \subseteq M$), and the enabling condition on t evaluates to true (i.e., $Eval(En(t), S, Q)$ where the function $Eval$ evaluates a restricted HSL formula for a given state). The function $\mathcal{EN}(M, S, Q)$ returns the set of enabled transitions for the given state. When a transition t becomes enabled, its clock is initialized to zero. The transition t can then fire at any time after its clock satisfies its lower delay bound and before it exceeds its upper delay bound (i.e., $d_l(t) \leq C(t) \leq d_u(t)$) as long as it remains continuously enabled.

From a state λ, a new state λ' can be reached by firing a transition t found in $\mathcal{EN}(M, S, Q)$. When a transition fires, the marking is updated by removing all tokens from places in the preset of t and adding tokens to all places in the postset of t. Boolean signal, value, rate, and rate range assignments associated with transition t are executed. The `span` function selects a value in the range of possible values.[1] The values of all inequalities are checked and updated accordingly based on changes that may have happened via the Boolean signal or value assignments. Clocks associated with newly enabled transitions are reset to 0. This new state is formally defined as follows:

- $M' = (M - \bullet t) \cup t\bullet$;
- $S'(b_i) = \begin{cases} s & \text{if } \exists (b_i, s) \in BA(t) \\ S(b_i) & \text{otherwise} \end{cases}$
- $Q'(v_i) = \begin{cases} \texttt{span}(a_l, a_u) & \text{if } \exists (v_i, a_l, a_u) \in VA(t) \\ Q(v_i) & \text{otherwise} \end{cases}$
- $R'(v_i) = \begin{cases} r_l & \text{if } \exists (v_i, r_l, r_u) \in RA(t) \\ R(v_i) & \text{otherwise} \end{cases}$
- $RR'(v_i) = \begin{cases} (r_l, r_u) & \text{if } \exists (v_i, r_l, r_u) \in RA(t) \\ RR(v_i) & \text{otherwise} \end{cases}$

[1]Note that using the `span` function to select a specific a value results in a less precise (and less complex) semantics than those described in [157]. Namely, the semantics disallow traces that may be explored using our DBM-based state space exploration engine. However, allowing for $[a_l, a_u]$ is easily handled in our DBM-based analysis method.

- $I'(v_i \geq k_i) = (Q'(v_i) \geq k_i)$
- $C'(t_i) = \begin{cases} 0 & \text{if } t_i \in \mathcal{EN}(M', S', Q') \\ & \wedge\, t_i \notin \mathcal{EN}(M, S, Q) \\ C(t_i) & \text{otherwise} \end{cases}$

In a state λ, time can advance by any value τ which is less than $\tau_{\max}(\lambda)$. The value of $\tau_{\max}(\lambda)$ is the largest amount of time that may pass before a transition is forced to fire (i.e., the clock associated with it exceeds its upper bound) or an inequality changes value (i.e., its continuous variable's value v_i crosses the constant k_i). It is defined as follows:

$$\tau_{\max}(\lambda) = min \begin{cases} C(t_i) - d_u(t_i) & \forall t_i \in \mathcal{EN}(M, S, Q) \\ \frac{k_i - Q(v_i)}{R(v_i)} & \forall v_i \geq k_i \in I. \\ & \ I(v_i \geq k_i) \neq (R(v_i) \geq 0) \end{cases}$$

When time advances by τ, the values of continuous variables are updated by adding the current value of the variable to τ multiplied by the current rate. The rate is changed to a potentially new rate between the lower and upper bound of the rate range. Inequalities are updated based on the new values of the continuous variables. If the newly updated inequalities have enabled transitions, the clocks for these newly enabled transitions are set to 0. The value of τ is added to the current value of all other clocks. The marking, Boolean signals, and rate ranges are unaffected. The new state after time advancement τ is defined formally as follows:

- $M' = M$
- $S' = S$
- $Q'(v_i) = Q(v_i) + \tau \cdot R(v_i)$
- $R' = \texttt{span}(r_l, r_u)$
- $RR' = RR$
- $I'(v_i \geq k_i) = \begin{cases} R(v_i) \geq 0 & \text{if } Q'(v_i) = k_i \\ I(v_i \geq k_i) & \text{otherwise} \end{cases}$
- $C'(t_i) = \begin{cases} 0 & \text{if } t_i \in \mathcal{EN}(M', S', Q') \\ & \wedge\, t_i \notin \mathcal{EN}(M, S, Q) \\ C(t_i) + \tau & \text{otherwise} \end{cases}$

The semantics of the LHPN model are illustrated using the LHPN example in Figure 2.11. In the initial state, p_0, p_1, and p_3 are marked; V_{in} and $b_{Vout[-22,-18]}$ are **false**;

$b_{Vout[18,22]}$ is **true**; V_{out} is -1000; and the rate of V_{out} is 18 to 22 mV/μs. The only transition enabled in the initial state is t_2 which limits the advancement of time to 100 μs due to its delay bound. This time advancement allows the value of V_{out} to be 800 mV if the rate of change for V_{out} is 18 mV/μs. Transition t_2 fires after 100 μs. The firing of t_2 removes the token from place p_1 and places it in place p_2 as well as assigning V_{in} to **true**. When V_{in} becomes to **true**, transition t_1 becomes enabled and fires in zero time changing the rate of V_{out} to -22 to -18 mV/μs and assigning $b_{Vout[18,22]}$ to **false** and $b_{Vout[-22,-18]}$ to **true**. At this point, the only enabled transition is t_3 which has a delay bound of $[100, 100]$. Therefore, time advances 100 μs before transition t_3 must fire. When t_3 fires, V_{in} is assigned to **false**. This enables t_0 to fire in zero time and change the rate of V_{out} to 18 to 22 mV/μs. Operation continues in this manner until the predicate on transition t_4 becomes **true**. At this point, transition t_4 fires setting the Boolean signal *fail* to **true** indicating a failure of the system.

2.6 LHPN Expansion for Constant Rates

The LHPNs generated from the VHDL-AMS compiler include ranges of rates. These LHPNs cannot be directly analyzed using `LEMA`'s DBM-based model checker described in Chapter 4. To enable analysis of these LHPN models using the DBM-based approach, a piecewise approximation of the range of rates is created by performing a transformation on the LHPN. The transformation of the LHPN can be performed on every LHPN generated by `LEMA`'s VHDL-AMS compiler and model generator. In general, there are restrictions on the types of LHPNs that can be transformed. There are two requirements for an LHPN to be transformed. The first requirement is that the rates for a given variable must only be assigned in one connected LHPN component. The second requirement is that the initial rate assignment must be consistent with a valid non-initial rate assignment for the initial marking. All LHPNs that satisfy these constraints may be transformed.

There are three cases. When the range of rates are nonnegative or nonpositive, this transformation initially sets the rate to the minimum of the absolute value of the rate bounds, $\texttt{min}(|r_l|, |r_u|)$. The rate is allowed to change to the maximum of the absolute value of the rate bounds, $\texttt{max}(|r_l|, |r_u|)$, at any time while the transition remains enabled. After the rate changes to its upper bound, it is not allowed to change back to its lower bound. Figure 2.13 illustrates three possible paths explored with this approximation using a variable whose rate of change is between 18 and 22 mV/μs and value is initially

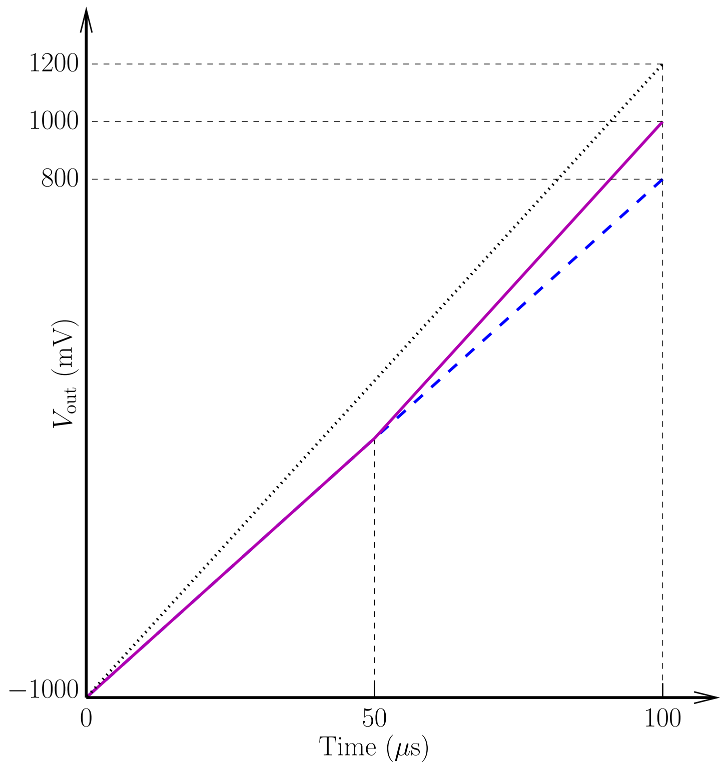
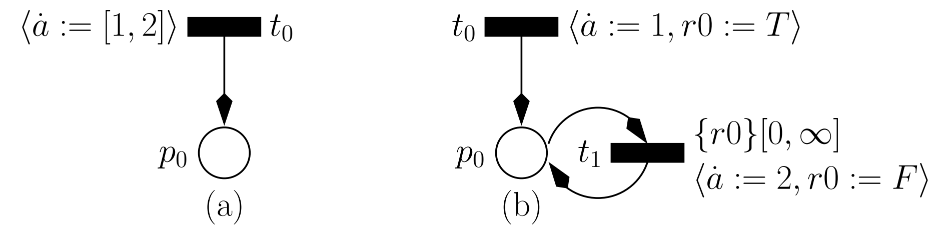


Figure 2.13. An example of the piecewise approximation used by LHPN expansion for nonnegative and nonpositive ranges of rates.

-1000 mV. The dashed line represents the path where the rate remains at the lower bound, 18 mV/μs, for the entire period. The dotted line represents the path where the rate immediately changes to the upper bound, 22 mV/μs. The solid line represents the path where the change to the upper bound of the rate happens at 50 μs. This approximation enables the analysis algorithm to represent all terminal paths in the model although it does not represent all intermediate paths. Note that nonpositive rate ranges are also handled in this way.

Figure 2.14 demonstrates how the transformation proceeds when the rate being transformed is either nonnegative or nonpositive. In this case, a single transition and Boolean signal are added, t_1 and $r0$. The original rate assignment is changed to the minimum of the absolute value of the rate bounds (i.e., $\langle \dot{a} := 1 \rangle$), and the associated Boolean signal is set to **true** on the originating transition (i.e., $\langle r0 := T \rangle$). The new transition, t_1, is enabled to fire any time after the Boolean signal $r0$ is **true**. When transition t_1 fires, the rate is set to the maximum of the absolute value of the rate bounds (i.e., $\langle \dot{a} := 2 \rangle$), and the Boolean signal $r0$ is set to **false** (i.e., $r0 := F$) to prevent vacuous firings of the transition.

Figure 2.15 illustrates why it is necessary to use the absolute value of the range of rates. In this case, if the absolute value is not used, a spurious deadlock results. An example where deadlock can occur is in Figure 2.15b. Deadlock occurs if transition t_0 fires and zero time later t_2 fires. When t_2 fires, the enabling condition $\{a \geq 5\}$ is still true



$\langle \dot{a} := [1,2] \rangle$
t_0
p_0
(a)
t_0
$\langle \dot{a} := 1, r0 := T \rangle$
p_0
t_1
$\{r0\}[0,\infty]$
$\langle \dot{a} := 2, r0 := F \rangle$
(b)

Figure 2.14. A template for transforming LHPNs where the range of rates are nonnegative or nonpositive. (a) Original LHPN. (b) Transformed LHPN.

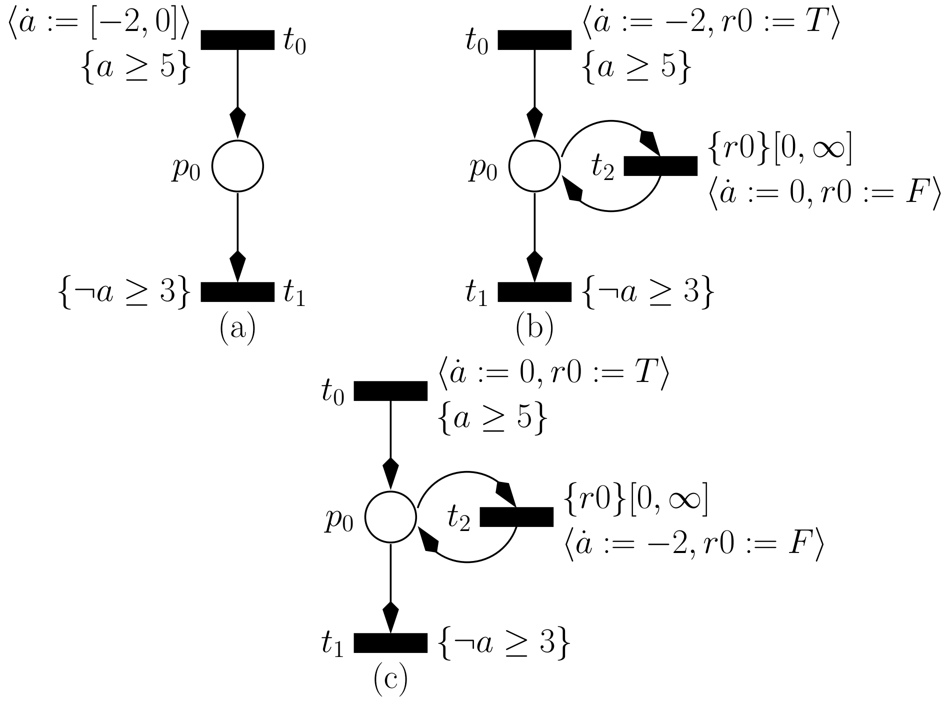


Figure 2.15. A template for transforming LHPNs where the range of rates is nonpositive. (a) Original LHPN. (b) Transformed LHPN without using absolute value before comparison. (c) Transformed LHPN using a correct transform.

because no time has passed. When t_2 fires, a stops moving which prevents the enabling condition $\{\neg a \geq 3\}$ on t_1 from being satisfied. From this state, the LHPN has no possible events resulting in a deadlock. The correct transformation in Figure 2.15c uses the upper bound of zero for the rate in t_0 (i.e., $\langle a := 0 \rangle$), and the lower bound for the rate assignment of t_2 (i.e., $\langle a := -2 \rangle$). Deadlock does not occur using this transformation. When t_0 fires, a is stopped and can remain stopped for an infinite amount of time. The difference from Figure 2.15b is that t_2 can fire and set the rate of a to -2 allowing a to decrease and eventually satisfy the enabling condition of t_1.

When rates span zero, the method used to achieve all possible final values sets the rate to zero initially. The rate is subsequently allowed to change to be either positive or negative but not both. Figure 2.16 illustrates the functionality of this method using a variable with a range of rates from -10 mV/μs to 10 mV/μs that is initialized to 0 mV/μs. If the rate changes to the negative bound of -10 mV/μs immediately, the most negative trace is produced. This is shown by the decreasing solid line that reaches -1000 mV in Figure 2.16. If the converse is true, and the positive rate change to 10 mV/μs occurs immediately the most positive possible value, 1000 mV, is reached as shown by the increasing solid line in Figure 2.16. If the rate stays at zero for a period of time then changes to either the positive or negative bound an intermediate trace value results. These intermediate traces are illustrated using the dashed lines in Figure 2.16. If the rate remains at zero for 50 μs then changes to 10 mV/μs a value of 500 mV is reached. If the rate remains at zero for 75 μs then changes to -10 mV/μs a value of -250 mV is reached. The final case in Figure 2.16 is when the rate does not change and a value of zero is reached as shown by the dotted line. Again all terminal paths are represented, but the transformed LHPN does not represent all intermediate paths.

Figure 2.17 depicts the second template which uses the method presented for rates spanning zero. This template requires the addition of two transitions and one Boolean signal, t_1, t_2, and $r0$, respectively. The rate of the originating transition, t_0, is set to zero (i.e., $\langle \dot{a} := 0 \rangle$), and the associated Boolean signal is set to **true** (i.e., $\langle r0 := T \rangle$). The rates of the two additional transitions are set to the lower and upper bounds, respectively. Each of the newly added transitions are enabled when the associated Boolean signal is **true** and set this signal to **false** upon firing to prevent both the positive and negative rate from occurring in the same trace.

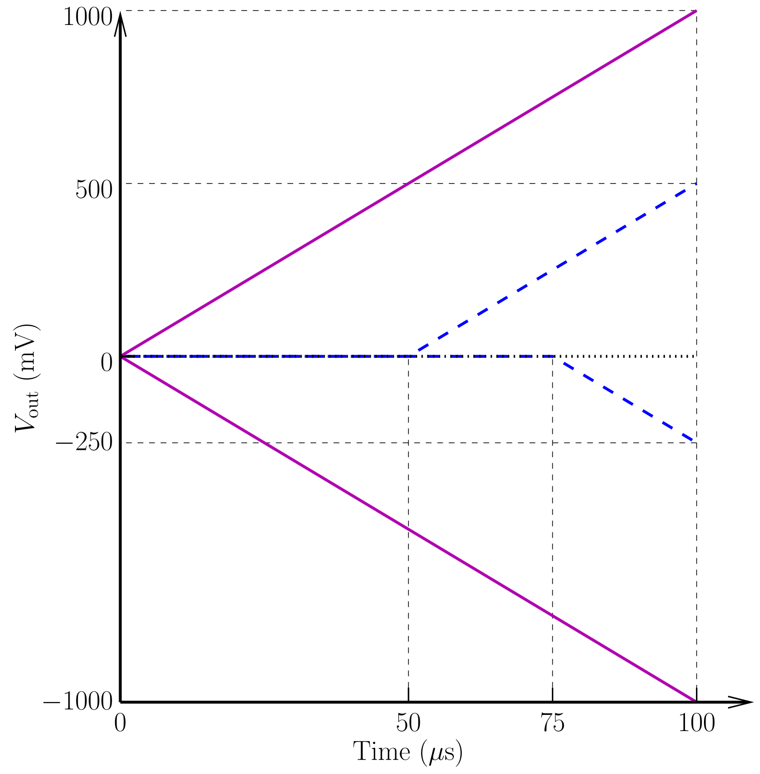


Figure 2.16. An example of the piecewise approximation used by LHPN expansion for ranges of rate that span zero.

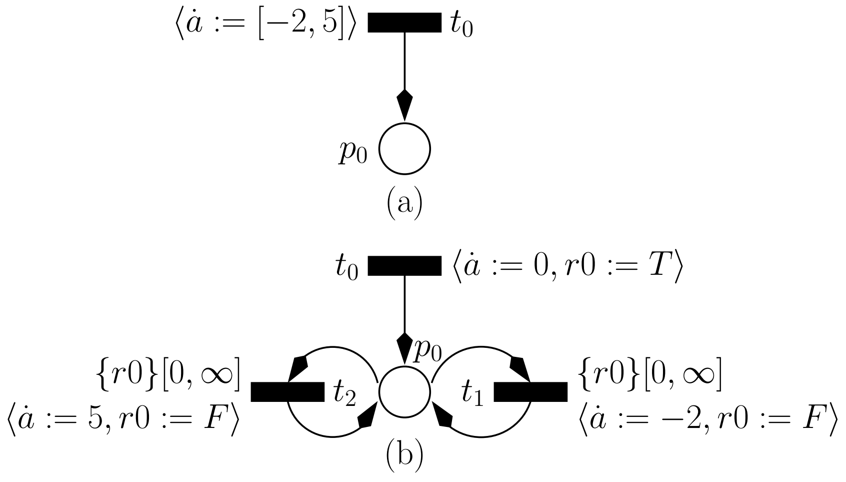

$\langle \dot{a} := [-2, 5] \rangle$
t_0
p_0
(a)
t_0
$\langle \dot{a} := 0, r0 := T \rangle$
p_0
$\{r0\}[0, \infty]$
$\langle \dot{a} := 5, r0 := F \rangle$
t_2
t_1
$\{r0\}[0, \infty]$
$\langle \dot{a} := -2, r0 := F \rangle$
(b)

Figure 2.17. A template for transforming LHPNs where the range of rates spans zero. (a) Original LHPN. (b) Transformed LHPN.

Figure 2.18 demonstrates how the transformation proceeds when the place has multiple different incoming rates for the same variable. Each transition is transformed separately using one of the three templates described previously. A different Boolean signal is used for the two transitions which enables the LHPN to track which transition firing caused the rate assignment, and therefore, which rate assignment is allowed in the current state.

Figure 2.19 depicts the third template which ensures that the rate can change even when an explicit assignment is not made. For every transition in a rate changing LHPN that does not have a rate assignment for each variable whose rate is changed in the LHPN, the rate from the previously made rate assignment is propagated forward. In Figure 2.19, transition t_1 does not have a rate assignment. In the transformed net, the rate from t_0 is propagated forward to allow a rate change to the upper bound in place p_1. In general, the additional transitions must be propagated along all paths in the LHPN until a new rate assignment is encountered for the given variable along each path.

The LHPNs in Figure 2.20 show the result of the LHPN expansion for constant rates applied to the LHPN in Figure 2.11a. In Figure 2.20a, the rate assignment on t_0 is changed from $[18, 22]$ to 18 in Figure 2.20b. Upon firing t_0, the Boolean signal $r0$ is set to **true** enabling the firing of t_5. The delay bound on t_5 is $[0, \infty]$ allowing t_5 to fire at any time in the future while the enabling condition remains satisfied. When t_5 fires it

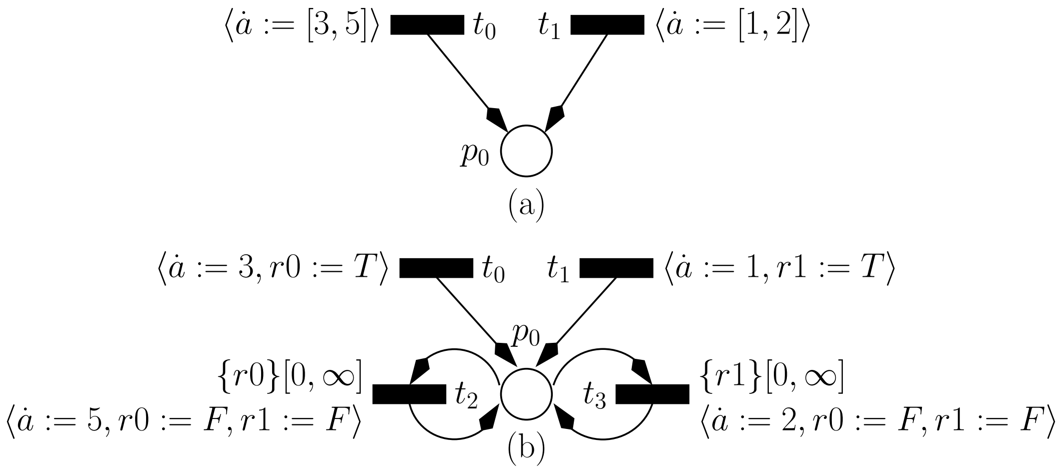

$\langle \dot{a} := [3,5] \rangle$
t_0
t_1
$\langle \dot{a} := [1,2] \rangle$
p_0
(a)
$\langle \dot{a} := 3, r0 := T \rangle$
t_0
t_1
$\langle \dot{a} := 1, r1 := T \rangle$
p_0
$\{r0\}[0,\infty]$
$\langle \dot{a} := 5, r0 := F, r1 := F \rangle$
t_2
t_3
$\{r1\}[0,\infty]$
$\langle \dot{a} := 2, r0 := F, r1 := F \rangle$
(b)

Figure 2.18. An example of an LHPN transformation where the LHPN has multiple different incoming rates for the same continuous variable. (a) Original LHPN. (b) Transformed LHPN.

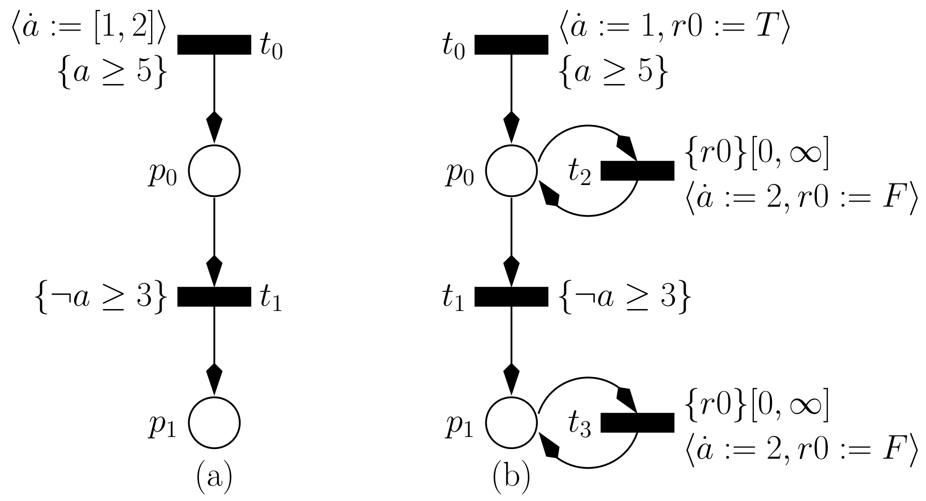
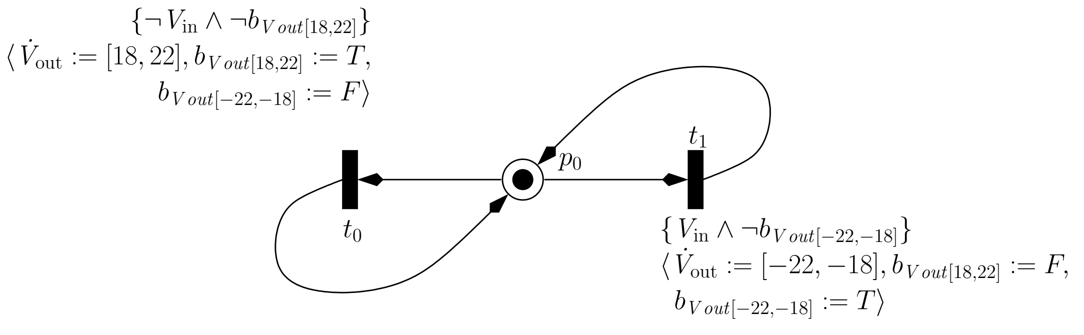


Figure 2.19. An example of an LHPN transformation where the LHPN does not have a rate assignment on every transition. (a) Original LHPN. (b) Transformed LHPN.

sets the rate to the upper bound, 22, and sets the Boolean signal $r0$ to **false** preventing vacuous rate assignments. A similar transformation is performed on transition t_1.

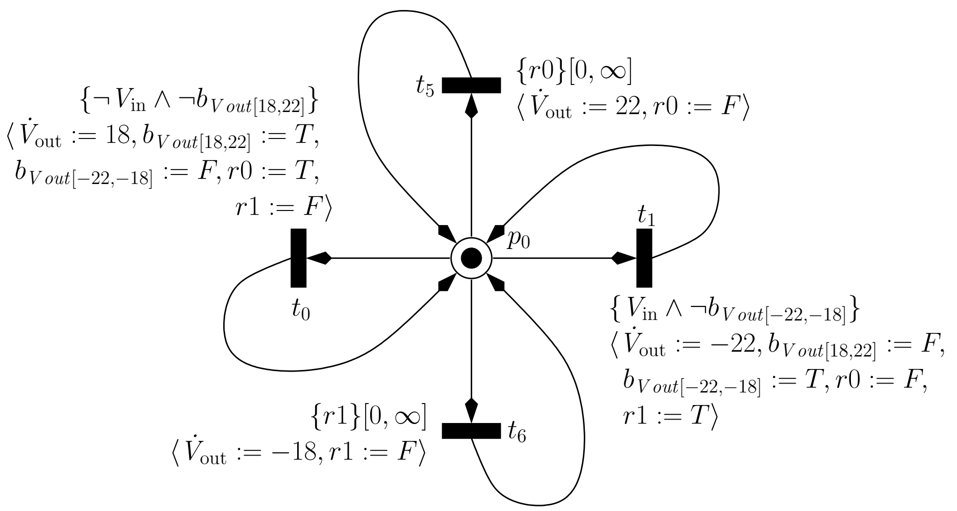
As this expansion is an approximation, it is possible that the approximation may add the potential for a false positive verification result. Namely, the introduction of the approximation may lead the verification engine to find that an LHPN satisfies a property when a failure exists. A specific type of property is required to expose this type of failure. The property requires that the system take a specific trajectory through the state space and check this trajectory in at least two locations. For example, the LHPN in Figure 2.21a has a failure. Initially, t_0 fires setting V_1 to zero and V_2 to zero. Next, t_5 fires setting the rate of V_1 to one, and the rate of V_2 to the range of one to two. If V_2 moves at a rate of one for the next five time units, V_1 is five and V_2 is five. After five time units, the rate of V_2 changes to two and remains stable at two for five time units. At the end of ten time units V_1 is ten and V_2 is fifteen. This state enables the firing of t_1. After firing t_1, the rate of V_2 changes back to one. When another ten time units pass V_1 is twenty and V_2 is twenty-five. This state results in the firing of t_4 which indicates a failure.

The LHPN in Figure 2.21a is transformed into the LHPN of Figure 2.21b using the transforms described in this section. The failure demonstrated on the original LHPN is no longer present in the transformed LHPN. Initially, t_0 fires and sets V_1 to zero and V_2 to zero. Next, t_5 fires setting the rate of V_1 to one, the rate of V_2 to one, and $r0$ to


$\{\neg V_{\text{in}} \wedge \neg b_{Vout[18,22]}\}$
$\langle \dot{V}_{\text{out}} := [18,22], b_{Vout[18,22]} := T,$
$b_{Vout[-22,-18]} := F\rangle$
t_1
p_0
t_0
$\{V_{\text{in}} \wedge \neg b_{Vout[-22,-18]}\}$
$\langle \dot{V}_{\text{out}} := [-22,-18], b_{Vout[18,22]} := F,$
$b_{Vout[-22,-18]} := T\rangle$

$$Q_0 = \{V_{\text{out}} = -1000\}$$
$$R_0 = \{\dot{V}_{\text{out}} = [18,22]\}$$
$$S_0 = \{\neg V_{\text{in}}, \neg \mathit{fail}, b_{Vout[18,22]}, \neg b_{Vout[-22,-18]}\}$$

(a)

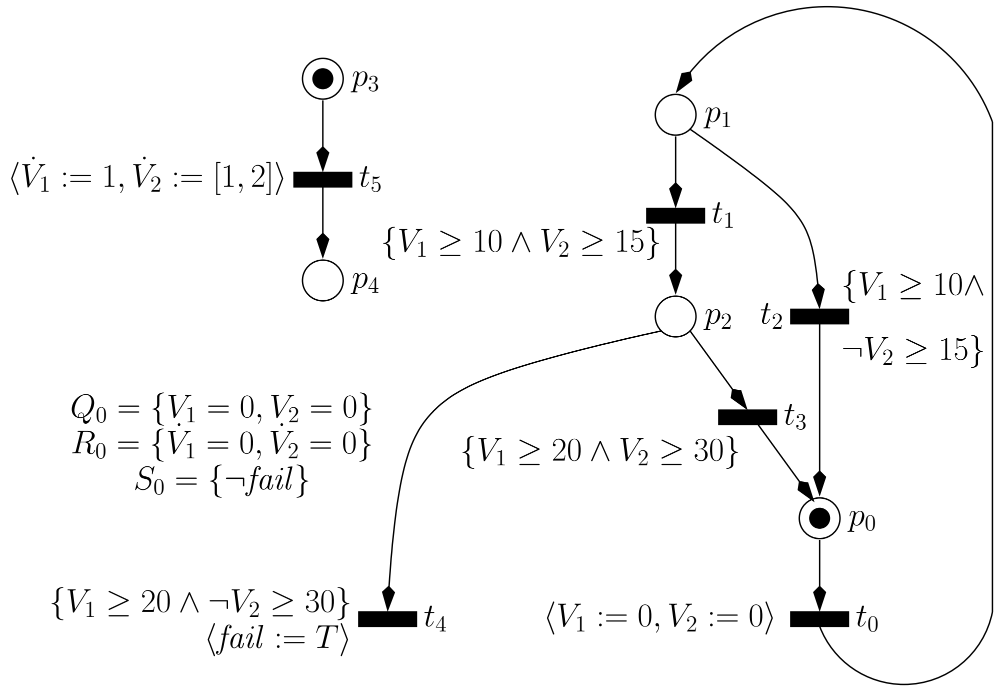

$\{r0\}[0,\infty]$
$\langle \dot{V}_{\text{out}} := 22, r0 := F\rangle$
t_5
$\{\neg V_{\text{in}} \wedge \neg b_{Vout[18,22]}\}$
$\langle \dot{V}_{\text{out}} := 18, b_{Vout[18,22]} := T,$
$b_{Vout[-22,-18]} := F, r0 := T,$
$r1 := F\rangle$
t_1
p_0
t_0
$\{V_{\text{in}} \wedge \neg b_{Vout[-22,-18]}\}$
$\langle \dot{V}_{\text{out}} := -22, b_{Vout[18,22]} := F,$
$b_{Vout[-22,-18]} := T, r0 := F,$
$r1 := T\rangle$
$\{r1\}[0,\infty]$
$\langle \dot{V}_{\text{out}} := -18, r1 := F\rangle$
t_6

$$Q_0 = \{V_{\text{out}} = -1000\}$$
$$R_0 = \{\dot{V}_{\text{out}} = 18\}$$
$$S_0 = \{\neg V_{\text{in}}, \neg \mathit{fail}, b_{Vout[18,22]}, \neg b_{Vout[-22,-18]}, r0, \neg r1\}$$

(b)

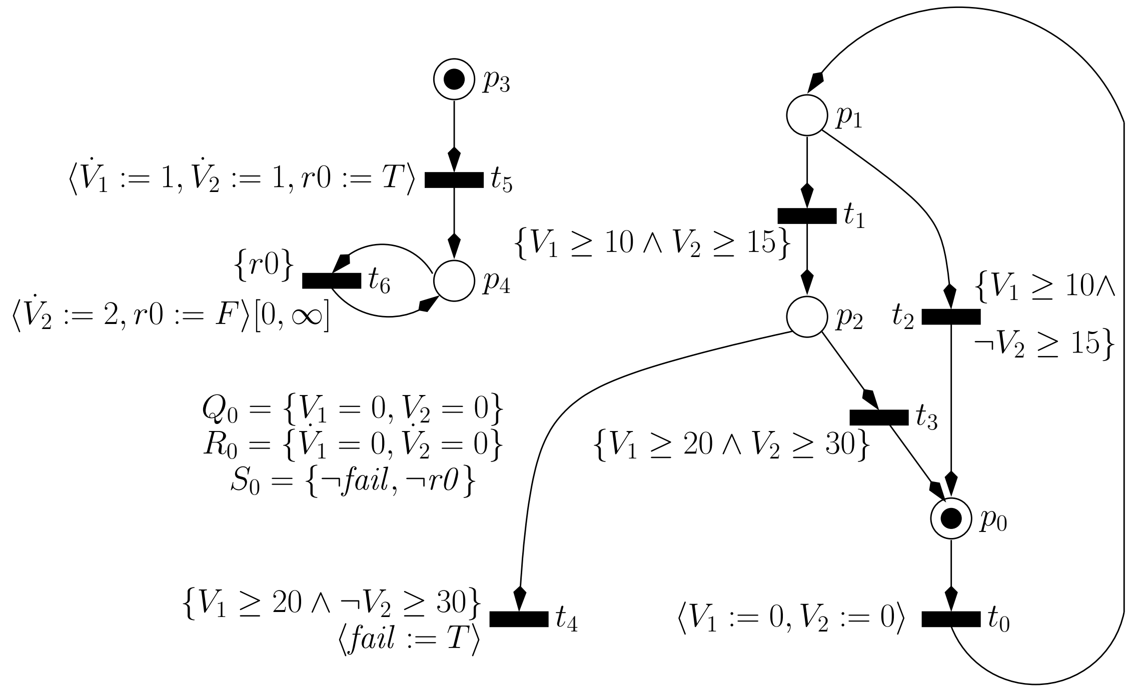
Figure 2.20. LHPNs demonstrating ranges of rates transformed to piecewise constant rates that change nondeterministically. (a) Original LHPN from Figure 2.11a. Parts b and c are not shown because they do not change. (b) Transformed LHPN.

true. After five time units, V_1 is five and V_2 is five. At this time, t_6 could fire setting the rate of V_2 to two and the Boolean signal v_0 to **false**. Note that if t_6 fires later or does not fire, then t_1 does not become enabled and t_2 fires also avoiding the failure. After another five time units, V_1 is ten and V_2 is fifteen. In this state, transition t_1 fires. Time advances by ten time units at which point V_1 is twenty and V_2 is thirty-five. Transition t_4 is not allowed to fire. The constant rate expansion of the LHPN has eliminated the failure. This demonstrates the potential introduction of false positives using the LHPN expansion approximation. However, in many cases this approximation is suitable and does not allow the potential for false positive traces. In our investigations, we have never encountered a real property with such a false positive.



p_3
$\langle \dot{V}_1 := 1, \dot{V}_2 := [1,2] \rangle$
t_5
p_4
p_1
t_1
$\{V_1 \geq 10 \wedge V_2 \geq 15\}$
p_2
t_2
$\{V_1 \geq 10 \wedge$
$\neg V_2 \geq 15\}$
$Q_0 = \{V_1 = 0, V_2 = 0\}$
$R_0 = \{\dot{V}_1 = 0, \dot{V}_2 = 0\}$
$S_0 = \{\neg fail\}$
t_3
$\{V_1 \geq 20 \wedge V_2 \geq 30\}$
p_0
$\{V_1 \geq 20 \wedge \neg V_2 \geq 30\}$
$\langle fail := T \rangle$
t_4
$\langle V_1 := 0, V_2 := 0 \rangle$
t_0

(a)



p_3
$\langle \dot{V}_1 := 1, \dot{V}_2 := 1, r0 := T \rangle$
t_5
$\{r0\}$
t_6
p_4
$\langle \dot{V}_2 := 2, r0 := F \rangle [0, \infty]$
p_1
t_1
$\{V_1 \geq 10 \wedge V_2 \geq 15\}$
p_2
t_2
$\{V_1 \geq 10 \wedge$
$\neg V_2 \geq 15\}$
$Q_0 = \{V_1 = 0, V_2 = 0\}$
$R_0 = \{\dot{V}_1 = 0, \dot{V}_2 = 0\}$
$S_0 = \{\neg fail, \neg r0\}$
t_3
$\{V_1 \geq 20 \wedge V_2 \geq 30\}$
p_0
$\{V_1 \geq 20 \wedge \neg V_2 \geq 30\}$
$\langle fail := T \rangle$
t_4
$\langle V_1 := 0, V_2 := 0 \rangle$
t_0

(b)

Figure 2.21. LHPNs demonstrating the potential for false positive results using the range of rate expansion technique. (a) Original LHPN with a failure. (b) Transformed LHPN without a failure.

CHAPTER 3

ABSTRACT MODEL GENERATION

Chapter 2 introduces a method to automatically obtain formal LHPN models from VHDL-AMS descriptions. While this method is effective because the design has already been abstracted by an expert, it is uncommon for AMS designers to create behavioral HDL models of their designs. These manually abstracted models tend to get out of date as the system evolves during the design process, but automatically generated abstract models can be easily updated. AMS designers typically work at the transistor-level using a SPICE (differential equation) simulator [118, 117, 132]. This chapter begins by revisiting the motivating example of the switched capacitor integrator using SPICE-level simulation. Next, this chapter covers a *simulation aided verification* (SAV) methodology to automatically generate abstract HDL and LHPN models from sets of simulation traces. This chapter concludes with a presentation and discussion of coverage metrics appropriate for the SAV methodology.

3.1 Motivating Example

Chapter 2 presents several behavioral models for the switched capacitor integrator circuit (Figure 2.1). This chapter focuses on the transistor-level model of the circuit. Figure 3.1 shows a plot of SPICE simulation data for V_{out} from the switched capacitor integrator circuit under nominal conditions. Comparing this waveform to the behavioral model's nominal waveform in Figure 2.2 highlights the accuracy difference in the models. The waveform produced by the SPICE simulation has a stair step artifact from the accurate representation of the switched capacitor. While this level of accuracy may be important for design and verification of some properties, it is not necessary to verify the saturation property described previously.

In the behavioral model of the switched capacitor integrator circuit, the designer had direct control over the output slew rate. This control allows straightforward exploration of the effects of different output slew rates of the system. Using the transistor-level

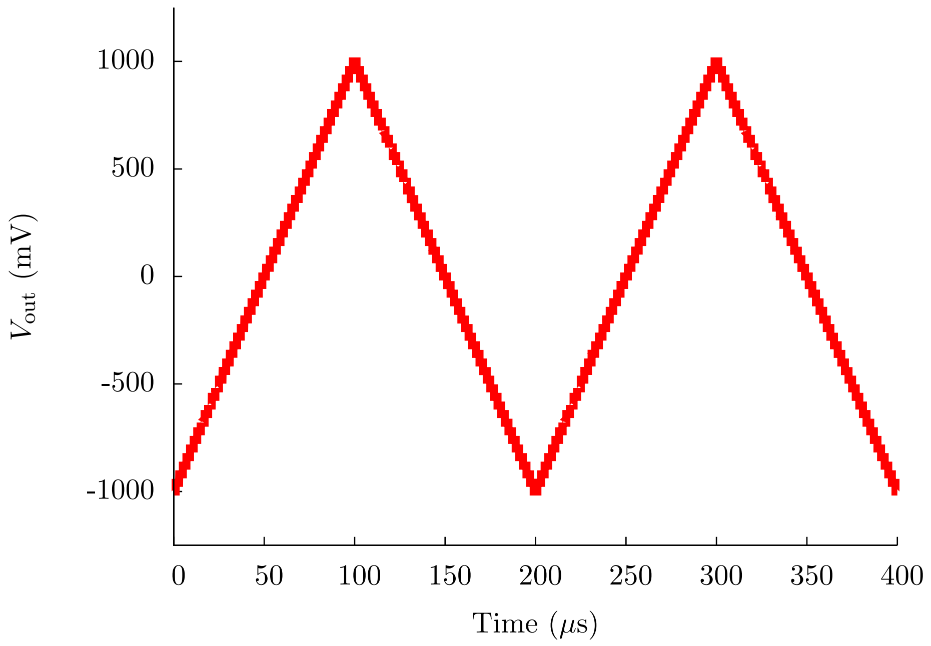


Figure 3.1. A SPICE simulation of the switched capacitor integrator circuit under nominal conditions, a C_2 value of 25 pF.

model, the designer has only indirect control over the output slew rate. This rate is indirectly controlled using the value of capacitance for capacitor C_2. Assuming variance in capacitance values, the designer may run several simulations for different capacitance values to understand the effects of such variation. Figure 3.2 and Figure 3.3 show two such SPICE simulations with capacitance values for C_2 of 23 pF and 27 pF, respectively. The 23 pF capacitor simulation has a faster output slew rate than the nominal condition while the 27 pF simulation has a slower output slew rate. Note that none of these three simulations demonstrate a failure of this circuit. This chapter presents a method that uses two of the three SPICE simulations presented to create an LHPN model that reveals the potential for failure.

3.2 Related Work

As system complexity continues to increase, the need for models at many levels of abstraction is increasing. System-level verification requires system-level simulation which is not feasible within the time constraints imposed by today's design schedules using an

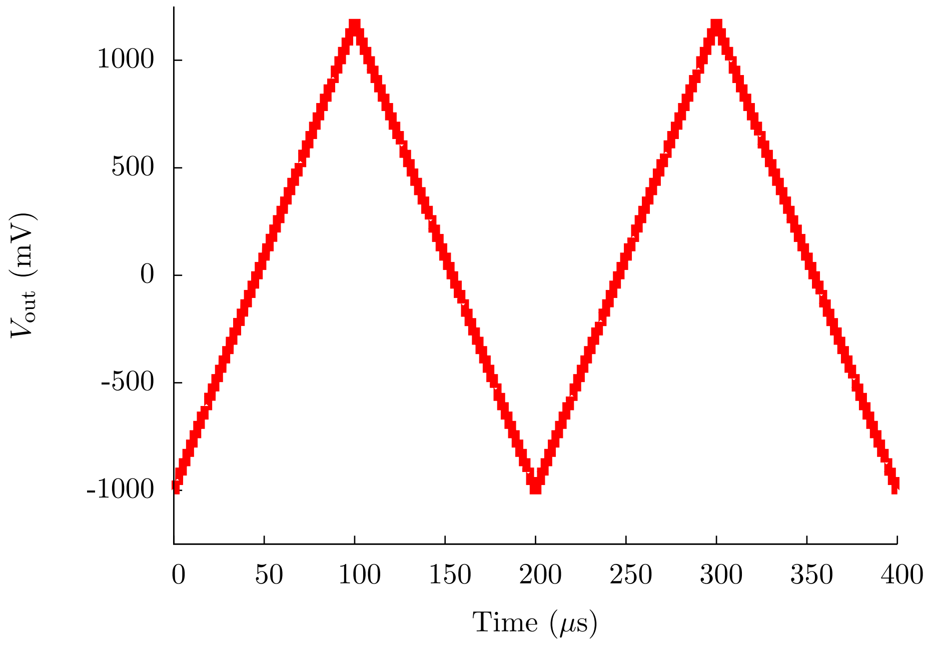


Figure 3.2. A SPICE simulation of the switched capacitor integrator circuit with a C_2 value of 23 pF.

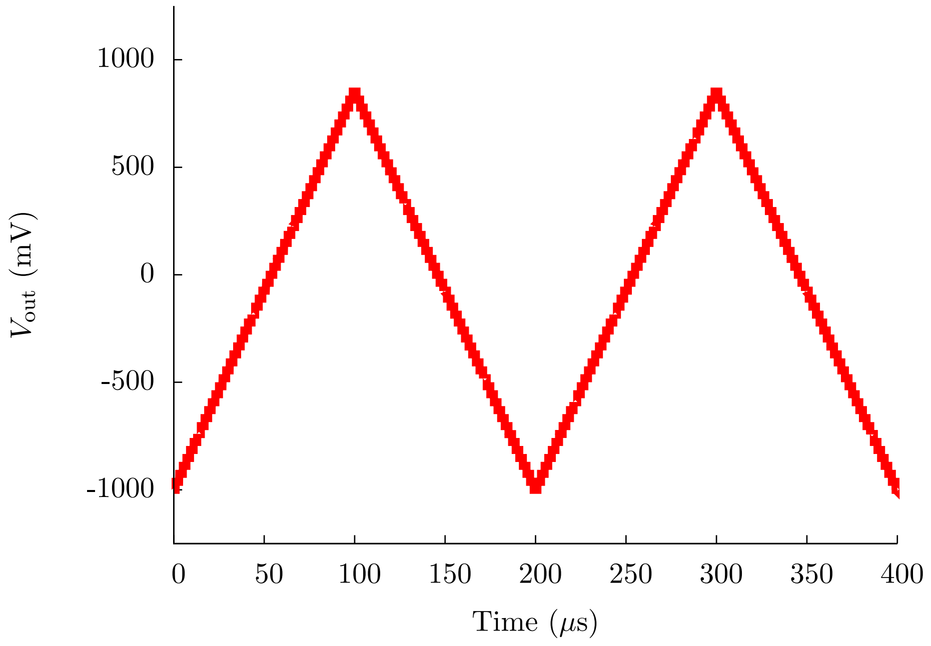


Figure 3.3. A SPICE simulation of the switched capacitor integrator circuit with a C_2 value of 27 pF.

unabstracted model. Currently, the most common approach to creating abstract models is manual abstraction of SPICE or Simulink models. Manual abstraction is an iterative process that is often time intensive and error-prone. Automatic abstraction methods are needed to improve the quality of AMS verification [140].

The success of abstract modeling techniques for AMS circuits depends on the complexity of the system. For purely digital systems, the assumption of digital circuit behavior greatly simplifies the complexity and has led to successful abstraction techniques. For AMS systems, there are four major classes of circuits: *linear time-invariant* (LTI), *linear time-varying* (LTV), *nonlinear*, and *oscillatory*. Automatic abstract model generation techniques have been developed for each class of circuit with varying levels of success.

LTI systems are the simplest class of circuits, and automatic abstraction techniques have proven to be most successful for LTI systems. An LTI system is composed of only resistors, inductors, and capacitors. The focus of the modeling work for LTI systems has been on interconnect as there is a great need to compactly model distributed interconnect networks such as clock distribution trees. Therefore, the goal of LTI abstract modeling is to reduce the number of variables required to represent an LTI system while still maintaining the important behaviors. In [130], the authors present *asymptotic waveform evaluation* (AWE) as a method to automatically abstract LTI systems. AWE produces abstract models whose behavior is nearly identical to the full system in time that is roughly linear to the size of the unabstracted system. The main drawback to AWE is that the abstract models produced by AWE become increasingly inaccurate as their size increases above about ten variables. To address this numerical inaccuracy, *Krylov-subspace* methods were introduced [68, 13]. While Krylov-subspace based abstraction methods are very effective, they do not guarantee the important properties of passivity and stability. A passive system cannot generate energy internally, and for LTI systems, passivity guarantees stability. Several Krylov-subspace based methods have been developed that guarantee passivity of the abstract model [123, 21, 59]. While the Krylov-subspace methods are computationally efficient, they are not optimal. In other words, they do not minimize the model error for a model of a given size. To address the optimality concerns, abstraction methods for LTI systems based on *truncated balance realizations* (TBR) have been proposed [70]. TBR methods are not as computationally efficient as Krylov-subspace methods, so recent work has focused on combining Krylov-subspace techniques with TBR [90] as well as extending these more efficient methods to creating parametrized models for use in design space exploration [45, 147].

While the work on LTI systems is promising, AMS circuits are composed largely of nonlinear components. There is a class of nonlinear circuits which includes switched-capacitor and sampling circuits that can be modeled as LTV systems. To create abstract models of LTV systems, the LTV systems are reformulated as an LTI system with additional artificial inputs to represent the time variation [139]. The different LTI abstraction methods have been used successfully within this framework [138, 139, 126].

Abstraction of nonlinear systems is the key to abstract modeling of AMS designs. Even for weakly nonlinear systems where AMS designers intend that their circuits operate in a linear way (e.g., linear amplifiers), it is critical to model the nonlinearities of the devices as the designer is often trying to minimize these effects. There are also many systems that exhibit and exploit highly nonlinear behavior such as ADCs, oscillators, and PLLs. No technique currently exists that is capable of producing accurate abstract models for general nonlinear systems, but methods do exist that take approximations of the original system and create abstractions of these approximations. There are a number of methods to abstract weakly nonlinear systems by leveraging LTI abstraction techniques and then accounting for the distortions introduced by the weak nonlinearities of the circuit [127, 128, 99].

The primary approach to creating abstract models of strongly nonlinear systems is to approximate the nonlinear system using a composition of *piecewise linear* (PWL) segments. This is advantageous because if it is assumed that each segment is linear then linear abstraction methods can be used within each segment. Given this method, the challenge is creating and composing these PWL segments. The *trajectory PWL* (TPWL) method proposed in [134] uses existing LTI abstraction methods to create abstracted linear models for each segment and stitches them together using a scalar weight function to create the final system model. The weakness of TPWLs is that they do not capture higher order derivative information well. This makes them a poor choice to model many nonlinear systems as higher order dynamics are common and often critical to correct behavior. To address this problem, *piecewise polynomial* (PWP) techniques are introduced that use weakly nonlinear abstraction techniques to create the abstractions for each segment [52]. One major difficulty encountered by the TPWL and PWP methods is the large number of training simulations required to create the model. Scalable trajectory models are introduced [153, 154] that use data mining techniques to reduce the number of simulations required for the training input. Although scalable trajectory models decrease the training complexity, it still remains unacceptably high, and input selection is difficult for complex dynamics.

Oscillatory behavior is a particularly difficult simulation and modeling challenge. Specialized simulators are often used to accurately simulate oscillators. The same has held true for abstraction techniques as specialized abstract models in the phase domain have been most successful for creating abstract models of oscillatory behavior [98, 159].

Coverage metrics quantify how well a set of simulations explores the state space of a system. Finding a set of simulations that thoroughly explores the state space is useful for characterizing abstract model quality. Understanding the design coverage of the set of simulations used to generate the model is also important to the SAV methodology presented in this chapter. When the verifier returns a verification result for a given model, it is important to understand what that result actually means. Namely, the completeness of the result depends on the completeness of the simulations used to build the abstract model. Ideally, a model generator would compute a set of coverage metrics that describe the model quality. Coverage metrics for digital systems have received significant attention [150] while little work has been done for analog coverage metrics [143, 120, 119, 89].

Sha et al. propose code coverage metrics for Verilog-A in [143]. The authors apply standard code coverage metrics to Verilog-A code in the form of *statement coverage*, *decision coverage*, and *condition coverage*. Statement coverage measures whether each line of code is executed. Decision coverage measures whether each conditional statement is executed for all truth values. Condition coverage measures whether all possible combinations of complex conditional statements are executed. A new coverage metric for analog circuits is proposed, *frequency coverage*. The frequency coverage metric measures the proportion of the frequency space that is covered with respect to the poles and zeroes of the circuit. There is some difficulty in obtaining the poles and zeroes. Solutions are provided for simple cases, but the complexity of these solutions grows quickly for more complex circuits.

The work by Nahhal and Dang in [120, 119] presents coverage measures for hybrid systems to guide test generation based on the *rapidly-exploring random tree* algorithm. This coverage measure is state-based and defined using the *star discrepancy* from statistics. The methodology is tested using two examples from hybrid systems theory which shows the potential for the methodology to scale to higher dimensional systems.

The notion of a *robust test* is introduced by Julius et al. in [89]. Particularly for AMS circuits, it is possible that one test may be representative of many other tests. A robust test is one where a slight but quantifiable perturbation in the model or initial conditions is guaranteed to result in a trace with the same qualitative properties as the nonperturbed trace. The qualitative properties can be safety properties. This paper presents several strategies to discover a robust neighborhood of initial states that can provide good coverage for the entire design space. This methodology is tested using hybrid system benchmarks where it does not achieve fifty percent coverage after hundreds of simulations.

3.3 Abstract Model Generation

Our modeling work differs from the previous work in several ways. The most pronounced differences are the accuracy of the abstract model and the use of nondeterminism. Previous methods attempt to abstract the model while maintaining transistor-level accuracy. The abstract models produced by `LEMA`'s model generator do not attempt to maintain this level of accuracy but do model ranges of parameters and conditions using nondeterminism. They are intended to be used in system-level simulations to verify properties such as connectivity between the digital and analog circuits or for use in formal verification. As a result, these models are less general, but the model generation process and simulations using these models are much more efficient.

During the course of traditional analog circuit verification, designers run many different simulations to verify that the circuit meets its specification. The goal of this work is to automatically generate abstract models from this simulation data. The generated circuit models are conservative and model all the provided simulation traces plus additional behavior. Due to the use of simulations already produced by the designer, no additional simulation time is required. However, the quality of the model is directly related to the simulations used to create it. If the designer has inadequately simulated the design, the model may not exhibit the full behavior of the system. In this case, there is a potential that the actual circuit may have a failing behavior that is not included in the generated model. To help address this issue, Section 3.4 discusses the use of coverage metrics.

Two simulations of the switched capacitor integrator are used to concretely illustrate the model generation process. In particular, the switched capacitor integrator is simulated with capacitance values of 23 pF (Figure 3.2) and 27 pF (Figure 3.3) for capacitor C_2.

The simulation data are recorded for the nodes representing the input voltage, V_{in}, and output voltage, V_{out}, during a 400 μs transient simulation for each capacitance value. Part of the data from these simulations are shown in Tables 3.1 and 3.2.

Algorithm 3.1 describes the process of taking simulation data and generating an abstract model. The inputs to the algorithm are Var, Σ, θ, $prop_{\mathrm{HSL}}$, ws, ϵ, $ratio$, $\tau_{\min}$, $nonC$, sig, and sep. Each variable $\nu \in Var$ is a design variable in the system being modeled. Σ is the set of time series simulation traces. Each trace $\sigma \in \Sigma$ is an n-tuple $\langle \tau, \nu_0, \ldots, \nu_n \rangle$ where $\tau \in \mathbb{R}$ is the timestamp for the data points $(\nu_0, \ldots, \nu_n) \in \mathbb{R}^n$ where $\nu_i \in Var$ and n is $|Var|$. For example, in Table 3.1 the first column is τ; the second column is ν_0; and the third column is ν_1. To access the timestamp for data point i the notation $\sigma_i(\tau)$ is used. Similarly, to access the data value i for variable ν the notation $\sigma_i(\nu)$ is used. In Table 3.1, $\sigma_1(\tau)$ is 0.51 μs and $\sigma_1(V_{\mathrm{out}})$ is -999 mV. θ is the set of thresholds on the signal levels of the design variables in Var. Thresholds are used during the model generation process to divide the state space of the design into regions of operation and do not need to be equidistant. Increasing the number of thresholds increases both the complexity and accuracy of the model. The thresholds, θ, for each variable ν are $\langle \theta_0(\nu), \ldots, \theta_m(\nu) \rangle$ where $\theta_0(\nu)$ is $-\infty$ and $\theta_m(\nu)$ is ∞. If the value of the variable is limited then $\theta_0(\nu)$ is set to the lower limit and $\theta_m(\nu)$ is set to the upper limit. The thresholds are used to group the simulation data into regions ξ, where $\xi_i(\nu) = [\theta_i(\nu), \theta_{i+1}(\nu))$. The lowest region for a variable is $\xi_0(\nu)$ and the highest region is $\xi_{m-1}(\nu)$. $prop_{\mathrm{HSL}}$ is a safety property specified using a restricted HSL formula. The remaining parameters to `genModel`, ws, ϵ, $ratio$, $\tau_{\min}$, $nonC$, sig, and sep, are optional parameters that can be specified to configure the model generation process. The default values for these optional parameters are shown in Table 3.3. ws is the window size used in rate calculation and has a default value of 200. ϵ, $ratio$, and $\tau_{\min}$ are used in the detection of digital-like signals and have default values of 0.1, 0.8, and 5e-6, respectively. $nonC$ is a function that maps a signal to a set of non-causal signals (i.e., $nonC\colon Var \rightarrow 2^{Var}$). sig and sep are used in the normalization process and have default values of 2 and 1, respectively.

In Algorithm 3.1, each data point for each variable for each simulation trace $\sigma_i(\nu)$ is given a region assignment reg_i based on the thresholds (lines 2-3). The variable's rate assignment at data point i is accessed using the notation $reg_i(\nu)$ and is an integer value between 0 and $|\xi(\nu)|$. In Table 3.1, the fourth column represents the region assignment,

Table 3.1. A partial simulation result with C_2 = 23 pF for the switched capacitor integrator circuit.

Time (μs)	V_{in} (mV)	V_{out} (mV)	Region	$\Delta V_{\text{in}}/\Delta t$ (mV/μs)	$\Delta V_{\text{out}}/\Delta t$ (mV/μs)
0.00	-1000	-1000	00	-40.07	21.85
0.51	-1000	-999	00	0.0	21.74
⋮	⋮	⋮	⋮	⋮	⋮
28.52	-1000	-391	00	0.0	23.74
32.00	-1000	-304	00	-	-
35.01	-1000	-217	00	-	-
38.51	-1000	-174	00	-	-
41.54	-1000	-87	00	-	-
45.00	-1000	5	01	0.0	21.72
48.01	-1000	43	01	0.0	21.18
⋮	⋮	⋮	⋮	⋮	⋮
100.48	-1000	1173	01	-	-
100.50	-1000	1174	01	-	-
100.54	-840	1174	01	-	-
100.62	-520	1176	01	-	-
100.78	120	1176	11	275.00	-21.08
101.00	1000	1174	11	0.0	-21.74
101.03	1.0	1173	11	0.0	-21.74
⋮	⋮	⋮	⋮	⋮	⋮
154.48	1000	11	11	-	-
154.98	1000	0.3	11	-	-
155.48	1000	-11	10	0.0	-21.74
155.98	1000	-21	10	0.0	-21.74
⋮	⋮	⋮	⋮	⋮	⋮
199.98	1000	-978	10	-	-
200.00	1000	-978	10	-	-
200.04	840	-979	10	-	-
200.12	520	-980	10	-	-
200.28	-120	-981	00	-275.00	21.08
200.50	-1000	-978	00	0.0	21.74
200.53	-1000	-976	00	0.0	21.74
⋮	⋮	⋮	⋮	⋮	⋮
400.00	1000	-957	10	-	-

Table 3.2. A partial simulation result with $C_2 = 27$ pF for the switched capacitor integrator circuit.

Time (μs)	V_{in} (mV)	V_{out} (mV)	Region	$\Delta V_{\text{in}}/\Delta t$ (mV/μs)	$\Delta V_{\text{out}}/\Delta t$ (mV/μs)
0.00	-1000	-1000	00	-227.85	18.14
0.50	-1000	-999	00	0.0	18.52
⋮	⋮	⋮	⋮	⋮	⋮
37.54	-1000	-296	00	0.0	19.17
41.00	-1000	-222	00	-	-
43.82	-1000	-185	00	-	-
46.72	-1000	-148	00	-	-
49.54	-1000	-74	00	-	-
53.00	-1000	17	01	0.0	17.49
55.82	-1000	37	01	0.0	17.40
⋮	⋮	⋮	⋮	⋮	⋮
100.48	-1000	851	01	-	-
100.50	-1000	852	01	-	-
100.54	-840	852	01	-	-
100.62	-520	853	01	-	-
100.78	120	854	11	275.00	-17.96
101.00	1000	852	11	0.0	-18.52
101.03	1000	850	11	0.0	-18.52
⋮	⋮	⋮	⋮	⋮	⋮
146.48	1000	10	11	-	-
146.98	1000	0.3	11	-	-
147.48	1000	-9	10	0.0	-18.52
147.98	1000	-18	10	0.0	-18.52
⋮	⋮	⋮	⋮	⋮	⋮
199.98	1000	-981	10	-	-
200.00	1000	-981	10	-	-
200.04	840	-982	10	-	-
200.12	520	-983	10	-	-
200.28	-120	-984	00	-275.00	17.96
200.50	-1000	-981	00	0.0	18.52
200.53	-1000	-980	00	0.0	18.52
⋮	⋮	⋮	⋮	⋮	⋮
400.00	1000	-963	10	-	-

Table 3.3. Default values for the model generator's optional parameters.

Parameter	Default Value
ws	200
ϵ	0.1
$ratio$	0.8
$\tau_{\min}$	5e-6
$nonC$	$\emptyset$
sig	2
sep	1

Algorithm 3.1: `genModel`($Var, \Sigma, \theta, prop_{\text{HSL}}, ws, \epsilon, ratio, \tau_{\min}, nonC, sig, sep$)

```
1  $N :=$ null;
2  forall $\sigma \in \Sigma$ do
3      $reg :=$ assignRegions($\sigma$, $Var, \theta$);
4      $rate :=$ calculateRates($\sigma$, $Var$, $ws$, $reg$);
5      $(dmv, start, end) :=$ detectDMV($\sigma$, $Var, \epsilon, ratio, \tau_{\min}$);
6      $N :=$ updateLHPN($N$, $Var, \sigma, reg, rate, dmv, start, end, nonC, \theta$);
7  $N :=$ addPseudoRegions($N, \Sigma, \theta$);
8  writeNormalizedLHPN($N, prop_{\text{HSL}}, sig, sep$);
9  writeVHDLAMS($N, prop_{\text{HSL}}$);
10 writeVerilogAMS($N$);
```

so $reg_{|\sigma_i|}$ is 10; $reg_{|\sigma_i|}(V_{\text{in}})$ is 1. Next, ranges of rates are calculated for each continuous variable within each region (line 4). The algorithm assumes nothing about the dependence or independence of the rates. Each rate is calculated individually for each region for each simulation trace. The variable's region assignment at data point i is accessed using the notation $rate_i(\nu)$. In Table 3.1, the fifth column represents the rate assignment for V_{in}, so $rate_0(V_{\text{in}})$ is -40.07 mV/μs. It is expected that the rates change during different phases of operation. For this reason, it is important that thresholds are selected to separate the different phases of operation into distinct regions. At this point, continuous variables which are mostly stable but occasionally change are identified as variables that can be approximated by discrete transitions, *discrete multi-valued* (DMV) variables (line 5). The variable dmv is the set of variables detected as DMV variables. All of this information is collected for the current simulation trace and combined with the information from other simulation traces in the LHPN representing the system model (line 6). Many continuous quantities are bounded by physical limits. For instance, the voltage in a circuit is often bounded by the voltage rails. *Pseudo-regions* are added to restrict the continuous values to such predefined limits (line 7). Finally, the abstract models are generated (lines 8-10).

Selecting quality thresholds is a critical step in the model generation process. As the number of variables and the complexity of the system increases, this becomes a tedious and error prone task. A tool to automatically select optimal thresholds using optimization algorithms aids in the selection of thresholds. The purpose of automatic threshold generation is to suggest a set of thresholds that can then be verified and adjusted, if needed, by the designer. The tool currently supports two cost functions and a single optimization algorithm. The two supported cost functions produce models whose data points are evenly distributed across the regions and regions whose rates span a minimal distance. A greedy algorithm is used as the optimization function. The tool is designed to make adding cost functions and optimization algorithms easy. Although, still in its infancy, this tool has been used to guide threshold selection for several examples.

3.3.1 Assigning the Regions

The first step of Algorithm 3.1 is to assign the data to regions based upon the user-provided thresholds (line 3). In this example, the thresholds chosen for both V_{in} and V_{out} are 0 V. Algorithm 3.2 analyzes each variable in the system at each time point in a simulation trace to assign the appropriate region encoding (lines 1-2). The region value is assigned based upon the location of the data point to the threshold values for the given variable (lines 3-5). The result of Algorithm 3.2 for the 23 pF simulation is shown graphically in Figure 3.4. Initially, both V_{in} and V_{out} are below the threshold of 0 V resulting in a region assignment of $\langle \texttt{00} \rangle$, where the order of the variables is $\langle V_{\text{in}} V_{\text{out}} \rangle$. V_{out} rises, crosses the threshold, and moves the system into region $\langle \texttt{01} \rangle$. V_{in} then rises and moves the system into region $\langle \texttt{11} \rangle$. Region assignment proceeds in this manner for the remainder of the simulation trace. In the data shown in Tables 3.1 and 3.2, each digit in the fourth column represents the region for that time point. For instance, at time 100.62 μs in Table 3.1, the region assigned is $\langle \texttt{01} \rangle$ indicating that V_{in} is below 0 V and V_{out} is above 0 V. When V_{in} moves above 0 V at time 100.78 μs, the region assignment changes to $\langle \texttt{11} \rangle$.

3.3.2 Calculating the Rates

After regions have been assigned to each data point, the rates are calculated for each region using Algorithm 3.3, `calculateRates`. A rate is calculated for each eligible data

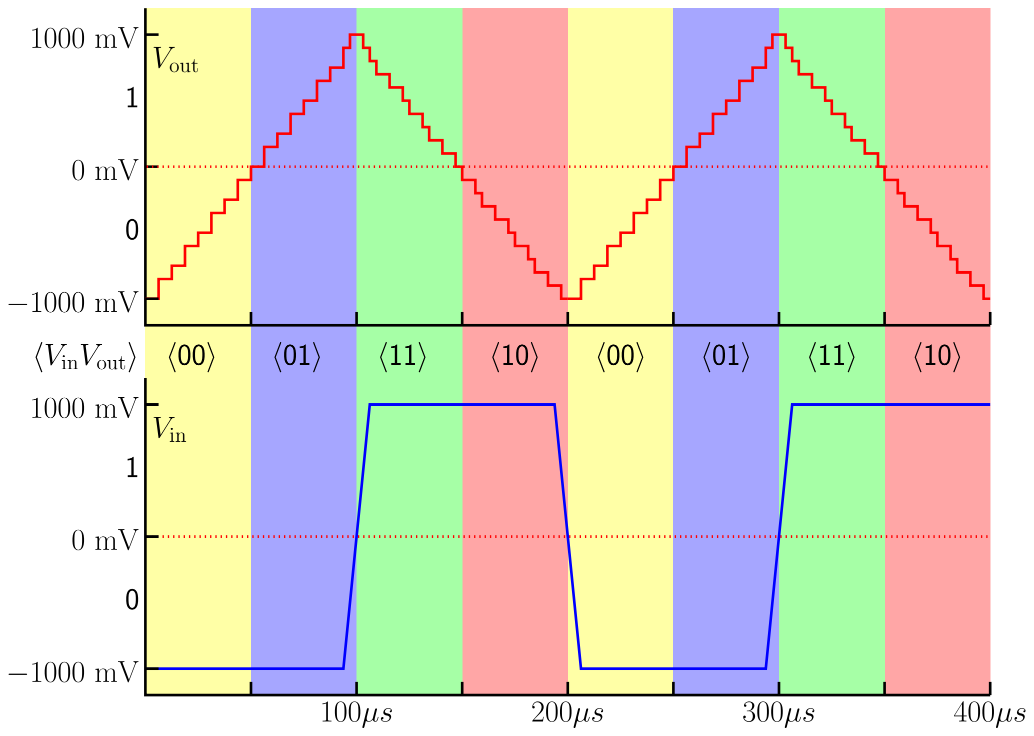


Figure 3.4. The results of the region assignment algorithm for the switched capacitor integrator example.

Algorithm 3.2: `assignRegions`(σ, *Var*, θ)

```
1 forall $\nu \in Var$ do
2     forall $i \in [0, |\sigma|]$ do
3         forall $j \in [0, |\xi(\nu)|]$ do
4             if $\sigma_i(\nu) \in \xi_j(\nu)$ then
5                 $reg_i(\nu) := j$;
6 return $reg$;
```

point in the trace (line 1). Not all points are eligible for rate calculations due to a low pass filtering technique used to smooth edge effects caused by region boundaries and transitory pulses. The low pass filtering uses a sliding window approach. The size of the window, *ws*, is an optional configuration parameter. The sliding window approach works by calculating the rate of change for a variable between the current point and a point *ws* points further in time if all points between are in the same region (lines 2-4). For instance, in Table 3.1, the rate of V_{out} is calculated for 28.52 μs but not for 32.00 μs. A rate is not calculated for 32.00 μs because the value of V_{out} that is *ws* points later is in a different region. For example, using a value of four for ws, the rate of change for V_{out} at time 28.52 μs in Table 3.1 is calculated by looking at its value at this time point and the value four points later, 41.54 μs. This value is determined to be 23.74 mV/μs. When the algorithm moves to calculate the rate for the next point, 32.00 μs, it finds that the data point four points later is in a different region and does not calculate a rate for the 32.00 μs point. A similar condition exists in Table 3.2 where a rate is calculated for 37.54 μs but not for 41.00 μs.

Algorithm 3.3: `calculateRates`(σ, Var, ws, reg)

```
1 forall $i \in [0, |\sigma| - ws]$ do
2     if $\forall j \in [0, ws].reg_i = reg_j$ then
3         forall $\nu \in Var$ do
4             $rate_i(\nu) := (\sigma_{i+ws}(\nu) - \sigma_i(\nu))/(\sigma_{i+ws}(\tau) - \sigma_i(\tau))$;
5 return $rate$;
```

3.3.3 Extracting Discrete Multivalued Variables

In AMS designs, it is expected that digital signals are present. To take advantage of the digital abstraction and reduce analysis complexity, digital-like signals are detected and modeled discretely. Instead of allowing them to change with a specific rate, a constant value can be directly assigned to the variable at a specified time after entering a region. A DMV variable is detected when it remains constant for a specified ratio of time with respect to the total time for the simulation. Remaining constant is defined as staying within an ϵ bound for a minimum time, τ_{min}. This condition is illustrated in Figure 3.5. Initially, the signal is constant within the lower ϵ bound. Then, it leaves the lower ϵ bound represented by the first light gray region and is considered unstable. It enters the upper ϵ bound for a brief period represented by the dark gray region. The period does not exceed τ_{min}, so the signal is not considered stable at that point. The signal settles at the upper ϵ bound before falling again. Algorithm 3.4 describes the DMV detection algorithm, `detectDMV`. The algorithm tests each variable in the trace separately (line 2). The analysis begins with the first point and checks to see if the second point is equivalent within the specified ϵ bound (lines 3-7). If it is within the ϵ bound, the next point is tested. This occurs until a point is found that is not equivalent. When this occurs, the time elapsed between the initial point and the current position is tested (line 8). If this

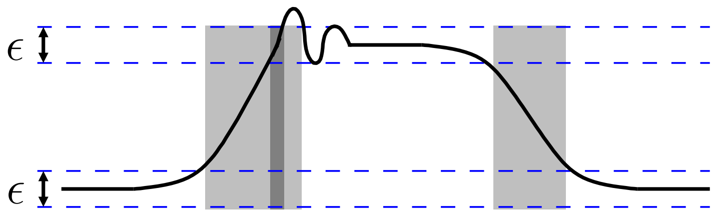


Figure 3.5. An illustration of a DMV variable.

time, $\sigma_j(\tau)-\sigma_i(\tau)$, is greater than τ_{min}, the value is added to the running total of constant time, t_{const} (lines 8-9). The start and end points for the constant run of the variable ν are also recorded (lines 10-11). When all points have been analyzed, the ratio of constant time, τ_{const}, to total time for the trace, $\sigma_{|\sigma|}(\tau)$, is calculated. If this ratio exceeds the specified *ratio*, the variable is added to the set of DMV variables (lines 12-13). In the switched capacitor integrator example, the square wave input voltage, V_{in}, is an example of a DMV variable. This can be inferred from Table 3.1 as V_{in} is largely constant.

Algorithm 3.4: `detectDMV`(σ, Var, ϵ, $ratio$, τ_{min})

```
1  $start,\ end = \emptyset$;
2  forall $\nu \in Var$ do
3      $i, j, \tau_{\text{const}} := 0$;
4      while $i < |\sigma|$ do
5          $i := j$;
6          while $(|\sigma_i(\nu) - \sigma_{j+1}(\nu)| \leq \frac{\epsilon}{2} \wedge j < |\sigma|)$ do
7              $j := j + 1$;
8          if $(\sigma_j(\tau) - \sigma_i(\tau)) \geq \tau_{\text{min}}$ then
9              $\tau_{\text{const}} := \tau_{\text{const}} + (\sigma_j(\tau) - \sigma_i(\tau))$;
10             $start(\nu) = start(\nu) \cup i$;
11             $end(\nu) = end(\nu) \cup j$;
12     if $(\tau_{\text{const}}/\sigma_{|\sigma|}(\tau)) \geq ratio$ then
13         $dmv := dmv \cup \nu$;
14 return $(dmv,$;
```

3.3.4 LHPN Generation

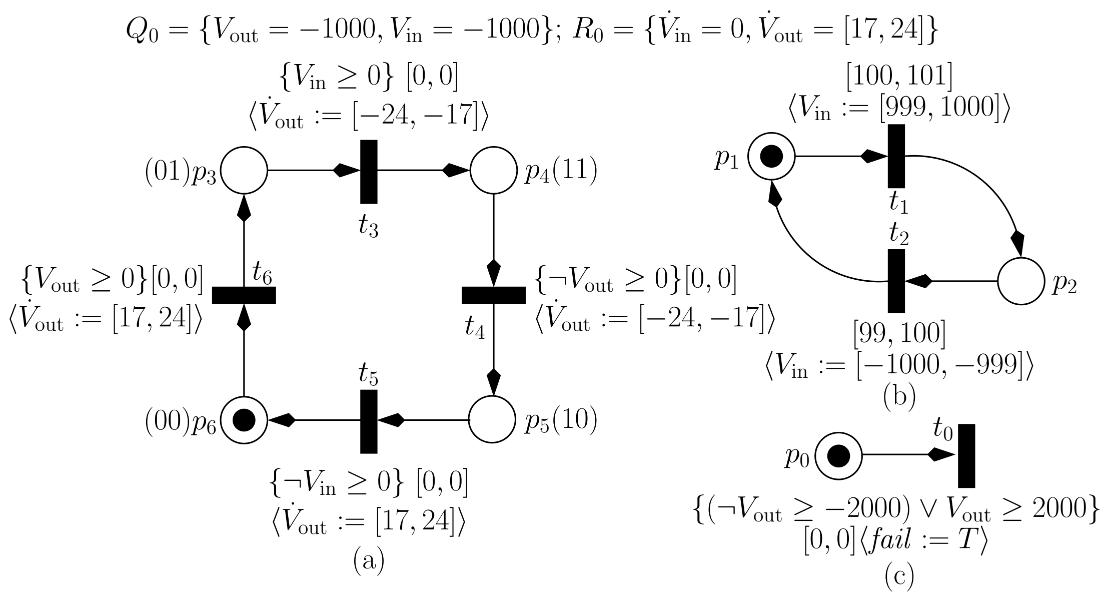
After the needed information has been calculated for a trace, `updateLHPN` shown in Algorithm 3.5 updates the LHPN, N, with the new information. The `updateRegionGraphs` function shown in Algorithm 3.6 examines each region in the simulation trace adding new regions and updated the rates of currently existing regions. The `updateDMVGraphs` function shown in Algorithm 3.7 adds information to the graphs for DMV variables (line 2). The initial value, rate, and region for each variable in each simulation trace are also recorded in the LHPN, N (lines 3-6).

Algorithm 3.5: `updateLHPN`$(N, \mathit{Var}, \sigma, \mathit{reg}, \mathit{rate}, \mathit{dmv}, \mathit{start}, \mathit{end}, \mathit{nonC}, \theta)$

```
1 updateRegionGraphs($N, \sigma, \mathit{reg}, \mathit{rate}, \theta$);
2 updateDMVGraphs($N, \sigma, \mathit{reg}, \mathit{dmv}, \mathit{start}, \mathit{end}, \mathit{nonC}$);
3 forall $\nu \in \mathit{Var}$ do
4     $Q_0(N)(\nu) := Q_0(N)(\nu) \cup \sigma_0(\nu)$;
5     $R_0(N)(\nu) := R_0(N)(\nu) \cup \mathit{rate}_0(\nu)$;
6     $M_0(N)(\nu) := M_0(N)(\nu) \cup p(\mathit{reg}_0(\nu))$;
7 return $N$;
```

The `updateRegionGraphs` algorithm shown in Algorithm 3.6 updates the LHPN with region information from each simulation trace. The notation $p(\mathit{reg}_i)$ returns the place representing reg_i. Similarly, the notation $t(\mathit{reg}_i, \mathit{reg}_j)$ returns the transition between the place representing reg_i and reg_j. The `newT` function takes the LHPN, a source place, a sink place, a delay bound, the thresholds, and an enabling condition. The function updates the flow relation, delay bounds, enabling condition, and rate assignments for the LHPN before returning the newly created transition. The **let** notation is used for convenience to demonstrate the contents of a tuple (line 1). The null value is used to initialize nonset based data types to an initial value (line 2). The algorithm then begins updating the regions for each data point in the given trace (line 3). A node in the graph is added for the new region if the region has not been found previously (lines 4-5). For example, Figure 3.6 contains four places $p_3 - p_6$ representing the four different regions discovered, 00, 01, 11, and 10. While in this example a place is created for every possible region assignment, in larger examples, many regions may never be encountered during simulation. Places are not generated for these unreached regions. If not previously seen, transitions between the current region and the previous region are created using `newT` and added to the set of transitions (lines 6-7). It is theoretically possible that this process could result in a fully connected graph, but in practice this is highly unlikely. The delay for the transition is set to [0,0] to make it fire immediately as the state of the system moves from one region to the next. The `diffR` function determines the differences between the two regions for use in generating the enabling condition. Each transition is given an enabling condition representing the threshold that is being crossed in the move from the first region to the second, $\{V_{\text{in}} \geq 0\}$ on t_3. In Figure 3.6, there are transitions from 00 to 01, from 01 to 11, from 11 to 10, and from 10 to 00. This cycle can be seen in Figure 3.4. The rate of change for the continuous variables in each region is recorded if it is outside the current range (lines 8-13). These rates are stored on the transitions between regions as shown in Figure 3.6. The range of rates for each region found from these two simulation runs for V_{out} from the switched capacitor integrator are shown in Table 3.4.

The `updateDMVGraphs` algorithm shown in Algorithm 3.7 updates the LHPN with DMV variable information from each simulation trace. The function examines the simu-



$Q_0 = \{V_{\text{out}} = -1000, V_{\text{in}} = -1000\}$; $R_0 = \{\dot{V}_{\text{in}} = 0, \dot{V}_{\text{out}} = [17, 24]\}$
$\{V_{\text{in}} \geq 0\}$ $[0, 0]$
$\langle \dot{V}_{\text{out}} := [-24, -17] \rangle$
$(01)p_3$
t_3
$p_4(11)$
$\{V_{\text{out}} \geq 0\}[0, 0]$
$\langle \dot{V}_{\text{out}} := [17, 24] \rangle$
t_6
$\{\neg V_{\text{out}} \geq 0\}[0, 0]$
$\langle \dot{V}_{\text{out}} := [-24, -17] \rangle$
t_4
t_5
$(00)p_6$
$p_5(10)$
$\{\neg V_{\text{in}} \geq 0\}$ $[0, 0]$
$\langle \dot{V}_{\text{out}} := [17, 24] \rangle$
(a)
$[100, 101]$
$\langle V_{\text{in}} := [999, 1000] \rangle$
p_1
t_1
t_2
p_2
$[99, 100]$
$\langle V_{\text{in}} := [-1000, -999] \rangle$
(b)
t_0
p_0
$\{(\neg V_{\text{out}} \geq -2000) \vee V_{\text{out}} \geq 2000\}$
$[0, 0]\langle fail := T \rangle$
(c)

Figure 3.6. The LHPN for the switched capacitor integrator after `updateLHPN`.

Table 3.4. Rates for V_{out} from the switched capacitor integrator circuit used by the LHPN in Figure 3.6.

Region	Place	Range of rates	Comment
00	p_6	[17,24]	$V_{\text{in}} < 0$ V; $V_{\text{out}} < 0$ V
01	p_3	[17,24]	$V_{\text{in}} < 0$ V; $V_{\text{out}} \geq 0$ V
11	p_4	[-24,-17]	$V_{\text{in}} \geq 0$ V; $V_{\text{out}} \geq 0$ V
10	p_5	[-24,-17]	$V_{\text{in}} \geq 0$ V; $V_{\text{out}} < 0$ V

Algorithm 3.6: `updateRegionGraphs`$(N, \sigma, reg, rate, \theta)$

```
1  let $N = (P, T, B, V, F, En, D, BA, VA, RA, M_0, S_0, Q_0, R_0)$;
2  $reg_{prev}$ := null;
3  forall $i \in [0, |\sigma|]$ do
4      if $p(reg_i) \notin P$ then
5          $P := P \cup$ newP$(reg_i)$;
6      if $t(p(reg_{prev}), p(reg_i)) \notin T)$ then
7          $T := T \cup$ newT$(N, p(reg_i), p(reg_{prev}), (0,0), \theta,$ diffR$(reg_i, reg_{prev}))$;
8      forall $t \in T$ do
9          if $(t, p(reg_i)) \in F$ then
10             if $rate_i < r_l(t)$ then
11                 $r_l(t) := rate_i$;
12             if $rate_i > r_u(t)$ then
13                 $r_u(t) := rate_i$;
14     $reg_{prev} := reg_i$;
```

lation trace looking for constant runs in the simulation trace marked using the *start* and *end* sets created by `detectDMV`. The algorithm begins by initializing the previous value variable and looping through each DMV variable (lines 2-3). Given a DMV variable ν, the algorithm begins with the first time value in the *start* set (lines 4-5). When a start point is found, the algorithm then searches for the next end point (lines 6-7). Once the start and end points are found, the enabling region and delay range are calculated, and the range of values for the given variable in the given constant run are extracted (lines 8-10). To determine the enabling region, the causal event set must be determined (line 8). The causal event set is determined by finding the region previous to the one where the constant run begins. The set of variables that remain constant between the regions is the causal set. If no variables are constant, then the previous region is examined, etc. The delay range is calculated using the `calcD` function (line 9). The lower bound of the range is calculated from the time the trace enters the previous region until the previous constant run ends. The upper bound of the range is calculated from the previous region until the start of the current constant run. This range allows for the variable to change in the uncertain region between constant runs. V_{in} is a DMV variable in the switched capacitor integrator example and changes to its high value in region 11. The previous region is 01. In this case, V_{in} would be assigned to change to its high value based on the time V_{out} crosses above the threshold of zero. This simplistic causal calculation may result in the incorrect or inconsistent choice for a causal event. For example, although a change on

V_{out} may appear to cause V_{in} this is not the case. This choice would result in a poor model as the rate of change of V_{out} would affect the time when V_{in} changes making V_{in} much less periodic. Therefore, a refinement to this first method is used. The user can specify the fact that there is not a causal relationship between two variables in *nonC*. As a result, these noncausal variables are ignored in the causal region calculation. If V_{out} is specified as noncausal in the previous example, V_{in} is the only remaining variable. If V_{in} is causal only with itself then the delay value is the amount of time V_{in} remains at a given value.

The value range is calculated by extracting the minimum and maximum values in the constant run. If a place does not already exist for this value range, then a new one is created (lines 11-12). For the integrator, place p_1 is added for V_{in} equal to $[-1000,-999]$ mV, and p_2 is added to represent that V_{in} is equal to [999,1000] mV. The next step is to create a transition between the current place and the previous place if one does exist (lines 13-14). Finally, the value and delay assignments are updated (lines 15-22). For the integrator example, the LHPN generated to control V_{in} is shown in Figure 3.6b. V_{in} is set to $[-1000,-999]$ mV and remains there for 100 μs to 101 μs after which it changes to [999,1000] mV and remains there for 99 μs to 100 μs as indicated in Figure 3.6.

3.3.5 Pseudo-regions

Although the model is built directly from simulation traces, during reachability analysis it is possible for the analysis to leave the regions explored by the simulation traces. Leaving the regions explored by the simulation traces can be useful in finding errors, but it may also exceed physical limitations (e.g., the voltage supply rail). Pseudo-regions are used to allow the model to leave regions exhibited by the simulation traces while remaining within physical bounds. As an example, consider a two variable system with a single threshold on each variable and the physical limitation that both variables remain between -1 and 1. In this case, there are four potential regions of operation labeled a-d in Figure 3.7. Rates for each variable in each region as well as the direction of transitions found between the regions are calculated during the model generation process. To add the limiting behavior to the model, our method adds pseudo-regions e-p shown in Figure 3.7 using dotted lines. These pseudo-regions would not be reachable via simulation as the simulator imposes the appropriate physical limits on the circuit. The state space

Algorithm 3.7: $\texttt{updateDMVGraphs}(N, \sigma, reg, dmv, start, end, nonC, \theta)$

```
let $N = (P, T, B, V, F, En, D, BA, VA, RA, M_0, S_0, Q_0, R_0)$;
$val_{prev}$ := null;
forall $\nu \in dmv$ do
    while $start(\nu) \neq \emptyset$ do
        $i = \texttt{min}(start(\nu))$;
        $start(\nu) = start(\nu) - i$;
        $j = \texttt{min}(end(\nu))$;
        $end(\nu) = end(\nu) - j$;
        $reg_{En} := \texttt{calcEn}(\sigma, i, nonC(\nu))$;
        $(\tau_l, \tau_u) := \texttt{calcD}(\sigma, i, j, reg_{En}, end)$;
        $(val_l, val_u) = val := \texttt{extractVals}(\sigma, \nu, i, j)$;
        if $p(val) \notin P$ then
            $P := P \cup \texttt{newP}(val)$;
        if $t(val_{prev}, val) \notin T$ then
            $T := T \cup \texttt{newT}(N, p(val_{prev}, val), (\tau_l, \tau_u), \theta, \texttt{diffR}(reg_{En}, reg_i))$;
        if $val_l < a_l(t(val_{prev}, val))$ then
            $a_l(t(val_{prev}, val)) := val_l$;
        if $val_u > a_u(t(val_{prev}, val))$ then
            $a_u(t(val_{prev}, val)) := val_u$;
        if $\tau_l < d_l(t(val_{prev}, val))$ then
            $d_l(t(val_{prev}, val)) := \tau_l$;
        if $\tau_u > d_u(t(val_{prev}, val))$ then
            $d_u(t(val_{prev}, val)) := \tau_u$;
        $val_{prev} := val$;
```

exploration engine for LHPNs, however, does not impose these same physical limitations and as such may end up exploring these pseudo-regions. In these pseudo-regions, the rate of change for the limited variable is modified to prevent the continuous variable from moving farther into a region of physical impossibility. For instance, in region a, if y increases above its upper threshold then it moves to pseudo-region f where it is prevented from increasing any further by changing $\dot{y} = [-5, 5]$ to $\dot{y} = [-5, 0]$. In region a, it is impossible for x to decrease below its limit as the rate of change for x is always positive. In this case, the rate is copied directly into pseudo-region i. The rates for all pseudo-regions are calculated in a similar manner. Figure 3.7 represents the possible paths through regions of operation by annotating the boxes with directional arrows. These arrows indicate potential transitions between regions of operation. The solid arrows represent transitions observed in the simulation data used to generate the model. The dashed arrows represent transitions between observed regions of operation and pseudo-regions.

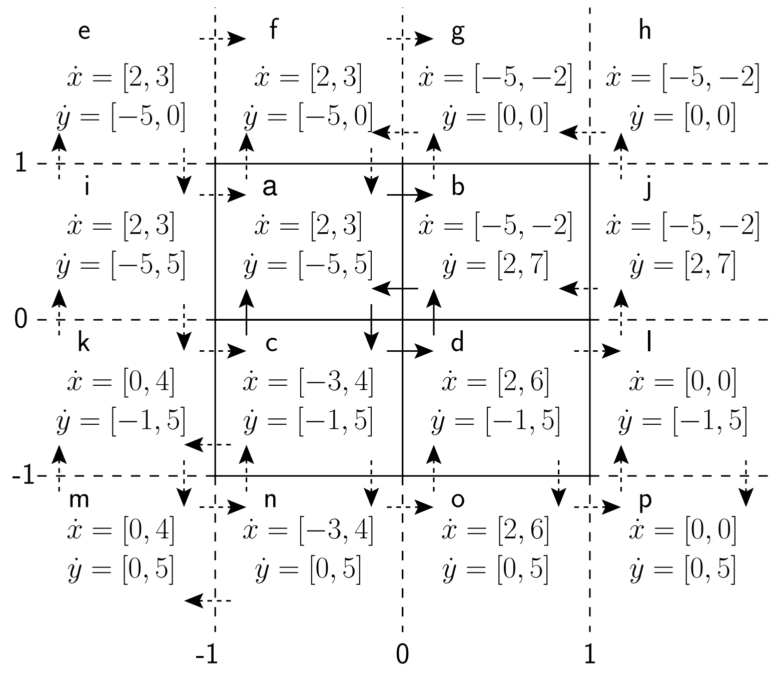
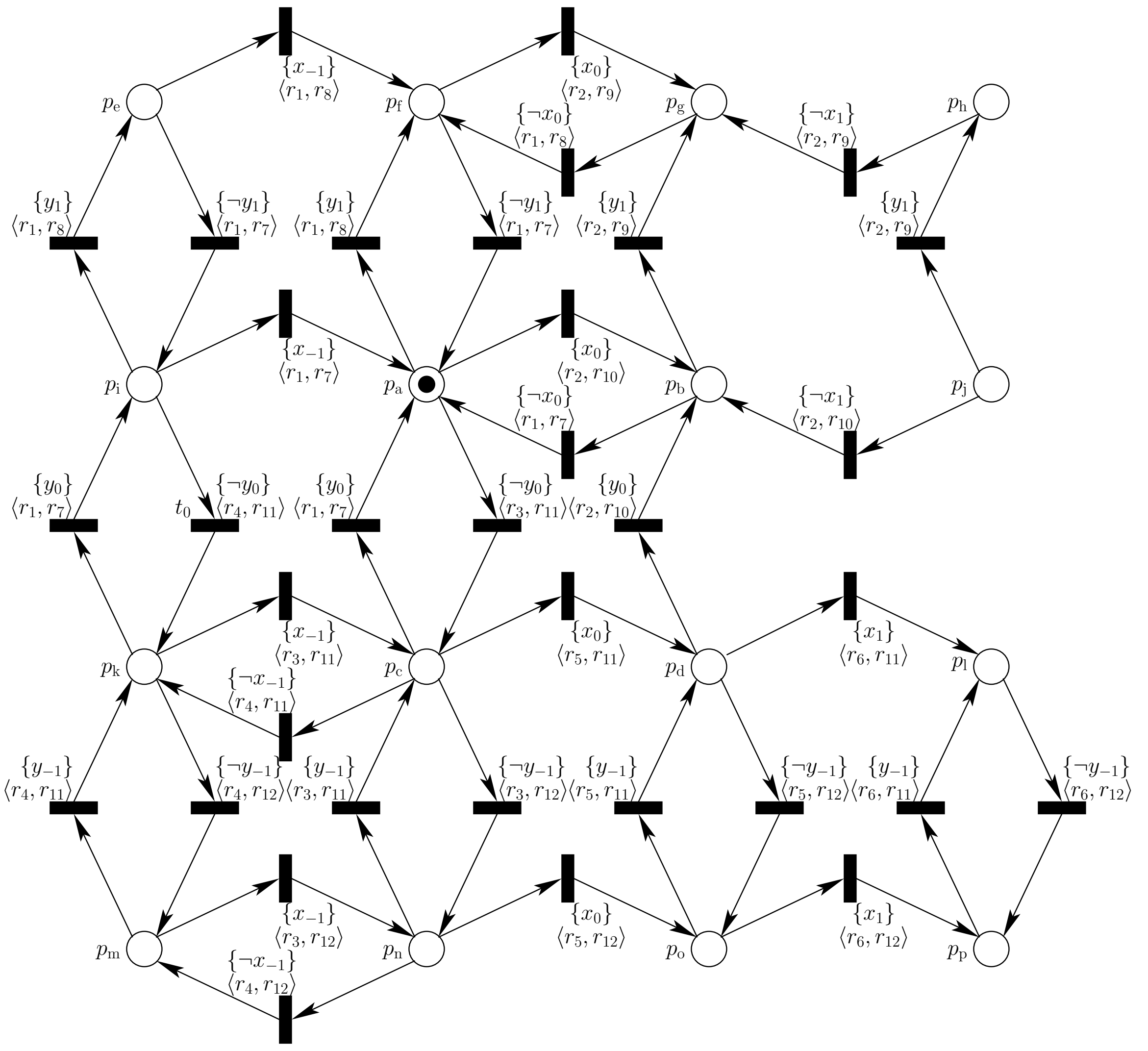


Figure 3.7. An example of a two variable system where each variable has a single threshold and an upper and lower value limit.

These transitions have not been observed in the simulation data but are possible based on the rates.

This box-like representation translates into a structurally similar LHPN as shown in Figure 3.8. Each region translates to a place in the LHPN while each arrow translates to a transition. The enabling conditions for each transition are derived from the thresholds and limits. For instance, when moving from region a to region b the enabling condition on the transition is $\{x \geq 0\}$ as x must cross 0 to change regions. The rate assignments are set to the rates calculated for the place where the transition leads. In the transition from region a to b, the rate is set to $\langle \dot{x} := [-5,-2], \dot{y} := [2,7] \rangle$, the rate of change for place p_b.

The addition of pseudo-regions and their associated transitions is done using Algorithm 3.8. The algorithm examines each variable of each node in the graph (line 2-5). For a given variable, if the rate of change for the variable is negative then a transition to a region with a lower threshold should be found or created (lines 6-18). The *region*(p) notation returns the region value for the given place. If there is a node where all variables of the system are in the same region except the selected variable, and the selected variable


p_e
p_f
p_g
p_h
p_i
p_a
p_b
p_j
p_k
p_c
p_d
p_l
p_m
p_n
p_o
p_p
t_0
$\{x_{-1}\}$ $\langle r_1, r_8\rangle$
$\{x_0\}$ $\langle r_2, r_9\rangle$
$\{\neg x_0\}$ $\langle r_1, r_8\rangle$
$\{\neg x_1\}$ $\langle r_2, r_9\rangle$
$\{y_1\}$ $\langle r_1, r_8\rangle$
$\{\neg y_1\}$ $\langle r_1, r_7\rangle$
$\{y_1\}$ $\langle r_1, r_8\rangle$
$\{\neg y_1\}$ $\langle r_1, r_7\rangle$
$\{y_1\}$ $\langle r_2, r_9\rangle$
$\{y_1\}$ $\langle r_2, r_9\rangle$
$\{x_{-1}\}$ $\langle r_1, r_7\rangle$
$\{x_0\}$ $\langle r_2, r_{10}\rangle$
$\{\neg x_0\}$ $\langle r_1, r_7\rangle$
$\{\neg x_1\}$ $\langle r_2, r_{10}\rangle$
$\{y_0\}$ $\langle r_1, r_7\rangle$
$\{\neg y_0\}$ $\langle r_4, r_{11}\rangle$
$\{y_0\}$ $\langle r_1, r_7\rangle$
$\{\neg y_0\}$ $\langle r_3, r_{11}\rangle$
$\{y_0\}$ $\langle r_2, r_{10}\rangle$
$\{x_{-1}\}$ $\langle r_3, r_{11}\rangle$
$\{x_0\}$ $\langle r_5, r_{11}\rangle$
$\{x_1\}$ $\langle r_6, r_{11}\rangle$
$\{\neg x_{-1}\}$ $\langle r_4, r_{11}\rangle$
$\{y_{-1}\}$ $\langle r_4, r_{11}\rangle$
$\{\neg y_{-1}\}$ $\langle r_4, r_{12}\rangle$
$\{y_{-1}\}$ $\langle r_3, r_{11}\rangle$
$\{\neg y_{-1}\}$ $\langle r_3, r_{12}\rangle$
$\{y_{-1}\}$ $\langle r_5, r_{11}\rangle$
$\{\neg y_{-1}\}$ $\langle r_5, r_{12}\rangle$
$\{y_{-1}\}$ $\langle r_6, r_{11}\rangle$
$\{\neg y_{-1}\}$ $\langle r_6, r_{12}\rangle$
$\{x_{-1}\}$ $\langle r_3, r_{12}\rangle$
$\{x_0\}$ $\langle r_5, r_{12}\rangle$
$\{x_1\}$ $\langle r_6, r_{12}\rangle$
$\{\neg x_{-1}\}$ $\langle r_4, r_{12}\rangle$

$x_1 = x \geq 1$	$r_1 = \dot{x} := [2,3]$	$y_1 = y \geq 1$	$r_7 = \dot{y} := [-5,5]$
$x_0 = x \geq 0$	$r_2 = \dot{x} := [-5,-2]$	$y_0 = y \geq 0$	$r_8 = \dot{y} := [-5,0]$
$x_{-1} = x \geq -1$	$r_3 = \dot{x} := [-3,4]$	$y_{-1} = y \geq -1$	$r_9 = \dot{y} := [0,0]$
	$r_4 = \dot{x} := [0,4]$		$r_{10} = \dot{y} := [2,7]$
	$r_5 = \dot{x} := [2,6]$		$r_{11} = \dot{y} := [-1,5]$
	$r_6 = \dot{x} := [0,0]$		$r_{12} = \dot{y} := [0,5]$

Figure 3.8. An LHPN for the regions graph shown in Figure 3.7.

is in a region with a lower threshold then an edge in the graph is added between those two nodes (lines 11-13). If a node with a lower threshold region for the selected variable is not present, a pseudo-region is added and an edge is added between the original node and the new node representing the pseudo-region (lines 14-18). In the pseudo-region, the rate of change is adjusted to prevent the variable from decreasing further in that region (line 17). A similar procedure is used if the rate of change for the selected variable is positive in the selected node (lines 19-31).

3.3.6 Writing Normalized LHPN Models

Since LEMA's DBM-based analyzer only supports integers, the values in the graph data structure must be normalized to produce a model analyzable by the DBM-based analyzer. The normalization process begins by scaling the minimum rate such that its integer value is represented using *sig* values. For instance, if *sig* is two and the minimum rate is 0.02 then all rates would be scaled by a factor of 1000 resulting in 0.02 being a rate of 20. If this process results in the maximum rate overflowing the integer space, LEMA reports an error and terminates. The next step is to adjust the constant values so that at least one integral time unit passes as a variable progresses between the thresholds at the new rates. The scaling of the constants involves scaling both thresholds and constant values for DMV variables such that there are *sep* orders of magnitude between the rates and constants. For instance, if the rate for a continuous variable is 20 and the thresholds are 0 and 1, the variable would pass between the thresholds in less than one time unit which poses a problem for the integer-based analysis. In this case, if the value of *sep* is one, the constants would be scaled to 0 and 100. This would now require five time units for the variable to pass between the thresholds.

Given the LHPN model generated in the previous steps, this structure is output verbatim. Additionally, an LHPN to check the safety property is created. This LHPN has a single initially marked place and a single transition. The transition's enabling condition is the complement of the safety property. This transition has a delay of [0,0] and indicates a failure when it fires by setting the Boolean signal *fail* to **true**. Therefore, to verify this safety property, a model checker only needs to determine if there exists any state in which this transition can fire. For the integrator example, the LHPN generated to check if the circuit can saturate is shown in Figure 3.6c.

Algorithm 3.8: `addPseudoRegions`(N, Σ, θ)

```
let $N = (P, T, B, V, F, En, D, BA, VA, RA, M_0, S_0, Q_0, R_0)$;
forall $p \in P$ do
    forall $t \in T$ do
        if $(p, t) \in F$ then
            forall $v \in V$ do
                if $r_l(t, v) < 0$ then
                    $p_{\text{new}} := p$;
                    $region_v(p_{\text{new}}) := region_v(p_{\text{new}}) - 1$;
                    $foundRegion :=$ False;
                    while $\neg foundRegion$ do
                        if $p_{\text{new}} \in P$ then
                            newT$(N, p, p_{\text{new}}, (0,0), \theta,$ diffR$(region(p), region(p_{\text{new}}))$);
                            $foundRegion :=$ True;
                        else if $region_v(p_{\text{new}}) = -1$ then
                            $P := P \cup p_{\text{new}}$;
                            newT$(N, p, p_{\text{new}}, (0,0), \theta,$ diffR$(region(p), region(p_{\text{new}}))$);
                            rmRateLtZero$(p_{\text{new}}, F, v)$;
                            $foundRegion :=$ True;
                    $region_v(p_{\text{new}}) := region_v(p_{\text{new}}) - 1$;
                if $r_u(t, v) > 0$ then
                    $p_{\text{new}} := p$;
                    $region_v(p_{\text{new}}) := region_v(p_{\text{new}}) - 1$;
                    $foundRegion :=$ False;
                    while $\neg foundRegion$ do
                        if $p_{\text{new}} \in P$ then
                            newT$(N, p, p_{\text{new}}, (0,0), \theta,$ diffR$(region(p), region(p_{\text{new}}))$);
                            $foundRegion :=$ True;
                        else if $region_v(p_{\text{new}}) = m$ then
                            $P := P \cup p_{\text{new}}$;
                            newT$(N, p, p_{\text{new}}, (0,0), \theta,$ diffR$(region(p), region(p_{\text{new}}))$);
                            rmRateGtZero$(p_{\text{new}}, F, v)$;
                            $foundRegion :=$ True;
                        $region_v(p_{\text{new}}) := region_v(p_{\text{new}}) - 1$;
    return $N$;
```

3.3.7 Writing VHDL-AMS Models

There two types of system models produced by the model generator based on how the system variables are modeled. The first type is a self-contained model that can be simulated without the aid of an external test bench. The second type is an abstract model intended to replace the transistor-level model, so it contains the same inputs and outputs as the original circuit. These two model types are derived based on user provided classification of the system variables. The user can classify a system variable as one of three types: input, output, or internal. Input variables are not modeled by the system, but the resulting model contains an input port where the system expects to receive external input for this variable (e.g., from a test bench). Output variables are modeled and assigned to an output port. Internal variables are used in the model but no input or output ports are provided. Any unclassified variables are unmodeled and not included in the model generation process.

Figure 3.9 is the VHDL-AMS description created by the model generator when `Vin` and `Vout` are marked internal. The creation of a VHDL-AMS description from the graph begins by creating a real **quantity** for each internal variable in the system. In this case, real variables are created for `Vin` and `Vout`. Each of these variables is assigned an initial value using a **break** statement which is −1000 mV for both `Vin` and `Vout`. DMV variables, `Vin` in this case, are assigned an initial rate of 0.0 using the `'dot` notation. This initial rate of 0.0 is never changed in the remaining description. The rates for non-DMV variables are set with nested **if-use** statements based upon the threshold values for each region. For instance, the **if** statement models region 00 where both `Vin` and `Vout` are below zero. The rate is set to $[17, 24]$ is this region as indicated by Table 3.4. The VHDL-AMS description supports ranges of rates using the `span` procedure, defined in Section 2.4, that accepts two real values and returns a random value within that range. The constant value assignments for DMV variables are specified using **process** statements without sensitivity lists. The **wait** statement waits for the proper Boolean condition and then waits for a range of delays before performing the assignment. In this example, the Boolean condition is implicit, so `Vin` is assigned to 1000 mV in 100 to 101 μs after it goes low and then assigned a value of −1000 mV in 99 to 100 μs after it goes high. Finally, **assert** statements are used to describe basic safety properties about the system using the restricted HSL grammar presented previously. For this example, the **assert** statement is used to check if *Vout* falls below −2000 mV or goes above 2000 mV.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.handshake.all;
use work.nondeterminism.all;
entity swCap is
end swCap;
architecture behavioral of swCap is
  quantity Vin:real;
  quantity Vout:real;
begin
  break Vin => -1000.0;
  break Vout => -1000.0;
  Vin'dot == 0.0;
  if not Vin'above(0.0) and not Vout'above(0.0) use
    Vout'dot == span(17.0,24.0);
  elsif not Vin'above(0.0) and Vout'above(0.0) use
    Vout'dot == span(17.0,24.0);
  elsif Vin'above(0.0) and Vout'above(0.0) use
    Vout'dot == span(-24.0,-17.0);
  elsif Vin'above(0.0) and not Vout'above(0.0) use
    Vout'dot == span(-24.0,-17.0);
  end use;
  process begin
    wait for Vin'above(0.0);
    wait for delay(100,101);
    break Vin => 1000;
    wait for Vin'above(0.0);
    wait for delay(99,100);
    break Vin => -1000;
  end process;
  assert (Vout'above(-2000.0) and not Vout'above(2000.0))
    report "Error:  The output voltage saturated."
    severity failure;
end behavioral;
```

Figure 3.9. VHDL-AMS code for the switched capacitor integrator circuit when both `Vin` and `Vout` are marked as internal variables.

Figure 3.10 is the VHDL-AMS code generated when `Vin` is marked as an input variable and `Vout` is marked as an output variable. In this case, `Vin` is exposed externally as an **in real** port, and `Vout` is exposed externally as an **inout real** port. Because `Vin` is an input to the system, no value or rate assignments for `Vin` are generated. As an output, the value and rate assignments for `Vout` remain the same as in the previous example.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.handshake.all;
use work.nondeterminism.all;
entity swCap is
  port(
    quantity Vin:in real;
    quantity Vout:inout real;
  );
end swCap;
architecture behavioral of swCap is
begin
  break Vout => -1000.0;
  if Vin'above(0.0) and Vout'above(0.0) use
    Vout'dot == span(-24.0,-17.0);
  elsif Vin'above(0.0) and not Vout'above(0.0) use
    Vout'dot == span(-24.0,-17.0);
  elsif not Vin'above(0.0) and Vout'above(0.0) use
    Vout'dot == span(17.0,24.0);
  elsif not Vin'above(0.0) and not Vout'above(0.0) use
    Vout'dot == span(17.0,24.0);
  end use;
  assert (Vout'above(-2000.0) and not Vout'above(2000.0))
    report "Error:  The output voltage saturated."
    severity failure;
end behavioral;
```

Figure 3.10. VHDL-AMS code for the switched capacitor integrator circuit when `Vin` is modeled as an input and `Vout` is modeled as an output.

3.3.8 Writing Verilog-AMS Models

In addition to producing self-contained and test bench style models, `LEMA`'s model generator can produce both deterministic and nondeterministic models. The Verilog-AMS model generation supports range of rates (nondeterminism) using the `$rdist_norm`()

function, but the model presented in this section is generated using the deterministic model generation option. As a result, ranges are averaged in this Verilog-AMS model as well as other deterministic models produced by the model generator. Figure 3.11 is Verilog-AMS code for the switched capacitor integrator circuit where `Vin` is marked as an input and `Vout` is marked as an output. Model generation begins by creating a top level **inout** variable for each input or output variable in the graph. `Vin_io` and `Vout_io` are the top level variables in the Verilog-AMS description. This is the only information provided by the Verilog-AMS model for input variables. The **real** variables of the form ⟨`varname`⟩`_var` are created to hold the current value of each output or internal variable, `Vout_var` for the switched capacitor integrator example. Variables with a rate (non-DMV variables) are also provided a **real** variable to store the current rate, for example `Vout_rate`.

The initial conditions for each variable and rate are set in the **initial_step** statement. Each edge containing a rate or constant value assignment is translated into a **cross** statement. The parameters for the **cross** statement are extracted from the nodes between the edge. The source and sink nodes are compared. The variable for the changing region is set as the compare variable. The threshold representing the division between the changing regions is the numerical value. The direction of the cross statement is calculated based on the signal's direction. If the signal is increasing, a '**1**' is used, and if the signal is decreasing, a '**-1**' is used. In Figure 3.11, the first **cross** statement is created for an edge where V_{in} changes from 1000 mV to -1000 mV by crossing the 0 V threshold. As a result, `Vin_io` is the compare variable; **0.0** is the numerical value; and '**-1**' is the direction. The sink place sets the rate of `Vout` to `0.020`, so this assignment is made to `Vout_rate` in the execution block of the **cross** statement. If there are multiple differences, an **if** statement is used in place of the **cross** statement. The **if** statement is considered a less optimal solution as it has a weaker interaction with the simulator and does not provide a method to specify the direction of signal change. A global timer is added to update all rates at an appropriate interval. For the switched capacitor integrator, an interval of 1 μs is used to update the value of `Vout_var` based on `Vout_rate`. Finally, a **transition** statement is added for each output variable to quickly transition the value of the internal variable to the external interface.

```
`include "disciplines.h"
module swCap(Vin_io,Vout_io);
  inout Vin_io, Vout_io;
  electrical Vin_io, Vout_io;
  real Vout_var, Vout_rate;
  analog begin
    @(initial_step) begin
      Vout_var = -1.00;
      Vout_rate = 0.020;
    end
    @(cross(V(Vin_io)-0.0,-1)) begin
      Vout_rate = 0.020;
    end
    @(cross(V(Vin_io)-0.0,1)) begin
      Vout_rate = -0.020;
    end
    @(timer(0.0,1e-06)) begin
      Vout_var = Vout_var + Vout_rate;
    end
    V(Vout_io) <+
      transition(Vout_var,1p,1p,1p);
  end
endmodule
```

Figure 3.11. Verilog-AMS code for the switched capacitor integrator circuit using the deterministic model generation option when `Vin` is modeled as an input and `Vout` is modeled as an output.

3.4 Coverage Metrics

Coverage information can be extracted from a set of simulations and is key to `LEMA`'s abstract model generation methodology. We have done some initial investigation into coverage metrics by implementing a metric where each simulation trace is given a score and uncrossed thresholds are reported. A higher score is achieved by a simulation trace that exhibits behavior not previously seen. From the perspective of the LHPN model, new behavior is entering a previously unvisited region, taking a previously untaken region-to-region transition, or altering the range of a recorded value (rate, constant value, or delay). A metric of this type gives a qualitative measure of the utility of an additional simulation trace. This type of metric could be used as an aid to determine the benefit of doing further simulations. Uncrossed thresholds potentially indicate an inadequate simulation set as the thresholds should characterize the operating regions of the system.

The simulation set should provide information for all operating regions of the system.

The coverage score for a given simulation is calculated using the following formula with unity weights: $cvg := p \cdot w_p + t \cdot w_t + c \cdot w_c + d \cdot w_d$ where:

- p is the number of new places;
- w_p is the places weight;
- t is the number of new transitions;
- w_t is the transitions weight;
- c is the number of times a range of rate or constant value is updated;
- w_c is the rates and constant values weight;
- d is the number of times a delay range is updated; and
- w_d is the delay weight;

For the integrator example, using just the simulation trace shown in Table 3.1 with C_2 equal to 23 pF would result in an LHPN with the same structure but different rates as the LHPN shown in Figure 3.6 and produce a coverage score of 200 using weights of one. Adding the simulation trace shown in Table 3.2 with C_2 equal to 27 pF results in the exact same LHPN structure, but the ranges of rate for V_{out} would be changed. Therefore, the value of the second trace run is only 94. Finally, if a third trace with C_2 equal to 25 pF is added at this point, the resulting LHPN would not change at all as the rates generated from this trace would be contained in those generated from the first two. Therefore, this trace adds no new knowledge, so the coverage metric would say that it has no value. As a final example, if a trace is added that changes V_{in} to be twice the frequency (i.e., it changes every 50 μs), it now becomes possible for V_{in} to change before V_{out} goes above 0V. This means that the LHPN generated would now have a new transition from p_6 to p_5. This LHPN would also have a wider range of delays for when V_{in} changes. Therefore, this additional trace results in a score of 67.

CHAPTER 4

DIFFERENCE BOUND MATRICES

Formal verification has the promise of verifying properties over the entire state space of the design including ranges of variation in parameters and initial conditions. It is used with great success in several industrial settings for digital circuits. This success for digital circuits has prompted exploration of formal verification for AMS circuits. A major hurdle for formal verification of AMS circuits is that AMS circuits complicate an already complex state space exploration process by requiring continuous quantities such as voltages and currents to be tracked accurately during state space exploration. As a result, one key component to the state space exploration process of AMS circuits is the state space representation.

This chapter discusses several state space representations for AMS circuits and hybrid systems reachability analysis. Next, the chapter introduces *difference bound matrices* (DBMs), the state space representation used by `LEMA`'s DBM-based model checker. DBMs are the key to the efficient state space representation used in `LEMA`'s DBM-based model checker. A novel development which enables the state space exploration of LHPNs using DBMs is the process of *warping*. This chapter concludes with a derivation of the algorithm for warping DBMs.

4.1 Related Work

Model checking is a common approach to formal verification of hybrid systems. One key component to any model checking algorithm is the state space representation. The major criteria in selecting a state space representation are the ability to accurately represent the state space and efficiently manipulate the representation. These two criteria are in conflict as accurate models are computationally expensive to manipulate and store. Numerous state space representations have been proposed, and the remainder of this section discusses the most prominent representations.

Hybrid systems can contain nonlinear dynamics, so it is ideal if the state space representation is able to exactly or very accurately represent nonlinear polygons. *Level sets* are represented using *Hamilton-Jacobi equations* which allows them to exactly represent nonlinear dynamics with discontinuities [113]. While very accurate, the method is not very efficient. Results are presented on hybrid systems examples with three states and three dimensions which require over an hour to complete reachability analysis.

As a result of the high complexity required to exactly model the nonlinear dynamics of the system, the majority of hybrid systems state space representations approximate the nonlinearities of the state space using a form of *convex polygons*. A convex polygon is a polygon where for any two points within the polygon, the midpoint of a line drawn between the two points is also in the polygon. Representing the state space as *arbitrary convex polygons* is the most accurate and expensive way to represent a nonlinear state space while being limited to convex polygons. This method is employed in many different ways [81, 55, 158, 156, 157, 155]. Initially, in [81] arbitrary convex polygons were represented using inequalities of variables and integer constants. This method was inefficient and prone to integer overflow. The work in [55] refined the method in [81] by employing the Parma Polyhedra Library [20] to help mitigate the integer overflow problems. Another feature of [55] is that the accuracy of the convex polyhedra can be tuned to improve efficiency. Because *binary decision diagrams* (BDDs) are used successfully to represent and manipulate the state space for digital systems, there are several methods that use BDDs [156, 157] or a BDD-like data structure, *hybrid restriction diagrams* (HRDs) [158], to represent and manipulate the state space. The BDD-based algorithm manipulates the state space by creating BDD nodes representing linear inequalities then performing standard BDD operations. HRDs represent conjunctions of convex polyhedra which may actually be concave. A *satisfiability modulo theories* (SMT) solver is used to explore a bounded version of the state space represented directly using inequalities representing arbitrary convex polyhedra [155]. While arbitrary convex polyhedra are a good choice from an accuracy standpoint, they are not very efficient.

During state space exploration, each state space exploration step requires that the reachability algorithm union the new state with the states found in previous iterations. This can be an expensive operation. To help ease the computation involved in the union operation, *flowpipes* [38, 39] and *orthogonal polyhedra* [14] are proposed. Flowpipes are sequences of arbitrary convex polyhedra. Even though efficiency is gained by a simpler union operation the number of faces required to accurately represent the state space grows prohibitively fast. To improve efficiency at the cost of accuracy, orthogonal polyhedra are finite unions of *hyperrectangles*. A hyperrectangle is an n-dimensional polygon using only 90° angles. Hyperrectangles are efficient to represent, but a large number of rectangles must be used to ensure an accurate approximation.

Oriented hyperrectangles [149] and *octagons* [111, 112] are proposed for improved efficiency over arbitrary convex polyhedra. Oriented hyperrectangles are hyperrectangles that have been rotated in space to represent the state space more accurately and efficiently than hyperrectangles that must conform to a predetermined orientation. Octagons obtain their efficiency by limiting the number of sides of each convex polygon to eight in two dimensions. These polyhedra can be represented by conjunctions of inequalities of the form $\pm x \pm y \leq c$.

As finding the right balance between efficiency and accuracy for convex polyhedra has proven difficult, several representations not based on convex polyhedra have been proposed. *Ellipsoids* [97, 30] are arbitrary elliptical surfaces. They are attractive because they handle increased dimensions efficiently. They lose accuracy due to the fact that unions of ellipsoids are not ellipsoids and as a result, conservative state space must be added to maintain the representation. In [151], it is proposed to use *eigenvectors* to represent the state space. These are only approximate for linear systems and thus add another level of approximation to the reachability analysis.

Recently, two promising state space representations have been proposed, *projectagons* [160] and *zonotopes* [61]. Projectagons are bounded polygons able to represent high dimensional, nonconvex state spaces thus addressing many accuracy concerns. Projectagons are formed by the intersection of a collection of prisms. Reachability results are provided for a seven-dimensional example. Zonotopes are proposed to address the computational concerns. Zonotopes are centrally symmetric convex polyhedra. They can also be represented as the *Minkowski sum* of a finite set of line segments. This property makes the union and intersection operations required by reachability analysis very efficient.

4.2 State Representation

To analyze and verify properties of LHPNs, reachability analysis needs to be performed. This analysis is complicated by the fact that LHPNs have an infinite number of states. Therefore, to perform reachability analysis on LHPNs, it is necessary to represent this infinite number of states using a finite number of state equivalence classes called *state sets*. Based upon their success in several timed circuits verification tools [161, 115, 28], `LEMA`'s DBM-based analysis method uses *zones* defined using DBMs to represent the continuous portion of the state space. DBMs are a restricted set of convex polygons that only allow 45° and 90° angles. This allows them to represent the state space more accurately than hyperrectangles and more efficiently than arbitrary convex polyhedra.

The state sets for `LEMA`'s DBM-based analyer are represented with the tuple $\psi = \langle M, S, Q, R, I, Z \rangle$ where:

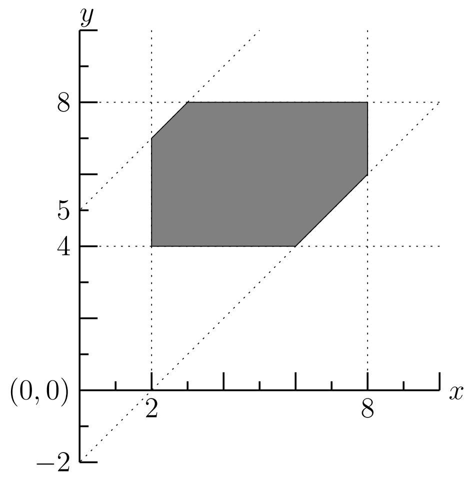
- $M \subseteq P$ is the set of marked places;
- $S : B \rightarrow \{0, 1\}$ is the value of each Boolean signal;
- $Q : V \rightarrow \mathbb{Z} \times \mathbb{Z}$ is the value of each inactive variable;
- $R : V \rightarrow \mathbb{Z}$ is the rate of each continuous variable;
- $I : v_i \geq k_i \rightarrow \{0, 1\}$ is the value of each inequality;
- $Z : (T \cup V \cup \{x_0\}) \times (T \cup V \cup \{x_0\}) \rightarrow \mathbb{Z}$ is a DBM composed of active transition clocks, active continuous variables, and x_0 (a reference variable which is always 0).

State sets and states differ in two ways. In a state set, inactive continuous variables (i.e., $R(v) = 0$) may have a range of values, and a DBM Z represents the ranges of values for clocks and active continuous variables. In the algorithms describing manipulation of state sets, there are several notations used to access pieces of the state set. The notations $q_l(v)$ and $q_u(v)$ are used to access the lower and upper values of variable v in the set of inactive continuous variables Q. The notation $R(v)$ is used to access the current rate for the variable v. The notation $v(i)$ is used to access the variable of inequality i and $k(i)$ is used to access the constant value of inequality i. The notation $x \in Z$ checks if the DBM Z is defined for x where x is a transition clock or continuous variable. The notation $Z \cup x$ indicates that another dimension is added to the DBM to accommodate the new transition clock or variable x while $Z - x$ indicates that x's dimension has been removed from the DBM. More notation is given for accessing the components of the DBM in the next section which provides an in depth discussion of DBMs.

4.3 Difference Bound Matrices

DBMs are a restricted form of convex polygons [51, 27, 137]. DBMs are restricted to using only 45° and 90° angles to form the convex polygons. This restriction means that states must evolve at a rate of one. Convex polygons are represented using a conjunction of inequalities of the form $x_i - x_j \leq k_i$ where x_i or x_j is the value of a transition's clock or a continuous variable, and k_i is a constant value or the symbol ∞ representing unboundedness. x_i or x_j can also be x_0 which is a reference variable that is always zero. For example, the convex polygon in Figure 4.1a can be specified using the equations in Figure 4.1b.

DBMs organize the sets of inequalities into an n-dimensional square matrix where n is the number of variables plus 1 (x_0). Figure 4.2a shows this organization. The inequalities in the matrix describe the separation between each of the variables in the polygon. The diagonal of the matrix is the separations between the variable and itself. Therefore, the diagonal in a valid DBM is filled with constants of zero. The first row of the matrix contains the lower bounds of the variables (i.e., $x_0 - x \leq -2$ and $x_0 - y \leq -4$). The equation $x_0 - v_i \leq k_i$ simplifies to $v_i \geq k_i$ when x_0 is replaced with zero (i.e., $x \geq 2$ and $y \geq 4$). The first column of the matrix contains the upper bounds of the variables,



(a)

$$
\begin{aligned}
x_0 - x &\leq -2 \\
x - x_0 &\leq 8 \\
x_0 - y &\leq -4 \\
y - x_0 &\leq 8 \\
x - y &\leq 5 \\
y - x &\leq 2
\end{aligned}
$$

(b)

Figure 4.1. A convex polygon and a set of inequalities to describe the polygon. (a) A convex polygon that can be represented using a DBM. (b) A system of inequalities describing the convex polygon in Figure 4.1a.

(i.e., $x - x_0 \leq 8$ and $y - x_0 \leq 8$). The equation $v_i - x_0 \leq k_i$ simplifies to $v_i \leq k_i$ when x_0 is replaced with zero, (i.e., $x \leq 8$ and $y \leq 8$). The remaining matrix entries contain separations between the variables in the DBM, (i.e., $x - y \leq 5$ and $y - x \leq 2$).

Given this organization for a DBM, the representation can be simplified by representing the inequalities using only the constants as shown in Figure 4.2b. It should be noted that the top row contains the negative of the lower bound for the variables. The notation `nlb`(Z, x) accesses the negative of the lower bound for x in Z (i.e., `nlb`$(Z, x) = -2$ for the DBM in Figure 4.2b) and `ub`(Z, x) accesses its upper bound (i.e., `ub`$(Z, x) = 8$ for the DBM in Figure 4.2b). The notation $\mathrm{Z}(i, j)$ is used to access the DBM entry containing the separation between i and j (i.e., $\mathrm{Z}(x, y) = 5$ for the DBM in Figure 4.2b).

To perform reachability analysis, it is necessary to union the newly found state into the previously explored state space. The union operation requires comparing DBMs. To efficiently perform these comparisons, a canonical form of a DBM is needed. For DBMs, canonicity is achieved when all bounds are maximally tight [51]. Maximal tightness is achieved by running Floyd's all pairs shortest path algorithm on the DBM as shown in the `recanonicalize` function in Algorithm 4.1.

Algorithm 4.1: `recanonicalize`(Z)

```
1 forall $x_i \in Z$ do
2     forall $x_j \in Z$ do
3         forall $x_k \in Z$ do
4             if $Z(x_j, x_k) > Z(x_j, x_i) + Z(x_i, x_k)$ then
5                 $Z(x_j, x_k) := Z(x_j, x_i) + Z(x_i, x_k)$
6 return Z;
```

$$\begin{bmatrix} x_0 & x & y \\ x_0 - x_0 \leq 0 & x_0 - x \leq -2 & x_0 - y \leq -4 \\ x - x_0 \leq 8 & x - x \leq 0 & y - x \leq 2 \\ y - x_0 \leq 8 & x - y \leq 5 & y - y \leq 0 \end{bmatrix}$$

(a)

$$\begin{bmatrix} x_0 & x & y \\ 0 & -2 & -4 \\ 8 & 0 & 2 \\ 8 & 5 & 0 \end{bmatrix}$$

(b)

Figure 4.2. A set of inequalities and the corresponding DBM. (a) Inequalities organized in the n-dimensional matrix form used by DBMs. (b) The DBM for the inequalities in Figure 4.2a.

The comparisons between DBMs in the union operation are used to exclude duplicate zones as well as zones that are smaller. To perform these comparisons, equality and subset relations for DBMs are defined. All other relationships (i.e., $\supset, \subseteq, \supseteq$) can be defined in terms of the equality and subset relations. Given a canonical form of the DBM, equality of two DBMs is determined using Algorithm 4.2. Two DBMs are equal when they contain the same variables and all entries are equal. To test equality, `equalDBM` verifies that both DBMs are the same size and contain the same variables (lines 1-4). For a given variable, the lower bounds, upper bounds, and separations between variables must also be equal between the two zones (line 4).

Algorithm 4.2: `equalDBM`(Z_1, Z_2)

```
1 if $|Z_1| = |Z_2|$ then
2     forall $x_i \in Z_1$ do
3         forall $x_j \in Z_1$ do
4             if $x_i \notin Z_2 \vee Z_1(x_i, x_j) \neq Z_2(x_i, x_j)$ then
5                 return False;
6     return True;
7 return False;
```

One DBM is considered to be a subset of another DBM if the two DBMs are not equal and all bounds on all variables are smaller or the same. One subtlety to this definition is that if a quantity does not appear in the zone, it is considered to be unbounded. For example, the DBM in Figure 4.3b is a subset of the DBM in Figure 4.3a. The DBM in Figure 4.3b is equivalent except for the variable y. In this case, y is considered unbounded meaning that the DBM in Figure 4.3b represents a subset of the state space represented by the DBM in Figure 4.3a. Algorithm 4.3 shows the algorithm to determine if Z_1 is a subset of Z_2. The DBMs are compared to ensure that Z_2 contains all variables contained in Z_1 (lines 1-3). Finally, the separations between all common variables are checked to ensure that the values for Z_1 are less than or equal to the values in Z_2 (lines 1-3). If all of these checks are satisfied, Z_1 is a subset of Z_2 (line 5).

$$\begin{bmatrix} x_0 & x \\ 0 & -2 \\ 8 & 0 \end{bmatrix}$$

(a)

$$\begin{bmatrix} x_0 & x & y \\ 0 & -2 & -4 \\ 8 & 0 & 2 \\ 8 & 5 & 0 \end{bmatrix}$$

(b)

Figure 4.3. A DBM and its subset. (a) A DBM. (b) A subset of the DBM in Figure 4.3a.

Algorithm 4.3: `subsetDBM`(Z_1, Z_2)

```
1 forall $x_i \in Z_1$ do
2     forall $x_j \in Z_1$ do
3         if $x_i \notin Z_2 \vee Z_1(x_i, x_j) \leq Z_2(x_i, x_j)$ then
4             return False;
5 return True;
```

4.4 Warping DBMs

The state space of an LHPN cannot be represented exactly in a DBM due to the requirement that all dimensions advance at rate one in a DBM. While clocks associated with transitions always increase at rate one, continuous variables may increase or decrease with any rate. Therefore, an approximation is necessary to analyze such a state space using DBMs. When a continuous variable advances with a rate other than one, a variable substitution is performed which has the effect of warping the DBM in the given dimension such that it advances with rate one. For example, in Figure 4.4a, a zone with two continuous variables, x and y, is shown. If x begins increasing at a rate of 2 and y begins increasing with a rate of 3, the warp occurs by substituting x with $\frac{x}{2}$ and y with $\frac{y}{3}$. This has the effect of warping the zone as shown in Figure 4.4b. Since a DBM can only represent polygons made with 45° and 90° angles, the zone in Figure 4.4b must be conservatively encapsulated in a larger zone which satisfies this requirement. The lighter gray box in Figure 4.4c shows the encapsulation that includes this zone while using only 45° and 90° angles. The final result of the zone being warped is shown in Figure 4.4d.

When a rate is negative, the DBM is first warped as described above and the resulting zone is then warped into the negative space. Warping into the negative space is accomplished by first swapping the lower and upper bounds in the zone. In the resulting zone, all 45° angles become 225° angles which cannot be represented in a DBM. This can be

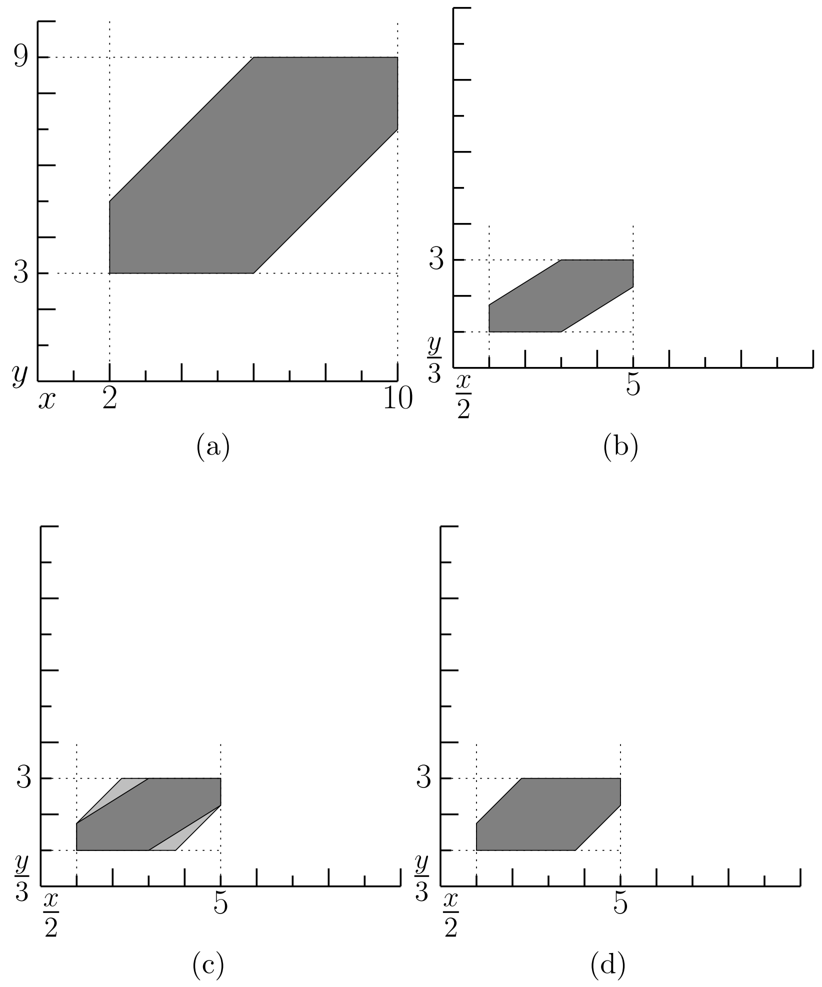


Figure 4.4. Warping a zone by positive rates in two dimensions.

seen in the darker box shown in Figure 4.5a. To address this problem, the algorithm must encapsulate the zone in a rectangle. The gray box in Figure 4.5b is the result of this encapsulation.

The following derivation explains the formal basis for warping the DBM. This derivation is done for the case of two dimensions. Cases that involve dimensions greater than two can be simplified to multiple applications of the two dimensional case.

Figure 4.6a is a zone whose DBM representation is shown in Figure 4.6b. The distances necessary for the warp calculations can be derived from Figure 4.6a using simple algebra, the slope-intercept equation ($y = mx + b$), and the fact that the slopes are always one. The result of these derivations is shown below.

$$d_1 = y_1 - b_1 - x_1$$
$$d_2 = x_1 + b_2 - y_1$$
$$d_3 = x_2 - y_2 + b_2$$
$$d_4 = y_2 - x_2 - b_1$$

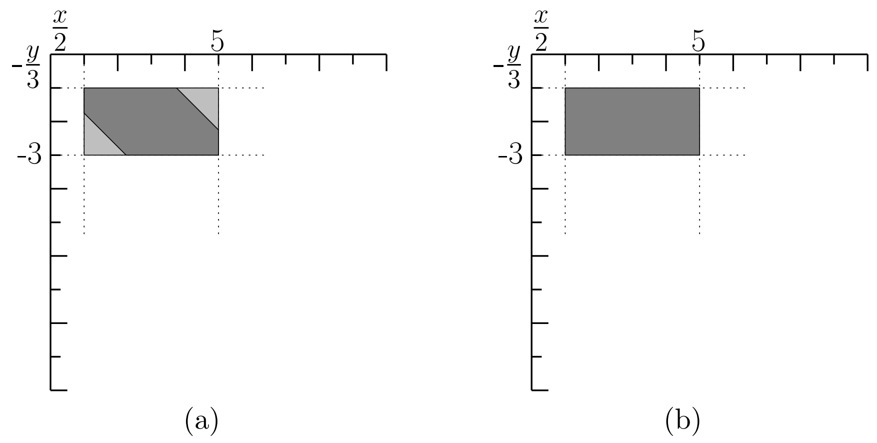
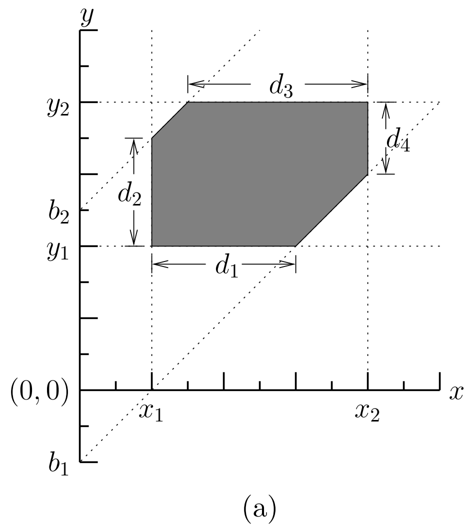


Figure 4.5. Warping a zone by a negative rate.



$$Z = \begin{bmatrix} 0 & -x_1 & -y_1 \\ x_2 & 0 & -b_1 \\ y_2 & b_2 & 0 \end{bmatrix}$$

(b)

Figure 4.6. A DBM and its corresponding zone. (a) A DBM representing the zone in Figure 4.6a. (b) A zone represented by the DBM in Figure 4.6b.

Given the above information, the zone is ready to be warped in both the x and y dimensions. α and β are ratios where the numerator is the old warp value and the denominator is the new warp value for the given dimension. The zone is warped by α in the x dimension and β in the y dimension as shown in Figure 4.7 for the $\alpha > \beta$ case and Figure 4.8 for the $\beta > \alpha$ case. The dark gray box represents the exact result of the warp while the light gray triangles indicate the space that is added to allow the warped polygon to fit within the restrictions imposed by the DBM representation. The first step in the calculation of the new zone is scaling the upper and lower bound by α and β as shown below.

$$
\begin{aligned}
x'_1 &= \alpha x_1 \\
x'_2 &= \alpha x_2 \\
y'_1 &= \beta y_1 \\
y'_2 &= \beta y_2
\end{aligned}
$$

To finish the warp calculation, the equations to calculate the values for the y intercepts, b'_1 and b'_2, must be derived. The derivations follow using the $b = y - mx$ form of the slope-intercept equation for a line where the slope is one. These calculations are dependent

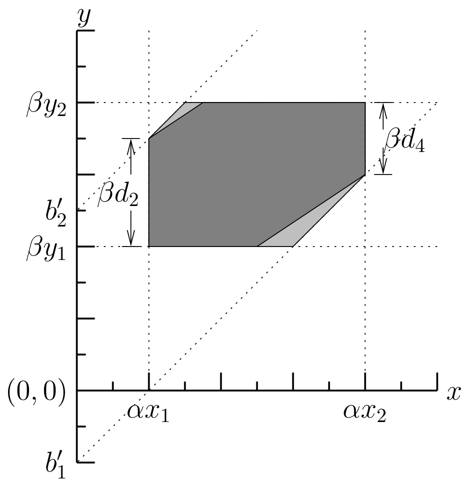


Figure 4.7. The zone from Figure 4.6a scaled by α in the x dimension and β in the y dimension where $\alpha > \beta$. The lighter polygon represents the new zone which encapsulates the warped zone.

on the relative values of α and β. When $\alpha > \beta$ the calculation of the new zone proceeds as follows:

$$\begin{aligned}
b_1' &= (\beta y_2 - \beta d_4) - \alpha x_2 \\
&= \beta y_2 - \beta(y_2 - x_2 - b_1) - \alpha x_2 \\
&= \beta y_2 - \beta y_2 + \beta x_2 + \beta b_1 - \alpha x_2 \\
b_1' &= \beta x_2 + \beta b_1 - \alpha x_2
\end{aligned}$$

$$\begin{aligned}
b_2' &= (\beta y_1 + \beta d_2) - \alpha x_1 \\
&= \beta y_1 + \beta(x_1 + b_2 - y_1) - \alpha x_1 \\
&= \beta y_1 + \beta x_1 + \beta b_2 - \beta y_1 - \alpha x_1 \\
b_2' &= \beta x_1 + \beta b_2 - \alpha x_1
\end{aligned}$$

Similarly, the zone could be warped such that $\beta > \alpha$ as shown in Figure 4.8. The calculation of b_1' and b_2' in this situation is performed as follows:

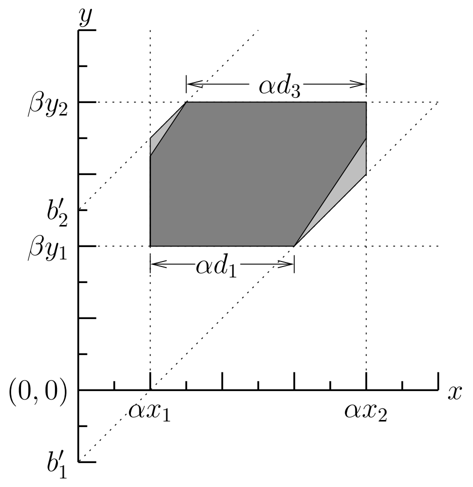


Figure 4.8. The zone from Figure 4.6a scaled by α in the x dimension and β in the y dimension where $\beta > \alpha$. The lighter polygon represents the new zone which encapsulates the warped zone.

$$\begin{aligned}
b_1' &= \beta y_1 - (\alpha x_1 + \alpha d_1) \\
&= \beta y_1 - \alpha x_1 - \alpha(y_1 - b_1 - x_1) \\
&= \beta y_1 - \alpha x_1 - \alpha y_1 + \alpha b_1 + \alpha x_1 \\
b_1' &= \beta y_1 - \alpha y_1 + \alpha b_1
\end{aligned}$$

$$\begin{aligned}
b_2' &= \beta y_2 - (\alpha x_2 - \alpha d_3) \\
&= \beta y_2 - \alpha x_2 + \alpha(x_2 - y_2 + b_2) \\
&= \beta y_2 - \alpha x_2 + \alpha x_2 - \alpha y_2 + \alpha b_2 \\
b_2' &= \beta y_2 - \alpha y_2 + \alpha b_2
\end{aligned}$$

Note that when $\alpha = \beta$, both b_1' and b_2' reduce to:

$$\begin{aligned}
b_1' &= \alpha b_1 = \beta b_1 \\
b_2' &= \alpha b_2 = \beta b_2
\end{aligned}$$

The equations presented for the calculations of b_1' and b_2' can be parameterized into a single function to simplify the implementation of this derivation. The equation is shown in Equation 4.1.

$$\texttt{warp}(z_1, z_2, r_1, r_2) = r_1 \cdot z_2 - r_1 \cdot z_1 + r_2 \cdot z_1 \tag{4.1}$$

It may also be necessary to scale a zone by a negative amount. In this case, the zone is first scaled by the absolute value of the warp value using the appropriate equations shown above. The next step involves the swapping of the values for the dimension(s) being warped negatively. In the equations shown below, the y dimension is being negatively warped and similar equations can be used when the x dimension is negatively warped.

$$\begin{aligned}
x_1' &= x_1 \\
x_2' &= x_2 \\
y_1' &= -y_2 \\
y_2' &= -y_1 \\
b_1' &= -y_1 - x_2 \\
b_2' &= -y_2 - x_1
\end{aligned}$$

The new zone is approximated with a rectangle as shown in Figure 4.9. The box encapsulation is a direct result of moving from a positive warp to a negative warp where

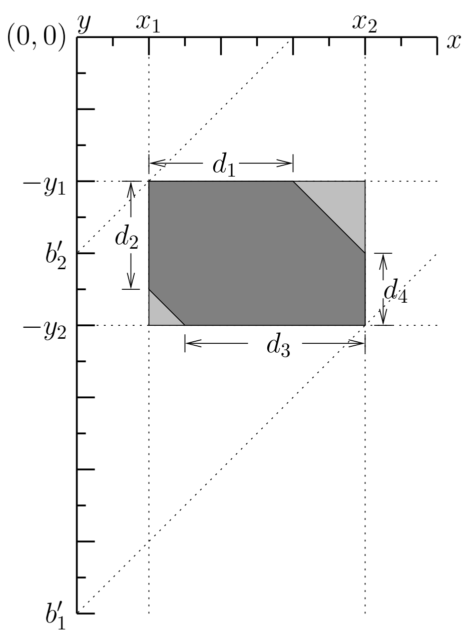


Figure 4.9. A zone warped from positive space into negative space.

the 45° angles of the zone in the positive space become unrepresentable 225° angles after the negative warp. These 225° angles are converted to 90° angles to be represented by the zone resulting in additional state space being conservatively introduced.

`LEMA`'s DBM-based model checker only supports integer values in the DBM. This restriction does not modify the state space exploration process for systems with variables changing only at a rate of one. However, when it is necessary to warp the DBM, the algorithms must be modified to ensure a conservative state space is represented. The problem arises because of the divide operation required by warping. To ensure that a conservative state space is found, two different divide functions are used, `cdiv` and `fdiv`. The `cdiv` function takes two integer values, performs a floating point division, and returns the ceiling of the result (i.e., $\lceil x/y \rceil$). The `fdiv` function works the same as the `cdiv` function except it returns the floor of the division (i.e., $\lfloor x/y \rfloor$).

This preceding derivation explains the formal basis for the `dbmWarp` algorithm, Algorithm 4.4, and in particular the `warp` function. The mapping between the variables used in the derivation and Algorithm 4.4 are as follows:

$$\begin{aligned}
-x_1 &= \texttt{nlb}(Z, x) \\
x_2 &= \texttt{ub}(Z, x) \\
-y_1 &= \texttt{nlb}(Z, y) \\
y_2 &= \texttt{ub}(Z, y) \\
-b_1 &= Z(x, y) \\
b_2 &= Z(y, x)
\end{aligned}$$

First, the algorithm performs the warping and encapsulation for positive rates (lines 1-12). After a variable has been warped, it is said to be in *warped space*. The third loop in Algorithm 4.4 is used when a rate is negative which requires that the values calculated in the previous parts of the algorithm to be warped into the negative warped space (lines 13-18). The resulting DBM Z is recanonicalized and returned (lines 19-20). The `warp` function used in Algorithm 4.4 is shown in Equation 4.1.

Algorithm 4.4: `dbmWarp`(R, R', Z)

```
1  forall $\{x, y\} | x \in Z, y \in Z, x \neq y$ do
2      $\alpha :=$ |fdiv($R(x), R'(x)$)|;
3      $\beta :=$ |fdiv($R(y), R'(y)$)|;
4      if $\alpha > \beta$ then
5          $Z(x, y) :=$ warp(ub($Z, x$), $Z(x, y), \beta, \alpha$);
6          $Z(y, x) :=$ warp(nlb($Z, x$), $Z(y, x), \beta, \alpha$);
7      else
8          $Z(x, y) :=$ warp(nlb($Z, y$), $Z(x, y), \alpha, \beta$);
9          $Z(y, x) :=$ warp(ub($Z, y$), $Z(y, x), \alpha, \beta$);
10 forall $x \in Z$ do
11     nlb($Z, x$) := cdiv($|R(x)|, |R'(x)|$)$*$ nlb($Z, x$);
12     ub($Z, x$) := cdiv($|R(x)|, |R'(x)|$)$*$ ub($Z, x$);
13 forall $x \in Z$ do
14     if $R(x)/R'(x) < 0$ then
15         $Z :=$ swap($Z$,nlb($Z, x$),ub($Z, x$));
16         forall $y \in Z$ do
17             if $x \neq y \wedge y \neq x_0$ then
18                 $Z(x, y) := Z(y, x) := \infty$;
19 $Z :=$ recanonicalize($Z$);
20 return $(R', Z)$;
```

4.5 Updating the State Set

After defining DBMs and introducing the notion of warping, the remaining algorithms for updating the state set can be described. Another set of basic operations on Z are the addition and removal of transition clocks and variables. The algorithm to add a transition clock to Z, `addT`, is shown in Algorithm 4.5. The transition clock is first added to Z (line 1). Then the remaining entries in Z for the newly added transition clock are set to unbounded for other variables and zero when the variable relates to itself (lines 2-7).

Algorithm 4.5: `addT`(Z, t)

```
1 $Z = Z \cup c_t$;
2 forall $x_i \in Z$ do
3     if $x_i = c_t$ then
4         $Z(c_t, x_i) := 0$;
5     else
6         $Z(c_t, x_i) := \infty$;
7         $Z(x_i, c_t) := \infty$;
8 return Z;
```

Algorithm 4.6 shows how a variable is added to Z using `addV`. The variable is inserted into Z (line 1). The remainder of the algorithm sets the correct values of the entries in Z. If the rate of the variable is greater than zero then the value of the variable's minimum value (i.e., $q_l(v)$) divided by the rate is placed in the lower bound of the DBM and the variable's maximum value (i.e., $q_u(v)$) divided by the rate is placed in the upper bound of Z (lines 2-4). Due to the fact that the bounds are swapped for negative rates, the process described above is opposite when the rate is less than zero (lines 5-7). Note that `cdiv` ensures that the variable is conservatively represented. For example, a value of 13 being warped by two would result in an exact value of 6.5. Using the `cdiv` function results in the conservative value of 7. To finish adding the variable, the separation between the newly added variable and the other variables in the DBM is set to unbounded and the relationship between the variable and itself is set to zero (lines 8-13).

The `rmT` algorithm removes transition clocks from the DBM (i.e., $Z = Z - c_t$). Removing a variable from the DBM is just the opposite of adding a variable to the DBM. Algorithm 4.7, `rmV`, performs a variable substitution out of the warped space (lines 1-6). The variable is then removed from the DBM (line 7).

Algorithm 4.6: `addV`(Q, R, Z, v)

```
$Z := Z \cup v$;
if $R(v) > 0$ then
    $\texttt{nlb}(Z, v) := \texttt{cdiv}(q_l(v), R(v))$;
    $\texttt{ub}(Z, v) := \texttt{cdiv}(q_u(v), R(v))$;
else
    $\texttt{nlb}(Z, v) := \texttt{cdiv}(q_u(v), R(v))$;
    $\texttt{ub}(Z, v) := \texttt{cdiv}(q_l(v), R(v))$;
forall $x_i \in Z$ do
    if $x_i = v$ then
        $Z(v, x_i) := 0$;
    else
        $Z(v, x_i) := \infty$;
        $Z(x_i, v) := \infty$;
return $(Q, Z)$;
```

Algorithm 4.7: `rmV`(Q, R, Z, v)

```
if $R(v) > 0$ then
    $q_l(v) := \texttt{nlb}(Z, v) * R(v)$;
    $q_u(v) := \texttt{ub}(Z, v) * R(v)$;
else
    $q_l(v) := \texttt{ub}(Z, v) * R(v)$;
    $q_u(v) := \texttt{nlb}(Z, v) * R(v)$;
$Z = Z - v$;
return $(Q, Z)$;
```

Algorithm 4.8 describes the algorithm to perform variable assignments, `doVarAsgn`. Each variable assignment on the given transition is done by removing the variable from Z, making the assignment, and adding the variable back into Z. The removal and addition are necessary to maintain the consistency of the relationships between the assigned variables and other members of Z. The procedure to assign rates and Boolean signals only assigns the rates and Boolean signals, and no additional state updates are needed.

Algorithm 4.8: `doVarAsgn`(Q, R, Z, VA, t)

```
1 forall $(v = [a_l, a_u]) \in VA(t)$ do
2     $(Q, Z) =$ rmV$(Q, R, Z, v)$;
3     $Q(v) = (a_l, a_u)$;
4     $(Q, Z) =$ addV$(Q, R, Z, v)$;
5 return $(Q, Z)$;
```

CHAPTER 5

DBM-BASED MODEL CHECKER

Model checking is a promising approach to formally verify AMS circuits. A number of different approaches and tools have been developed for AMS circuit model checking. Several decisions are made when developing an AMS circuit model checker, namely the model to be analyzed, the state space representation, and the types of properties that can be verified.

This chapter discusses several model checking tools for hybrid systems and AMS circuit verification. Next, this chapter introduces a set of algorithms to perform reachability analysis of constant rate LHPNs using DBMs based upon a reachability analysis algorithm for analyzing timed systems [25, 26]. This chapter includes an error trace generation method for LHPNs and concludes with an example of reachability analysis on the motivating example.

5.1 Related Work

Chapter 2 discusses the pros and cons of the different models with respect to efficiently representing hybrid systems and creating formal models from commonly used modeling formalisms. When selecting a model for use in reachability analysis, one concern is the decidability of the model. In [82], Henzinger et al. provide a thorough exploration of the decidability of the reachability problem for hybrid system models and identify a boundary between decidability and undecidability. There are two constraints that must hold for a hybrid system model to be decidable. The first is that the values of two continuous variables with different rates are never compared. The second is that whenever a continuous variable's rate changes, the continuous variable must be reinitialized. These constraints make reachability analysis of most hybrid system models, including LHPNs, undecidable.

The two major property classifications are *safety properties* and *liveness properties*. Safety properties specify that nothing bad happens and can be checked on finite traces.

Liveness properties specify that something good happens eventually and are satisfied by infinite traces. A class of liveness properties, *bounded liveness properties*, are bounded by time. These bounded liveness properties are represented as safety properties because they can be checked using finite traces. For simplicity, most hybrid system verification tools only support safety properties and possibly bounded liveness properties.

To illustrate how these considerations work in practice, the remainder of this section discusses the implementation of several tools for hybrid system and AMS circuit verification. HyTech [81, 10, 6] and `red` [158] are tools that operate on *linear hybrid automata* (LHA). LHA are a restricted set of hybrid automata that only allow rates governed by linear constraints on the variables and their first derivatives. Nonlinear behaviors can be approximated by LHA using piecewise linear envelopes to approximate the nonlinearities. The differences in these tools become evident when examining how the state space is represented in the tool. Both tools use convex polyhedra in the state space exploration, but `red` uses HRDs to represent sets of convex polyhedra as concave polyhedra. The `red` tool only verifies safety properties while HyTech supports the verification of safety properties and time-bounded properties. Both methods perform exact reachability.

TReX [12] is a modular tool supporting any type of data structure that provides a symbolic representation structure, a symbolic successor/predecessor function, and an extrapolation procedure. It can be exact or not based on the exactness of the given data structures and functions. TReX checks safety properties and generates a symbolic graph of the reachable state space which can be processed by other tools. An example of a model supported by TReX is *parametric difference bound matrices* (PDBMs) [11]. This model is not a true hybrid model, but it is more powerful than timed automata. PDBMs can be used to analyze linear and a subset of nonlinear constraints. There is, however, only a single continuous quantity available in the analysis which is limiting.

The **d/dt** tool [16] operates on a restricted set of hybrid automata that allow rate equations described by linear differential equations. This model is more complicated than LHA but does not directly model nonlinear dynamics. Convex polyhedra are also used by **d/dt** to perform the individual state exploration steps, but orthogonal polyhedra are used to represent the entire state space. The reachability analysis is approximate and can verify safety properties.

CheckMate [42, 145, 144, 37] uses *polyhedral-invariant hybrid automata* (PIHA) as a model. PIHA support a very general set of rate values including ordinary differential equations to verify safety properties. The verification procedure is conservative and uses flowpipe approximations, which are unions of convex polyhedra, to explore the state space. False negatives are handled by a partition refinement procedure that depends on a test of bisimulation. Due to the refinement procedure, CHECKMATE can perform exact reachability analysis. However, the iterative process to find a bisimulation might not terminate and in this case the reachability analysis is conservative.

PHAVer extends the work done by CHECKMATE. It analyzes LHA models of AMS circuits using arbitrary convex polyhedra to represent the exact continuous state space [55, 56, 57, 58]. While arbitrary convex polyhedra are quite complex, one unique feature of PHAVer is that it allows for performance to be tuned at the expense of a conservative state space by reducing the accuracy of the polyhedra. PHAVer demonstrates the effectiveness of this approach by verifying safety properties on several benchmark examples.

Two reachability analysis methods for hybrid net condition/event systems (HNCESs) are proposed in [34]. The first method is a predicate-transformation method based on LHA reachability algorithms. The second method is a path-based approach that enumerates all possible firing sequences of the discrete transitions. Both algorithms verify safety properties using constraint solving.

Amcheck divides the continuous state space into regions and represents each region in a Boolean manner [76]. From this decomposition, it creates a transition relation by selecting test points in each region to determine reachable next states. This Boolean abstraction allows it to model check safety properties using standard Boolean based approaches. While the model checking algorithms are exact, the approximation of the original system is conservative and may be very inaccurate.

LEMA's BDD-based model checking engine uses BDDs to represent the exact state space of the LHPN model [156]. This exact representation is very expensive, so the model checking engine is implemented in a conservative manner to reduce the complexity. To avoid the complexity of a full state space exploration, LEMA's SMT-based bounded model checker verifies safety properties on bounded length traces of the system [155]. The SMT-based method is exact, but only for the bounded length of the trace. Results of both model checking engines verifying safety properties on several benchmark examples demonstrate the effectiveness of this method.

5.2 Reachability Analysis of LHPNs Using DBMs

The top-level algorithm for `LEMA`'s DBM-based model checker is shown in Algorithm 5.1. The `modelCheck` algorithm takes as input N, T_{fail}, *allowDeadlock*, and $prop_{\text{CTL}}$. N is the LHPN representing the system being model checked. t_{fail} is a set of transitions that when fired indicate a failure of the system. The set of fail transitions may be empty. The value of *allowDeadlock* indicates whether the reachability analysis should stop upon finding a deadlock in the system. If the Boolean variable is **true** then the reachability algorithm continues upon finding a deadlock. Deadlock often indicates a problem in the model, but there are situations where a model may contain an intended deadlock (e.g., a system that terminates). $prop_{\text{CTL}}$ is a CTL property that if provided is checked against the state space of the LHPN model found by the reachability algorithm.

The model checking algorithm, Algorithm 5.1, performs reachability analysis on the LHPN (line 2). The **let** notation is used for convenience to demonstrate the contents of the tuple (line 1). This notation is used similarly in algorithms throughout the remainder of this chapter. If a CTL property has been specified, it is checked on the state space returned by the reachability algorithm (lines 3-4). If the CTL property fails then a pair of error traces are generated (lines 5-6).

Algorithm 5.1: `modelCheck`($N, T_{\text{fail}}, allowDeadlock, prop_{\text{CTL}}$)

```
1 let $N = (P, T, B, V, F, En, D, BA, VA, RA, M_0, S_0, Q_0, R_0)$;
2 $(\Psi, \Gamma, \psi_0, (\Lambda_{\max}, \Lambda_{\min}))$ := reach($N, t_{\text{fail}}, allowDeadlock$);
3 if $prop \neq \emptyset$ then
4     $(fail, \psi_{\text{fail}})$ := checkCTL($prop, \Psi, \Gamma$);
5     if $fail$ then
6         $(\Lambda_{\max}, \Lambda_{\min})$ := error($\psi_0, \psi_{\text{fail}}, \Psi, \Gamma, V, T$);
```

Algorithm 5.2 describes `LEMA`'s DBM-based reachability algorithm for LHPNs. The algorithm is a depth first search of the state space. The algorithm begins by constructing the initial state set of the LHPN and adding it to the set of reachable state sets, Ψ (lines 2-4). The `reach` algorithm then calls `findPossibleEvents` which determines all possible sets of event sets, $\mathcal{E}$, that can result in a new state set (line 6). Given this set of event sets, an *event set*, E, is arbitrarily chosen by the `select` function (line 8). After the selection of an event set, if event sets still remain in $\mathcal{E}$, the remaining event sets and the current state set are pushed onto the stack (lines 9-10). Given the current state set, ψ, and possible event set, E, the state set is updated (line 11). If the resulting state set has not been seen before, it is added to the set of reachable state sets; an edge is added into the state graph between the old and new state; the recently executed event set is checked for failure transitions; new events are found; deadlock is detected; and the loop continues (lines 12-22). If a failure trace has been fired or deadlock is found a pair of error traces are created and the current state space and error traces are returned (lines 16-18 and 20-22, respectively). If the state set has been seen before, an edge is added to the graph, the stack is popped and the loop continues (lines 23-26). If the stack is empty, the reachable state space has been found and is returned with an empty error trace (lines 27-28). The remainder of this section explains these steps in more detail.

Algorithm 5.2: `reach`$(N, T_{\text{fail}}, \mathit{allowDeadlock})$

```
1  let $N = (P, T, B, V, F, En, D, BA, VA, RA, M_0, S_0, Q_0, R_0)$;
2  $\psi_0 = (M, S, Q, R, I, Z) :=$ initialStateSet$(T, V, En, M_0, S_0, Q_0, R_0)$;
3  $\psi := \psi_0$;
4  $\Psi := \{\psi\}$;
5  $\Gamma := \emptyset$;
6  $\mathcal{E} :=$ findPossibleEvents$(T, En, D, R, I, Z)$;
7  while $true$ do
8      $E :=$ select$(\mathcal{E})$;
9      if $\mathcal{E} - \{E\} \neq \emptyset$ then
10         push$(\mathcal{E} - \{E\}, \psi)$;
11     $\psi' :=$ updateState$(T, V, En, D, BA, VA, RA, \psi, E)$;
12     if $\psi' \notin \Psi$ then
13         $\Psi := \Psi \cup \{\psi'\}$;
14         $\Gamma := \Gamma \cup \{(\psi, \psi')\}$;
15         $\psi := \psi'$;
16         if $E \subseteq T_{\text{fail}}$ then
17             $(\Lambda_{\max}, \Lambda_{\min}) =$ error$(\psi_0, \psi, \Psi, \Gamma, T, V)$;
18             return $(\Psi, \Gamma, \psi_0, (\Lambda_{\max}, \Lambda_{\min}))$;
19         $\mathcal{E} :=$ findPossibleEvents$(T, En, D, R, I, Z)$;
20         if $\mathcal{E} := \emptyset \wedge \neg \mathit{allowDeadlock}$ then
21             $(\Lambda_{\max}, \Lambda_{\min}) =$ error$(\psi_0, \psi, \Psi, \Gamma, T, V)$;
22             return $(\Psi, \Gamma, \psi_0, (\Lambda_{\max}, \Lambda_{\min}))$;
23     else
24         $\Gamma := \Gamma \cup \{(\psi, \psi')\}$;
25         if $stack\ not\ empty$ then
26             $(\mathcal{E}, \psi) :=$ pop();
27         else
28             return $(\Psi, \Gamma, \psi_0, \emptyset)$;
```

5.2.1 Initial State Set Construction

In Algorithm 5.3, the initial state set ψ is constructed. First, the marking M is set to the initial marking M_0 (line 1). The Boolean signals S are set to their initial state S_0 (line 2). The rates R are set to their initial rates R_0 (line 3). Initially, Q includes inactive continuous variables set to their initial value, Q_0, while the DBM Z includes active continuous variables (i.e., $R_0(v) \neq 0$) set to their initial value (lines 4-8). The clocks for enabled transitions are set to zero and added to Z (lines 9-11). Finally, I contains the initial value for all inequalities (i.e., $I(i) = (Q_0(v) \geq k)$) (lines 12-13). The function $\mathit{ineq}(\mathit{En})$ returns the set of inequalities found on the enabling conditions in N.

Algorithm 5.3: `initialStateSet`$(T, V, \mathit{En}, M_0, S_0, Q_0, R_0)$

```
1  $M := M_0$;
2  $S := S_0$;
3  $R := R_0$;
4  **forall** $v \in V$ **do**
5      **if** $R(v) \neq 0$ **then**
6          $(Q, Z) :=$ addV$(Q_0, R_0, Z, v)$;
7      **else**
8          $Q(v) := Q_0(v)$;
9  **forall** $t \in T$ **do**
10     **if** $t \in \mathcal{EN}(M_0, S_0, Q_0)$ **then**
11         $Z :=$ addT$(Z, t)$;
12 **forall** $(v \geq k) \in \mathit{ineq}(\mathit{En})$ **do**
13     $I(v \geq k) := Q_0(v) \geq k$;
14 **return** $(M, S, Q, R, I, Z)$;
```

5.2.2 Finding Possible Events

The `findPossibleEvents` algorithm shown in Algorithm 5.4 determines which events are possible from the current state. There are two types of possible events: a transition firing or an inequality changing value due to the advancement of time. Each event e is grouped into an event set E. An event set is a set of events that must occur simultaneously in the state space exploration. There are two types of event sets, a single transition and a set of inequalities. Transitions are not grouped together in sets because the transition clocks in the zone exactly measure their firing time. Because inequalities may include variables that are in warped space, it may be necessary for sets of inequalities to happen simultaneously. This happens when the time at which the constant value of the inequality is reached for two separate inequalities is indistinguishable in the warped space.

Algorithm 5.4: `findPossibleEvents`(T, En, D, R, I, Z)

```
1 $\mathcal{E} := \emptyset$;
2 forall $t \in Z$ do
3     if ub$(Z, t) \geq d_{\mathrm{l}}(t)$ then
4         $\mathcal{E} :=$ addSetItem$(T, En, D, R, Z, \mathcal{E}, t)$;
5 forall $i \in ineq(En)$ do
6     if ineqCanChange$(R, I, Z, i)$ then
7         $\mathcal{E} :=$ addSetItem$(T, En, D, R, Z, \mathcal{E}, i)$;
8 return $\mathcal{E}$;
```

A transition may fire anytime after the lower bound of the delay for that transition has been reached, and it must fire before the upper bound of the delay is exceeded. The notation used to access the lower bound of the delay is $d_l(t)$. The notation to access the upper bound of the delay is similar, $d_u(t)$. Clocks are activated when a transition becomes enabled and only enabled transitions are in Z. Therefore, any transition in Z whose clock can reach its lower bound may fire (i.e., `ub`$(Z, t) \geq d_{\mathrm{l}}(t)$) (lines 2-4). An inequality may change value when time can advance to the point where the value of the continuous variable associated with the inequality crosses the constant in the inequality (lines 5-7). This is determined in the `ineqCanChange` function by examining the current values of R, I, and Z with respect to i.

Once an event has been selected as a possible event, the `addSetItem` function is called to determine if this event can be the next to occur (lines 4 and 7). There are two possible outcomes. The first outcome is that the newly found event cannot actually happen before some other event already in the set of event sets $\mathcal{E}$, and it is not added to the set. The second outcome is that the newly found event can occur next, so it is added in the set of event sets, $\mathcal{E}$. Adding the newly found event to the set of event sets may cause other events already in $\mathcal{E}$ to be removed from $\mathcal{E}$ as this newly added event may occur before the previously added events.

The `addSetItem` function in Algorithm 5.5 divides the determination of whether an item is added to the current set of event sets into three main cases: both are inequalities, the new event is an inequality and the current event is a transition firing, and the new event is a transition firing and the current event is an inequality. The fourth case when both events are transition firings does not need to be considered because as mentioned previously, transition clocks are represented exactly in Z. The idea behind the three functions to handle these three cases is to examine the results of a `restrict` and `recanonicalize` for each event to determine which event actually happens first or if the two events would happen at the same time and should be included in the same event set.

The purpose of `restrict` is to modify Z to reflect that time must have advanced to the point necessary for the events in the event set to have occurred (i.e., the clock for the transition firing reaches its lower bound or the continuous variable reaches the constant in an inequality). To restrict Z to the point where the clock for a transition firing reaches its lower bound, the lower bound in Z for transition t is set to the lower bound of the delay for transition t (i.e., $\texttt{nlb}(Z, t) := d_l(t)$). To restrict Z to the point where a continuous variable reaches the constant in an inequality, the lower bound in Z for v is set to the result of the inequality's constant k divided by the rate of change for the variable $R(v)$ (i.e., $\texttt{nlb}(Z, v) := \texttt{cdiv}(k, R(v))$).

Algorithm 5.5: $\texttt{addSetItem}(T, En, D, R, Z, \mathcal{E}, e_{\text{new}})$

```
1  forall $E \in \mathcal{E}$ do
2      forall $e \in E$ do
3          if $e_{\text{new}} \in \mathit{ineq}(En) \wedge e \in \mathit{ineq}(En)$ then
4              $(E, \mathit{status}) := \texttt{happensFirstII}(R, Z, e_{\text{new}}, e, E)$;
5          else if $e_{\text{new}} \in \mathit{ineq}(En) \wedge e \in T$ then
6              $(E, \mathit{status}) := \texttt{happensFirstIT}(D, R, Z, e_{\text{new}}, e, E)$;
7          else if $e_{\text{new}} \in T \wedge e \in \mathit{ineq}(En)$ then
8              $(E, \mathit{status}) := \texttt{happensFirstTI}(D, R, Z, e_{\text{new}}, e, E)$;
9          if $\mathit{status} = \mathit{NotPossible} \vee \mathit{status} = \mathit{Possible}$ then
10             return $\mathcal{E}$;
11 $\mathcal{E} = \mathcal{E} \cup \{\{e_{\text{new}}\}\}$;
12 return $\mathcal{E}$;
```

The first case of `addSetItem` is executed when the newly found event and the event currently under inspection are both inequalities (lines 3-4). This case results in the calling of `happensFirstII`. The second case in `addSetItem` is executed when the newly found event is an inequality, and the event currently under inspection is a transition (lines 5-6). This case results in the calling of `happensFirstIT`. The third case in `addSetItem` is executed when the newly found event is a transition, and the event currently under inspection is a inequality (lines 7-8). This case results in the calling of `happensFirstTI`. Finally, if the status of the newly found event is decided, either *Possible* or *NotPossible*, the function returns (lines 9-10). If the status of the newly found variable remains *Undecided* after examining all of the event sets it is added to the set of event sets as a new, singleton event set (line 11).

The `happensFirstII` function, shown in Algorithm 5.6, determines how the DBM would be restricted for a change upon the corresponding inequalities for the newly found event, i_{new}, and the event under inspection, i (lines 3-4). If the inequalities are on the same variable, then the restrict values can be compared directly using the `compareSameV` function (line 6). If the two inequalities are on different variables, the determination of the event firing becomes more complicated as shown in Algorithm 5.8, `compareDifferentV` (line 8).

Algorithm 5.6: `happensFirstII`$(R, Z, i_{\text{new}}, i, E)$

```
1 let $i = (v \geq k)$;
2 let $i_{\text{new}} = (v_{\text{new}} \geq k_{\text{new}})$;
3 $restrictVal$ := cdiv($k$,$R(v)$);
4 $restrictVal_{\text{new}}$ := cdiv($k_{\text{new}}$,$R(v_{\text{new}})$);
5 if $v_{\text{new}} = v$ then
6     return compareSameV($i_{\text{new}}, i, E, restrictVal_{\text{new}}, restrictVal$);
7 else
8     return compareDifferentV($Z, i_{\text{new}}, i, E, restrictVal_{\text{new}}, restrictVal$);
```

The `compareSameV` function, shown in Algorithm 5.7, compares two inequalities on the same variable based on their restrict values. If the restrict value for the newly found event is greater (farther along in time) then it cannot happen first and is not added to the set (lines 1-2). If the restrict value for the event under inspection is higher then it cannot happen first and is removed from the event set, but nothing can be said about the possibility or impossibility of the newly found event (lines 3-5). If neither of the previous cases is true, then the two events are restricted to the same point in time and can happen simultaneously, and the newly found event is added to the event set (lines 6-8).

The `compareDifferentV` function, shown in Algorithm 5.8, compares two inequalities on different variables based on the values in Z after `restrict` and `recanonicalize`. If

Algorithm 5.7: `compareSameV`$(i_{\text{new}}, i, E, \mathit{restrictVal}_{\text{new}}, \mathit{restrictVal})$

```
1 if $\mathit{restrictVal}_{\text{new}} > \mathit{restrictVal}$ then
2     return $(E, \mathit{NotPossible})$;
3 else if $\mathit{restrictVal} > \mathit{restrictVal}_{\text{new}}$ then
4     $E := E - \{i\}$;
5     return $(E, \mathit{Undecided})$;
6 else
7     $E := E \cup \{i_{\text{new}}\}$;
8     return $(E, \mathit{Possible})$;
```

the restrict value for the newly found event would result in the DBM entry for the variable under inspection growing then the newly found event is not added to the set (lines 3-4 and 11-12). If the opposite case is true then the variable under inspection is removed from the event set but nothing can be said about the newly found event (lines 5-10). Finally, if the entries in the DBM would be equal, the newly found event is added to the event set (lines 13-15).

Algorithm 5.8: `compareDifferentV`$(Z, i_{\text{new}}, i, E, \mathit{restrictVal}_{\text{new}}, \mathit{restrictVal})$

```
1 let $i = (v \geq k)$;
2 let $i_{\text{new}} = (v_{\text{new}} \geq k_{\text{new}})$;
3 if $-\mathit{restrictVal} > -\mathit{restrictVal}_{\text{new}} + Z(v_{\text{new}}, v)$ then
4     return $(E, \mathit{NotPossible})$;
5 else if $-\mathit{restrictVal}_{\text{new}} > -\mathit{restrictVal}_j + Z(v, v_{\text{new}})$ then
6     $E := E - \{i\}$;
7     return $(E, \mathit{Undecided})$;
8 else if $\mathit{restrictVal}_j > \texttt{nlb}(v_{\text{new}}) + Z(v, v_{\text{new}})$ then
9     $E := E - \{i\}$;
10    return $(E, \mathit{Undecided})$;
11 else if $\mathit{restrictVal}_{\text{new}} > \texttt{nlb}(v) + Z(v_{\text{new}}, v)$ then
12    return $(E, \mathit{NotPossible})$;
13 else
14    $E := E \cup \{i_{\text{new}}\}$;
15    return $(E, \mathit{Possible})$;
```

The `happensFirstIT` function, shown in Algorithm 5.9, determines how the DBM would be restricted for a change upon the corresponding inequality for the newly found event, i_{new}, and the transition for the event under inspection, t (lines 2-3). If the value for the newly found event is more restrictive than the value for the event under inspection, the event under inspection is removed from the event set but nothing can be said about the newly found event (lines 4-6). If the newly found event is less restrictive then it is found to be not possible and is not added (lines 7-8).

Algorithm 5.9: `happensFirstIT`$(D, R, Z, i_{\text{new}}, t, E)$

```
1 let $i_{\text{new}} = (v_{\text{new}} \geq k_{\text{new}})$;
2 $\mathit{restrictVal} := d_l(t)$;
3 $\mathit{restrictVal}_{\text{new}} :=$ cdiv($k_{\text{new}}$,$R(v_{\text{new}})$);
4 if $-\mathit{restrictVal}_{\text{new}} > -\mathit{restrictVal} + Z(t, v_{\text{new}})$ then
5     $E := E - \{t\}$;
6     return $(E, \mathit{Undecided})$;
7 else if $\mathit{restrictVal}_{\text{new}} >$ ub$(Z, t) + Z(v_{\text{new}}, t)$ then
8     return $(E, \mathit{NotPossible})$;
```

The `happensFirstTI` algorithm, shown in Algorithm 5.10, is very similar to the `happensFirstIT` function where the result is either that the newly found event is not possible or the event under inspection is removed from the set and nothing is decided about the newly found event.

Algorithm 5.10: `happensFirstTI`$(D, R, Z, t_{\text{new}}, i, E)$

```
1 let $i = (v \geq k)$;
2 $\mathit{restrictVal} :=$ cdiv($k$,$R(v)$);
3 $\mathit{restrictVal}_{\text{new}} := d_l(t_{\text{new}})$;
4 if $-\mathit{restrictVal} > -\mathit{restrictVal}_{\text{new}} + Z(t_{\text{new}}, v)$ then
5     return $(E, \mathit{NotPossible})$;
6 else if $\mathit{restrictVal} > \mathit{restrictVal}_{\text{new}} + Z(v, t_{\text{new}})$ then
7     $E := E - \{i\}$;
8     return $(E, \mathit{Undecided})$;
```

The `ineqCanChange` function (called at line 6 of Algorithm 5.4) in Algorithm 5.11 is used to determine if a continuous variable can cross an inequality and result in an inequality change event. Since the inequalities in LHPNs are restricted to greater than or equal to, there are only two cases that need to be checked. In the first case, the rate of change for the variable in the inequality is decreasing and the inequality is currently **true** (line 2). This means that the value of the variable is greater than or equal to the inequality's constant but may decrease below that constant value and result in an inequality change. To test for this case, the upper bound for the variable is checked to see if it can reach or has gone below the value of the inequality's constant in warped space (line 3). If the variable has decreased below the inequality's constant, then the inequality can change value, so **true** is returned (line 4). In the second case, the rate of change for the variable is increasing and the inequality is **false** (line 2). This means that the value of the variable is below the inequality's constant but may increase above the inequality's constant and result in a inequality change. To test for this case, the upper bound of the variable is checked to see if it can reach or has gone above the value of the inequality's constant in the warped space (line 3). If this case is true, then the inequality can change value, so **true** is returned (line 4). The value **false** is returned in all other cases (line 5).

Algorithm 5.11: `ineqCanChange`(R, I, Z, i)

```
1 let $i = (v \geq k)$;
2 if $(R(v) < 0 \wedge I(i)) \vee (R(v) > 0 \wedge \neg I(i))$ then
3     if ub$(Z, v) \geq$ fdiv$(k, R(v))$ then
4         return True;
5 return False;
```

5.2.3 Updating the State

Algorithm 5.12 updates the state set, ψ, as a result of an event set, E. The algorithm begins by calling `restrict`, shown in Algorithm 5.13 (line 2). Next, `recanonicalize`, shown in Algorithm 4.1, applies Floyd's all-pairs shortest path algorithm to restore Z to its canonical form (line 3). When an inequality changes, the next step is to update the value of I by complementing the value of the inequality in I (lines 5-6). When a transition fires (line 7-8), the state update required is more involved as shown in `fireTransition` which is described below in Algorithm 5.14 described below. Next, the transitions are checked to see if any have become newly enabled or disabled (lines 9-13). A clock for a transition t not in Z that is enabled must be added to Z while a clock for a transition t in Z that is not enabled must be removed from Z. Finally, time is advanced using Algorithm 5.15, Z is recanonicalized again, and the new state set is returned (lines 14-16).

The `restrict` function, shown in Algorithm 5.13, modifies Z to reflect that time must have advanced to the point necessary for the events in the event set to have

Algorithm 5.12: `updateState`$(T, V, En, D, BA, VA, RA, \psi, E)$

```
1  let $\psi = (M, S, Q, R, I, Z)$;
2  $Z :=$ restrict$(T, D, Z, E)$;
3  $Z :=$ recanonicalize$(Z)$;
4  forall $e \in E$ do
5      if $e \in ineq(En)$ then
6          $I(e) := \neg I(e)$;
7      else
8          $\psi :=$ fireTransition$(V, VA, RA, BA, \psi, e)$;
9  forall $t \in T$ do
10     if $t \notin Z \wedge t \in \mathcal{EN}(M, S, Q)$ then
11         $Z :=$ addT$(Z, t)$;
12     else if $t \in Z \wedge t \notin \mathcal{EN}(M, S, Q)$ then
13         $Z :=$ rmT$(Z, t)$;
14 $Z :=$ advanceTime$(En, D, R, I, Z)$;
15 $Z :=$ recanonicalize$(Z)$;
16 return $\psi$;
```

occurred (i.e., the clock for the transition firing reaches its lower bound (lines 2-4) or the continuous variable reaches the constant in an inequality (lines 5-9)) as described previously in Section 5.2.2.

Algorithm 5.13: `restrict`(T, D, Z, E)

```
1  forall $e \in E$ do
2      if $e \in T$ then
3          let $t = e$;
4          nlb$(Z, t) := d_l(t)$;
5      else
6          let $e = (v \geq k)$;
7          nlb$(Z, v) :=$ cdiv$(k, R(v))$;
8          if ub$(Z, v) <$ cdiv$(k, R(v))$ then
9              ub$(Z, v) :=$ cdiv$(k, R(v))$;
10 return Z;
```

5.2.4 Firing a Transition

The `updateState` function calls `fireTransition`, shown in Algorithm 5.14, to fire a transition t in state set ψ. This algorithm first updates the marking by removing the tokens from all places in $\bullet t$ and adding tokens to all places in $t\bullet$ (line 2). Next, the transition clock for t must be removed from Z (line 3). Then, all assignments labeled on t must be performed. This includes variable assignments, rate assignments, and Boolean signal assignments using the algorithms described in Section 4.5 (lines 4-6). The rate assignments may have activated or deactivated a continuous variable, so all continuous variables are checked and added or removed from Z as necessary (lines 7-11). Finally, Z is warped using Algorithm 4.4 to properly account for any rate changes that may have occurred (line 12).

Algorithm 5.14: `fireTransition`(V, VA, RA, BA, ψ, t)

```
1  let $\psi = (M, S, Q, R, I, Z)$;
2  $M := (M - \bullet t) \cup t\bullet$;
3  $Z :=$ rmT($Z$,$t$);
4  $(Q, Z) :=$ doVarAsgn($Q, R, Z, VA, t$);
5  forall $(v := r) \in RA(t)$ do $R'(v) := r$;
6  forall $(b := ba) \in BA(t)$ do $S'(b) := ba$;
7  forall $v \in V$ do
8      if $v \notin Z \wedge R'(v) \neq 0$ then
9          $(Q, Z) :=$ addV($Q, R, Z, v$);
10     else if $v \in Z \wedge R'(v) = 0$ then
11         $(Q, Z) :=$ rmV($Q, R, Z, v$);
12 $(R, Z) :=$ dbmWarp($R, R', Z$);
13 return $\psi$;
```

5.2.5 Advancing Time

The `updateState` function calls `advanceTime` shown in Algorithm 5.15 to advance time in Z. The idea behind `advanceTime` is that time should be allowed to advance as far as possible without missing an event. To ensure that the firing of an enabled transition t is not missed, `advanceTime` sets the upper bound value for the clock associated with t to the upper delay bound for t (lines 1-2). To ensure that a change in inequality value is not missed on a variable v, all inequalities involving variable v are checked by the function `checkIneq`, and the largest amount of time that can advance before one of these inequalities changes value is assigned to the upper bound value for v, (lines 3-4).

The `checkIneq` function, shown in Algorithm 5.16, determines the minimum amount of time that can pass before a inequality can change. This is done for each variable individually. The process checks every inequality involving the current variable (line 2).

Algorithm 5.15: `advanceTime`(En, D, R, I, Z)

```
1  forall $t \in Z$ do
2      ub$(Z, t) := d_{\mathrm{u}}(t)$;
3  forall $v \in Z$ do
4      ub$(Z, v) :=$ checkIneq$(En, R, I, Z, v)$;
5  return $Z$;
```

Initially, the minimum value is set to ∞ or unbounded (line 1). For this reason, the cases that result in an unbounded assignment are not shown (e.g., if the variable is increasing and the upper bound of the DBM is greater than the inequality's constant value in the warped space). If the rate of change for the variable is increasing and the inequality is not true, the algorithm determines how much time must pass until the inequality can become true (lines 3-8). In the first case, if the upper bound of the DBM is less than the inequality's value in the warped space then the minimum value is the amount of time required to reach the inequality's constant value at the current rate of change (lines 5-6). The final case is when the value is equal to the inequality's constant. In this case the current value of the DBM is reported as the minimum value (lines 7-8). A similar set of cases are used when the variable's rate of change is less than zero and the inequality is true (lines 9-14). Once the minimum amount of time that can be advanced before a change of inequality for the given variable is calculated, it is returned (line 15).

Algorithm 5.16: `checkIneq`(En, R, I, Z, v)

```
1  min := $\infty$;
2  forall $(v_i \geq k_i) \in ineq(En).v_i = v$ do
3      if $R(v) > 0$ then
4          if $\neg I((v_i \geq k_i))$ then
5              if ub$(Z, v) <$ fdiv$(k_i, R(v))$ then
6                  min := min(min,fdiv$(k_i, R(v))$);
7              else if nlb$(Z, p) \leq$ fdiv$(k_i, R(v))$ then
8                  min := min(min,ub$(Z, v)$);
9      else
10         if $I((v_i \geq k_i))$ then
11             if ub$(Z, p) \leq$ fdiv$(k_i(i), R(v))$ then
12                 min := min(min,fdiv$(k_i(i), R(v))$);
13             else if nlb$(Z, v) <$ fdiv$(k_i(i), R(v))$ then
14                 min := min(min,ub$(Z, v)$);
15 return min;
```

5.2.6 Error Trace Generation

When a failure is found during state space exploration, the `error` function, shown in Algorithm 5.17, is called. This algorithm creates a state set-based error trace Π demonstrating the error condition (line 1). A state set-based error trace Π is an ordered set of state sets ψ. From this state set-based trace, two state-based error traces (Λ_{max} and Λ_{min}) are created showing the error. A state-based error trace Λ is an ordered set of pairs (λ, τ) where λ is the state at time τ. These state-based traces can be used to generate waveforms representing the error which designers can use to understand the failure and correct the design or model. If the error trace is a false negative, an error created due to the conservative exploration of the reachability algorithm, then the model can potentially be refined to avoid the addition of the conservative state space. If the error trace is an actual error the circuit should be corrected and a new model should be generated from the corrected circuit.

Algorithm 5.17: `error`$(\psi_0, \psi_{\text{fail}}, \Psi, \Gamma, V, T)$

```
1 $\Pi$ := createErrorTrace$(\psi_0, \psi_{\text{fail}}, \Psi, \Gamma)$;
2 $\Lambda_{\text{max}}$ := extractMaxTrace$(\Pi, V, T)$;
3 $\Lambda_{\text{min}}$ := extractMinTrace$(\Pi, V, T)$;
4 return $(\Lambda_{\text{max}}, \Lambda_{\text{min}})$;
```

The `createErrorTrace` function, shown in Algorithm 5.18, finds the shortest path from the initial state set, ψ_0, to the fail state set, ψ_{fail}, given the set of state sets, Ψ, and the set of edges between the state sets, Γ. The `createErrorTrace` algorithm begins by doing a breadth first search of the entire state space while labeling each state set with the number of state transitions it is away from the initial state set (lines 1-12). The labeling happens by accessing and changing the level of a state set using the `lvl` function. Once the state space has been labeled, the algorithm begins to create a trace, Π starting from the fail state set ψ_{fail} (line 13). The shortest trace is found by searching for the state set with the smallest level leading into the current state set (lines 15-20). When the state with the minimum level is found it is pushed onto the trace stack and used as the previous state in the next iteration (lines 20-21). When the initial state is found, the algorithm returns the trace (line 22).

One difficulty in generating a state-based error trace is determining how much time passes between states in the trace. The time between two state sets is calculated based on

Algorithm 5.18: `createErrorTrace`$(\psi_0, \psi_{\text{fail}}, \Psi, \Gamma)$

```
$lvl := 0$;
lvl$(\psi_0) := lvl$;
$\psi_j := \psi_0$;
while $\psi_j \neq \psi_{fail}$ do
    $lvl := lvl + 1$;
    forall $\psi_i \in \Psi$ do
        if lvl$(\psi_i) = lvl - 1$ then
            forall $(\psi_i, \psi_j) \in \Gamma$ do
                if $\psi_j = undef$ then
                    lvl$(\psi_j) := lvl$;
                    if $\psi_j = \psi_{fail}$ then
                        break;
$\Pi := \psi_{\text{fail}}$;
$\psi_{\text{prev}} := \psi_{\text{fail}}$;
while $lvl > 0$ do
    $lvl := lvl - 1$;
    forall $\psi_i \in \Psi$ do
        forall $(\psi_i, \psi_{\text{prev}}) \in \Gamma$ do
            if lvl$(\psi_i) = lvl$ then
                $\psi_{\text{prev}} := \psi_i$;
    push$(\psi_{\text{prev}}, \Pi)$;
return $\Pi$;
```

the change in value of a continuous variable and its rate of change or a persistent transition clock. As a result of this dependency, an error trace cannot be generated for LHPN models where there are no continuous variables in the system, where all continuous variables in the system have a rate of zero, or there is not a transition clock persistent between two state sets. Two state-based error traces are generated from the state set-based error trace. One is the minimum path and the other is the maximum path. The actual property violation is shown in one of the traces. Algorithm 5.19 shows how `LEMA` extracts the maximum error trace given a state set-based failure trace, Π. Each state set ψ in the failure trace Π is examined for an active variable (lines 5-6). An active variable is defined as a continuous variable with a non-zero rate or a transition that is in both state sets. If an active variable is not found, a failure results (lines 24-25). If an active variable is found, the time elapsed between the current state and the next state is calculated using the values in both state sets and the rate of the current state set. The time elapsed between the two state sets is added to the total or global time, τ (line 8). The state λ is extracted from the state set ψ_i. The extraction is trivial for $M_\lambda, S_\lambda, R_\lambda$, and I_λ (lines 9-10 and 16-17). The extraction of Q_λ is done by extracting the value from the upper bound of Z_ψ if present or from Q_ψ if not present (lines 11-15). A similar process is used to extract C_λ although if the transition is not found in Z_ψ no information is known, so the value of C_λ is set to ∞ for that transition (lines 18-22). This process continues until all state sets in the trace Π have been analyzed. This process is similar for extracting minimal traces except the lower bounds are extracted from Z_ψ and Q_ψ.

Algorithm 5.19: `extractMaxTrace`(Π, V, T)

```
let$\psi = (M_\psi, S_\psi, Q_\psi, R_\psi, I_\psi, Z_\psi)$;
let$\lambda = (M_\lambda, S_\lambda, Q_\lambda, R_\lambda, RR_\lambda, I_\lambda, C_\lambda)$;
$\tau := 0$;
$\Lambda := \emptyset$;
forall $i \in \texttt{len}(\Pi) - 1$ do
    $v :=$ findActiveVar$(\psi_i, \psi_{i-1})$;
    if $v$ then
        $\tau := \tau + \frac{\left|\texttt{ub}(Z_{\psi_{i-1}}, v) * R_{\psi_{i+1}}(v) - \texttt{ub}(Z_{\psi_i}, v) * R_{\psi_i}(v)\right|}{R_{\psi_i}(v)}$;
        $M_\lambda := M_{\psi_i}$;
        $S_\lambda := S_{\psi_i}$;
        forall $v_j \in V$ do
            if $v_j \in Z_{\psi_i}$ then
                $Q_\lambda(v_j) := \texttt{ub}(Z_{\psi_i}, v_j) * R_{\psi_i}(v_j)$;
            else
                $Q_\lambda(v_j) := q_{u_{\psi_i}}(v_j)$;
        $R_\lambda := R_{\psi_i}$;
        $I_\lambda := I_{\psi_i}$;
        forall $t \in T$ do
            if $t \in Z_{\psi_i}$ then
                $C_\lambda(t) := \texttt{ub}(Z_{\psi_i}, t)$;
            else
                $C_\lambda(t) := \infty$;
        $\Lambda := \Lambda \cup (\tau, \lambda)$;
    else
        return $\emptyset$;
return $\Lambda$;
```

5.3 Reachability Analysis Example

Figure 5.1 is the LHPN for the switched capacitor integrator circuit generated by the VHDL-AMS compiler and expanded using the LHPN constant rate transformation. This section uses this figure to illustrate the DBM reachability algorithm from Section 5.2. In the initial state shown in Equation 5.2, V_{out} is between -1008 mV and -990 mV ($-56 \leq V_{\text{out}} \leq -55$ with a warp of 18); $\dot{V}_{\text{out}}$ is 18 mV/μs; $b_{Vout[18,22]}$ is **true**; and V_{in}, $b_{Vout[-22,-18]}$, *r0*, and *r1* are **false**. Initially the only enabled transition is t_2. The clock for t_2, c_{t_2}, is initialized to zero. In the first iteration of the algorithm, time is allowed to advance up to but not cross the point where t_2 must fire, (i.e., 100 μs) as shown in Equation 5.3. If 100 μs have passed at a rate of 18 mV/μs, then V_{out} may be as high as 800 mV. In this state, the DBM represents a range of values for c_{t_2} between 0 μs and 100 μs, and V_{out} is between -1008 mV and 810 mV. In this state, t_2 is the only enabled action and is therefore selected. When t_2 fires, the state space exploration engine determines that based on the delay bound of $[100, 100]$ for t_2 exactly 100 μs have passed. This fact restricts the range of values for V_{out} to between 810 mV and 792 mV as shown in Equation 5.4. This range represents the possibility that V_{out} has been increasing at a rate of 18 mV/μs for 100 μs. Equation 5.4 shows the state of the LHPN after firing t_2 where t_1 and t_3 are enabled. t_1 is enabled because V_{in} is set to **true** using a Boolean signal assignment on t_2. Transition t_3 is enabled because the firing of t_2 marks the place in its preset. At this point in the state space exploration, the first negative half cycle of V_{in} is complete.

The positive half cycle of V_{in} begins with the state in Equation 5.4. From this state, it is possible to fire transition t_1 or t_3. The `select` function chooses to fire t_1 which when fired sets the rate of V_{out} to -22 mV/μs, $b_{Vout[18,22]}$ to **false**, $b_{Vout[-22,-18]}$ to **true**, and *r1* to **true**. This state is shown in Equation 5.5. Since t_1 is no longer in Z to hold back time, time is allowed to advance to the point where 100 μs can pass. This allows the range of V_{out} to be between -1408 mV and 814 mV. In this state, the possible actions are to fire transition t_3 or t_6. The firing of transition t_6 is selected. The firing of t_6 changes the rate of V_{out} to -18 mV/μs and sets *r1* to **false** as shown in Equation 5.6. After the warp, the value of V_{out} is between -1422 mV and 828 mV. In this state, the only possible action is to fire transition t_3. Firing this transition restricts the values in the DBM to the point where c_{t_3} is 100 μs. Equation 5.7 shows that this restriction changes the value of V_{out} to be between 972 mV and 1422 mV. The firing of t_3 also sets V_{in} to **false**. This ends the first cycle of V_{in}.

Equation 5.8 represents the state set several cycles later. In this state, the value of V_{out} is between -1980 mV and 1650 mV. In this state, there are three possible actions: firing

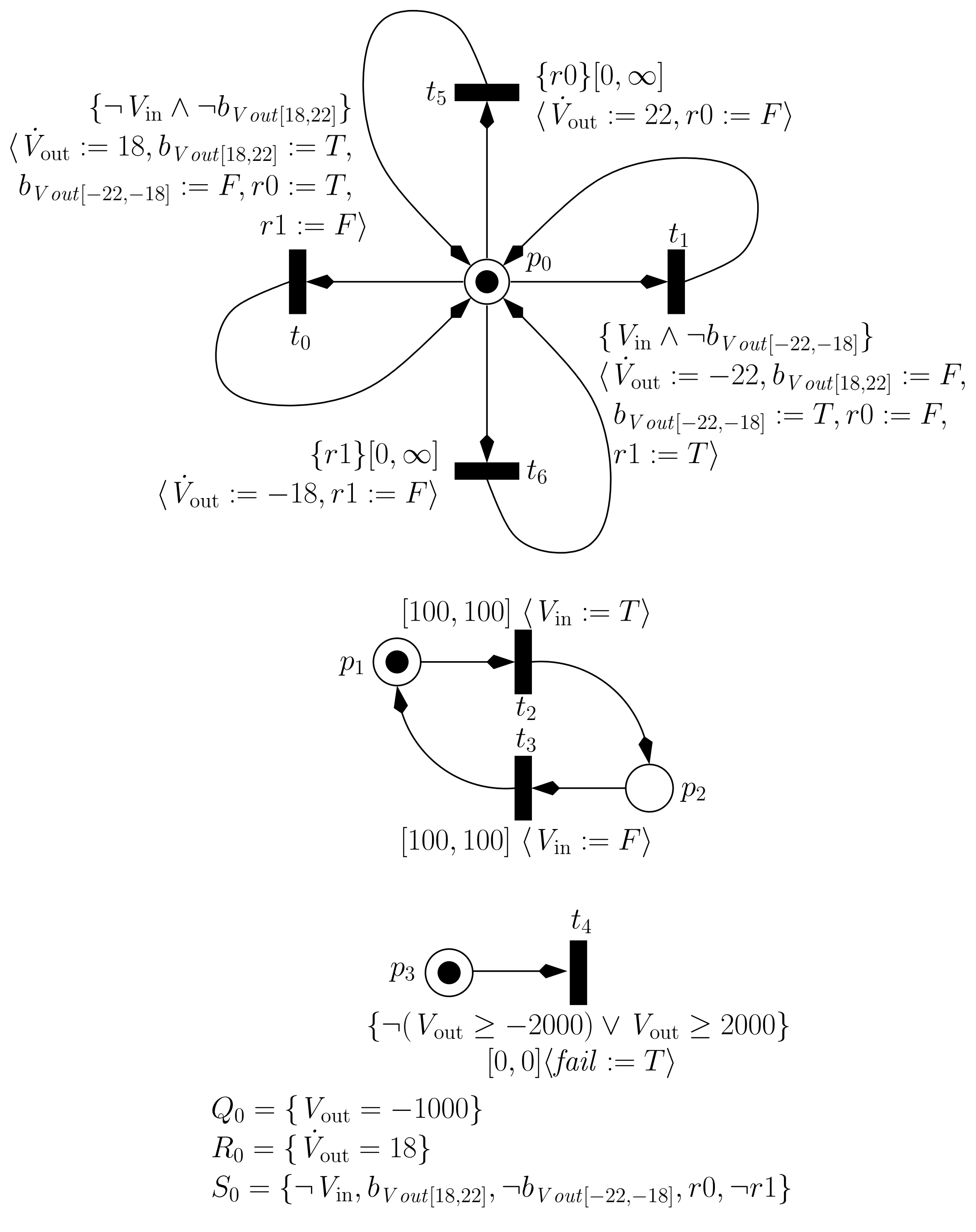

$\{r0\}[0,\infty]$
$\langle \dot{V}_{\text{out}} := 22, r0 := F\rangle$
t_5
$\{\neg V_{\text{in}} \wedge \neg b_{Vout[18,22]}\}$
$\langle \dot{V}_{\text{out}} := 18, b_{Vout[18,22]} := T,$
$b_{Vout[-22,-18]} := F, r0 := T,$
$r1 := F\rangle$
t_1
p_0
t_0
$\{V_{\text{in}} \wedge \neg b_{Vout[-22,-18]}\}$
$\langle \dot{V}_{\text{out}} := -22, b_{Vout[18,22]} := F,$
$b_{Vout[-22,-18]} := T, r0 := F,$
$r1 := T\rangle$
$\{r1\}[0,\infty]$
$\langle \dot{V}_{\text{out}} := -18, r1 := F\rangle$
t_6
$[100,100]\ \langle V_{\text{in}} := T\rangle$
p_1
t_2
t_3
p_2
$[100,100]\ \langle V_{\text{in}} := F\rangle$
t_4
p_3
$\{\neg(V_{\text{out}} \geq -2000) \vee V_{\text{out}} \geq 2000\}$
$[0,0]\langle fail := T\rangle$
$Q_0 = \{V_{\text{out}} = -1000\}$
$R_0 = \{\dot{V}_{\text{out}} = 18\}$
$S_0 = \{\neg V_{\text{in}}, b_{Vout[18,22]}, \neg b_{Vout[-22,-18]}, r0, \neg r1\}$

Figure 5.1. An LHPN demonstrating the DBM-based reachability algorithm.

transition t_3, firing transition t_6, or changing inequality $V_{\text{out}} \geq -2000$. The inequality change action is selected and changed resulting in the state shown in Equation 5.9. The change of inequality has now enabled transition t_4. The restrict step to acknowledge the arrival of V_{out} at the constant value of -2000 mV results in a value of V_{out} between -2002 mV between -1980 mV. The firing of t_4 is selected and results in the analyzer detecting a firing of a transition in the fail set.

The method used by `extractMaxTrace` and `extractMinTrace` is illustrated using the state sets shown in Equations 5.2-5.5 as the state set-based trace Π. Equation 5.10 shows the initial state which corresponds to the state set in Equation 5.2. The state in Equation 5.10 contains information for both the max and min trace by using a pair of values for the parts of the state that differ between the traces. These value pairs are enclosed in square brackets with the min trace values listed first. For example, τ may vary between traces and is listed first in Equation 5.10 as [0,0]. Initially the value of V_{out} is -990 mV for the max trace as shown in Equation 5.10. In the next state shown in Equation 5.11, this value increases to 810 mV which is a change of 1800 mV at a rate of 18 mV/μs. Using the equation presented in line 8 of Algorithm 5.19, the value of τ is calculated to be 100 μs. In the next state shown in Equation 5.12, the values for V_{out} in the max trace do not change. The clock c_{t_2} has fired after 100 μs, so the minimum value has changed as expected by 100 μs and its state reflects this change. In Equation 5.13, the rate of V_{out} has changed from 18 mV/μs to -22 mV/μs. The maximum trace is allowed to move forward by 100 μs while the minimum trace is being held back as reflected by their times and values. The extraction of a state-based error trace proceeds in this manner. When finished the state-based error trace can be used to generate waveforms like the one shown in Figure 5.2 to represent the error in a graphical manner familiar to designers.

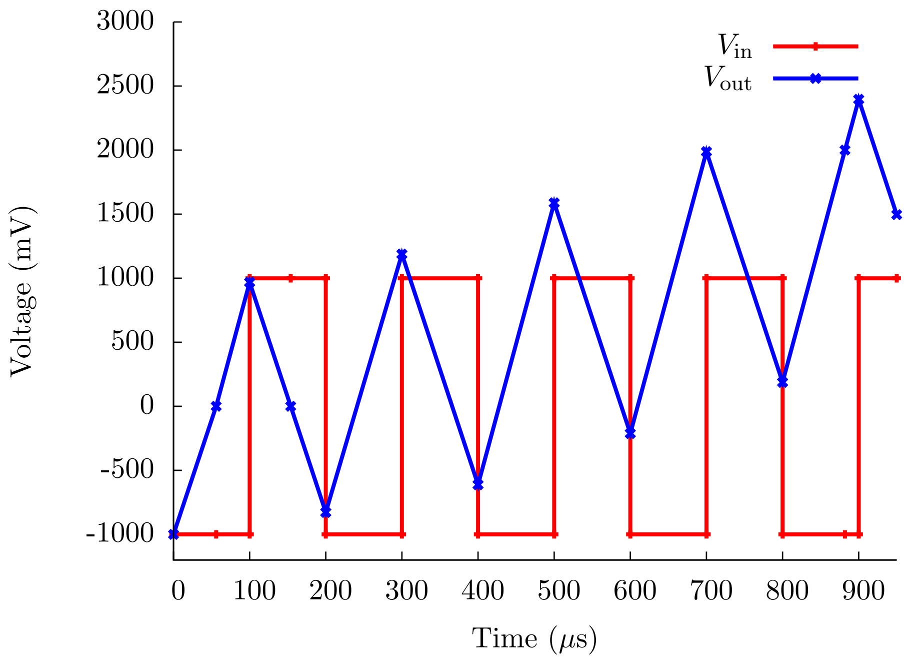


Figure 5.2. A waveform generated from an error trace produced by verification of the switched capacitor integrator circuit.

$$\begin{bmatrix} x_0 & c_{t_2} & V_{\text{out}} \\ 0 & 0 & 56 \\ 0 & 0 & 56 \\ -55 & -55 & 0 \end{bmatrix} \quad \begin{gathered} \dot{V}_{\text{out}} = 18, V_{\text{in}} = F \\ b_{Vout[18,22]} = T, b_{Vout[-22,-18]} = F \\ r0 = F, r1 = F \\ V_{\text{out}} \geq -2000 = T, V_{\text{out}} \geq 2000 = F \end{gathered} \tag{5.2}$$

$$\begin{bmatrix} x_0 & c_{t_2} & V_{\text{out}} \\ 0 & 0 & 56 \\ 100 & 0 & 56 \\ 45 & -55 & 0 \end{bmatrix} \quad \begin{gathered} \dot{V}_{\text{out}} = 18, V_{\text{in}} = F \\ b_{Vout[18,22]} = T, b_{Vout[-22,-18]} = F \\ r0 = F, r1 = F \\ V_{\text{out}} \geq -2000 = T, V_{\text{out}} \geq 2000 = F \end{gathered} \tag{5.3}$$

$$\begin{bmatrix} x_0 & V_{\text{out}} & c_{t_1} & c_{t_3} \\ 0 & -44 & 0 & 0 \\ 45 & 0 & 45 & 45 \\ 0 & -44 & 0 & 0 \\ 0 & -44 & 0 & 0 \end{bmatrix} \quad \begin{gathered} \dot{V}_{\text{out}} = 18, V_{\text{in}} = T \\ b_{Vout[18,22]} = T, b_{Vout[-22,-18]} = F \\ r0 = F, r1 = F \\ V_{\text{out}} \geq -2000 = T, V_{\text{out}} \geq 2000 = F \end{gathered} \tag{5.4}$$

$$\begin{bmatrix} x_0 & V_{\text{out}} & c_{t_3} & c_{t_6} \\ 0 & 37 & 0 & 0 \\ 64 & 0 & -36 & -36 \\ 100 & 37 & 0 & 0 \\ 100 & 37 & 0 & 0 \end{bmatrix} \quad \begin{gathered} \dot{V}_{\text{out}} = -22, V_{\text{in}} = T \\ b_{Vout[18,22]} = F, b_{Vout[-22,-18]} = T \\ r0 = F, r1 = T \\ V_{\text{out}} \geq -2000 = T, V_{\text{out}} \geq 2000 = F \end{gathered} \tag{5.5}$$

$$\begin{bmatrix} x_0 & V_{\text{out}} & c_{t_3} \\ 0 & 46 & 0 \\ 79 & 0 & -21 \\ 100 & 46 & 0 \end{bmatrix} \quad \begin{gathered} \dot{V}_{\text{out}} = -18, V_{\text{in}} = T \\ b_{Vout[18,22]} = F, b_{Vout[-22,-18]} = T \\ r0 = F, r1 = F \\ V_{\text{out}} \geq -2000 = T, V_{\text{out}} \geq 2000 = F \end{gathered} \tag{5.6}$$

$$\begin{bmatrix} x_0 & V_{\text{out}} & c_{t_0} & c_{t_2} \\ 0 & -54 & 0 & 0 \\ 79 & 0 & 79 & 79 \\ 0 & -54 & 0 & 0 \\ 0 & -54 & 0 & 0 \end{bmatrix} \quad \begin{gathered} \dot{V}_{\text{out}} = -18, V_{\text{in}} = F \\ b_{Vout[18,22]} = F, b_{Vout[-22,-18]} = T \\ r0 = F, r1 = F \\ V_{\text{out}} \geq -2000 = T, V_{\text{out}} \geq 2000 = F \end{gathered} \tag{5.7}$$

$$\begin{bmatrix} x_0 & V_{\text{out}} & c_{t_3} & c_{t_6} \\ 0 & 75 & 0 & 0 \\ 90 & 0 & 2 & 2 \\ 100 & 75 & 0 & 0 \\ 100 & 75 & 0 & 0 \end{bmatrix} \quad \begin{gathered} \dot{V}_{\text{out}} = -22, V_{\text{in}} = T \\ b_{Vout[18,22]} = F, b_{Vout[-22,-18]} = T \\ r0 = F, r1 = T \\ V_{\text{out}} \geq -2000 = T, V_{\text{out}} \geq 2000 = F \end{gathered} \tag{5.8}$$

$$\begin{bmatrix} x_0 & V_{\text{out}} & c_{t_3} & c_{t_6} & c_{t_4} \\ 0 & -90 & -88 & -88 & 0 \\ 91 & 0 & 2 & 2 & 91 \\ 100 & 10 & 0 & 0 & 100 \\ 100 & 10 & 0 & 0 & 100 \\ 0 & -90 & -88 & -88 & 0 \end{bmatrix} \quad \begin{gathered} \dot{V}_{\text{out}} = -22, V_{\text{in}} = T \\ b_{Vout[18,22]} = F, b_{Vout[-22,-18]} = T \\ r0 = F, r1 = T \\ V_{\text{out}} \geq -2000 = F, V_{\text{out}} \geq 2000 = F \end{gathered} \tag{5.9}$$

$$\begin{aligned} \tau &= [0,0] \\ M_\lambda &= \{p_0, p_1, p_3\} \\ S_\lambda &= \{\neg V_{\text{in}}, b_{Vout[18,22]}, \neg b_{Vout[-22,-18]}, \neg r0, \neg r1\} \\ Q_\lambda &= \{V_{\text{out}} = [-1008, -990] \\ R_\lambda &= \{\dot{V}_{\text{out}} = 18\} \\ I_\lambda &= \{V_{\text{out}} \geq -2000, \neg V_{\text{out}} \geq 2000\} \\ C_\lambda &= \{c_{t_0} = \infty, c_{t_1} = \infty, c_{t_2} = [0,0], c_{t_3} = \infty, c_{t_4} = \infty, c_{t_5} = \infty, c_{t_6} = \infty\} \end{aligned} \tag{5.10}$$

$$\begin{aligned} \tau &= [0,100] \\ M_\lambda &= \{p_0, p_1, p_3\} \\ S_\lambda &= \{\neg V_{\text{in}}, b_{Vout[18,22]}, \neg b_{Vout[-22,-18]}, \neg r0, \neg r1\} \\ Q_\lambda &= \{V_{\text{out}} = [-1008, 810] \\ R_\lambda &= \{\dot{V}_{\text{out}} = 18\} \\ I_\lambda &= \{V_{\text{out}} \geq -2000, \neg V_{\text{out}} \geq 2000\} \\ C_\lambda &= \{c_{t_0} = \infty, c_{t_1} = \infty, c_{t_2} = [0,100], c_{t_3} = \infty, c_{t_4} = \infty, c_{t_5} = \infty, c_{t_6} = \infty\} \end{aligned} \tag{5.11}$$

$$\begin{aligned} \tau &= [100,100] \\ M_\lambda &= \{p_0, p_1, p_2\} \\ S_\lambda &= \{V_{\text{in}}, b_{Vout[18,22]}, \neg b_{Vout[-22,-18]}, \neg r0, \neg r1\} \\ Q_\lambda &= \{V_{\text{out}} = [792, 810]\} \\ R_\lambda &= \{\dot{V}_{\text{out}} = 18\} \\ I_\lambda &= \{V_{\text{out}} \geq -2000, \neg V_{\text{out}} \geq 2000\} \\ C_\lambda &= \{c_{t_0} = \infty, c_{t_1} = [0,0], c_{t_2} = \infty, c_{t_3} = [0,0], c_{t_4} = \infty, c_{t_5} = \infty, c_{t_6} = \infty\} \end{aligned} \tag{5.12}$$

$$\begin{aligned} \tau &= [101,200.82] \\ M_\lambda &= \{p_0, p_1, p_2\} \\ S_\lambda &= \{V_{\text{in}}, \neg b_{Vout[18,22]}, b_{Vout[-22,-18]}, \neg r0, r1\} \\ Q_\lambda &= \{V_{\text{out}} = [814, -1408]\} \\ R_\lambda &= \{\dot{V}_{\text{out}} = -22\} \\ I_\lambda &= \{V_{\text{out}} \geq -2000, \neg V_{\text{out}} \geq 2000\} \\ C_\lambda &= \{c_{t_0} = \infty, c_{t_1} = \infty, c_{t_2} = \infty, c_{t_3} = [0,100], c_{t_4} = \infty, c_{t_5} = \infty, c_{t_6} = [0,100]\} \end{aligned} \tag{5.13}$$

CHAPTER 6

RESULTS

Using the SAV methodology described in the previous chapters, `LEMA` can generate abstract HDL models and LHPNs of hybrid systems and AMS circuits. The abstract HDL models are simulated using standard HDL-based simulators. These simulation results are compared against simulation results from the transistor-level models to quantify efficiency and model quality. The LHPNs are used to formally verify safety properties of the associated systems. This chapter presents results for several classic examples in hybrid systems theory, a tunnel diode oscillator, two versions of the switched capacitor integrator, a *phase locked loop* (PLL) phase detector, and a CMOS ring oscillator with feedforward inverters. The results for these examples show the promise of the SAV methodology. All results are run on a 2.16 GHz Intel Core 2 Duo with 2 GB of RAM.

6.1 Hybrid Systems Examples

In [6], the authors present several hybrid system examples that contain different hybrid system dynamics. VHDL-AMS descriptions of three of these examples are compiled into LHPNs and verified using `LEMA`'s DBM-based model checker described in Chapter 5. Table 6.1 shows the results for several hybrid system benchmark examples using `LEMA`'s DBM-based analyzer.

6.1.1 Water Level Monitor

The water level monitor is used to regulate the water level in a tank. The tank is twenty-five meters tall, and the system must maintain the level of the water in the tank such that the tank is never dry and never overflows. The tank contains a pump that fills the tank at a rate of one meter/second. When the pump turns off the tank drains at a rate of two meters/second. Changing between the modes of filling and draining the tank requires between two and four seconds. Figure 6.1 is the VHDL-AMS description for the water level monitor. The water tank and pump are modeled as well as the monitoring

Table 6.1. Hybrid systems benchmark verification results for `LEMA`'s DBM-based verifier.

	`LEMA` DBM-based			
Example	Zones	Time(s)	Verifies?	Correct?
Water level monitor (2x)	10	0.03	Yes	Yes
Water level monitor (1x)	11	0.04	Yes	No
Temperature controller (80)	48	0.05	Yes	Yes
Temperature controller (100)	50	0.06	No	Yes
Billiards game (20,40)	154	0.06	Yes	Yes
Billiards game (20,20)	20	0.05	No	Yes

system. The monitoring system detects when the water level increases above twenty meters and turns off the pump. The monitoring system turns on the pump when the tank level drops below ten meters. This VHDL-AMS description (water level monitor (2x)) compiles into an LHPN with eight places and seven transitions and verifies after finding ten zones in 0.03 seconds.

A false negative result from `LEMA`'s DBM-based model checker is produced by changing the numbers in the water level monitor to approximately half of their value. If the controller waits until 11 meters to begin closing the valve, 5 meters to begin opening the valve, the valve closing time requires 1-2 seconds, and the tank size is 14 meters a false negative failure results. This failure happens because the water level in the tank should be between twelve and thirteen meters when the water starts draining. When the water starts draining, the rate changes to two. When the zone is warped to represent this state, a value of thirteen (i.e., 6.5 in a warped space of two) cannot be represented because of the integer approximation. The value of thirteen is conservatively approximated to be fourteen (i.e., 7 in a warped space of two) which causes the failure.

6.1.2 Temperature Controller

The next example is a temperature controller that controls the temperature of a reactor using two independent control rods. The goal of the system is to maintain the temperature within a safe range. Figure 6.3 is the VHDL-AMS description for the temperature controller. When the temperature t reaches its maximum value, 1100, one of the rods must be inserted to begin cooling the system. If the first rod is inserted,

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.handshake.all;
use work.nondeterminism.all;
entity waterLevel is
end waterLevel;
architecture monitor of waterLevel is
  quantity y:real;
  signal inc:std_logic := '1';
begin
  break y => 2.0;
  if inc = '1' use
    y'dot == 1.0;
  elsif inc = '0' use
    y'dot == 2.0;
  end use;
  process begin
    wait until y'above(20.0);
    assign(inc,'0',2,4);
    wait until not y'above(10.0);
    assign(inc,'1',2,4);
  end process;
  assert (y'above(1.0) and not y'above(25.0))
    report "Error:  Overfill/underfill of the tank."
    severity failure;
end monitor;
```

Figure 6.1. VHDL-AMS code for the water level monitor.

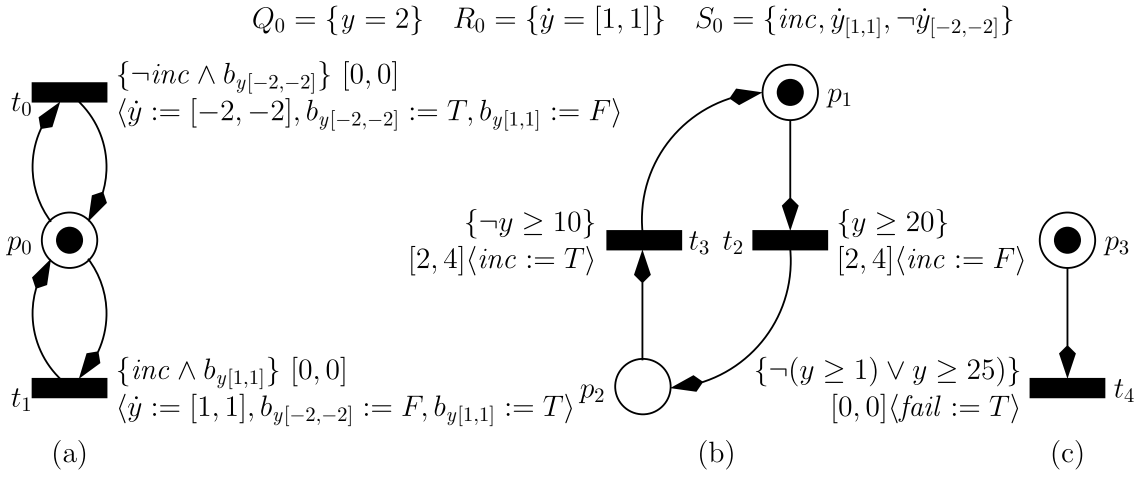

$Q_0 = \{y = 2\}$ $R_0 = \{\dot{y} = [1,1]\}$ $S_0 = \{inc, \dot{y}_{[1,1]}, \neg\dot{y}_{[-2,-2]}\}$
t_0
$\{\neg inc \wedge b_{y[-2,-2]}\}$ $[0,0]$
$\langle \dot{y} := [-2,-2], b_{y[-2,-2]} := T, b_{y[1,1]} := F\rangle$
p_0
t_1
$\{inc \wedge b_{y[1,1]}\}$ $[0,0]$
$\langle \dot{y} := [1,1], b_{y[-2,-2]} := F, b_{y[1,1]} := T\rangle$
(a)
p_1
$\{\neg y \geq 10\}$
$[2,4]\langle inc := T\rangle$
t_3
t_2
$\{y \geq 20\}$
$[2,4]\langle inc := F\rangle$
p_2
(b)
p_3
$\{\neg(y \geq 1) \vee y \geq 25)\}$
$[0,0]\langle fail := T\rangle$
t_4
(c)

Figure 6.2. An LHPN model of the water level monitor compiled from the VHDL-AMS code in Figure 6.1.

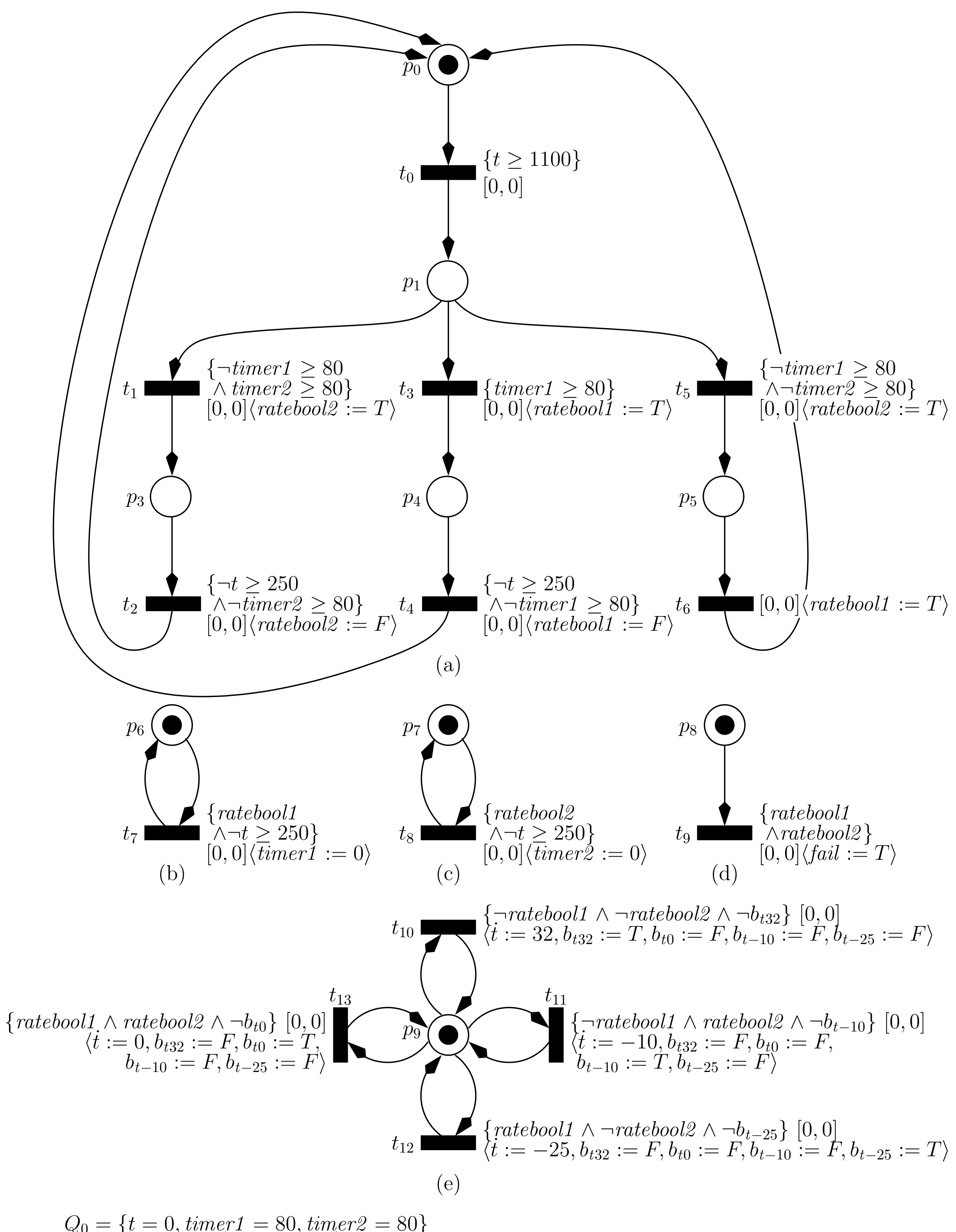
```
library IEEE;
use IEEE.std_logic_1164.all;
use work.handshake.all;
use work.nondeterminism.all;
entity temperature is
end temperature;
architecture monitor of temperature is
  quantity t, timer1, timer2:real;
  signal x1, x2:std_logic := '1';
  signal ratebool1, ratebool2:std_logic := '0';
begin
  break t => 0.0, timer1 => 80.0, timer2 => 80.0;
  break timer1 => 0.0 when ratebool1 = '1' and not t'above(250.0);
  break timer2 => 0.0 when ratebool2 = '1' and not t'above(250.0);
  timer1'dot == 1.0;
  timer2'dot == 1.0;
  if ratebool1 = '0' use
    if ratebool2 = '0' use
      t'dot == 32.0;
    else
      t'dot == -10.0;
    end use;
  else
    if ratebool2 = '0' use
      t'dot == -25.0;
    else
      t'dot == 0.0;
    end use;
  end use;
  rods:process begin
    wait until t'above(1100.0);
    if timer1'above(80.0) then
      assign(ratebool1,'1',0,0);
      wait until not t'above(250.0) and not timer1'above(80.0);
      assign(ratebool1,'0',0,0);
    elsif timer2'above(80.0) then
      assign(ratebool2,'1',0,0);
      wait until not t'above(250.0) and not timer2'above(80.0);
      assign(ratebool2,'0',0,0);
    else
      assign(ratebool2,'1',0,0);
      assign(ratebool1,'1',0,0);
  end process rods;
  assert ratebool1='0' or ratebool2='0'
    report "Error:  Reactor must shut down."
    severity failure;
end monitor;
```

Figure 6.3. VHDL-AMS code for the temperature controller.

the temperature decreases by 10 degrees/minute. If the second rod is inserted, the temperature decreases by 25 degrees/minute. Once a rod is inserted, it remains inserted until the temperature drops below 250 degrees. At this time, the rod is removed. After a rod is removed, it cannot be used again for 80 minutes. The assertion verifies that the reactor does not enter the shutdown condition due to the temperature exceeding 1100 degrees when a control rod is not available for cooling the reactor. The assert statement does this by monitoring the values of the `ratebool` variables. When a `ratebool` variable is `1` it indicates that the corresponding control rod is unavailable for use in cooling. Therefore, the assert statement looks for a condition when both `ratebool` variables are set to `1` indicating that neither rod is available to cool the reactor, if needed. This condition results in a failure. The VHDL-AMS description compiles into an LHPN with 18 places and 19 transitions shown in Figure 6.4 and verifies after finding 48 zones in 0.05 seconds. If the time that the rod must remain unused is increased to 100, the temperature controller fails verification after finding 50 zones in 0.06 seconds.

6.1.3 Billiards Game

The billiards game example consists of a billiards table 320 centimeters long by 120 centimeters wide. There are two balls, white and gray, with a 2 centimeter radius placed on the table at (60,300) and (20,40), respectively. The gray ball is put into play and can bounce off the walls of the table. The goal of the system is to show that the gray ball does not strike the white ball as shown in Figure 6.5. Figure 6.6 is the VHDL-AMS code for this example. In this code, the assert statement is used to notify the user when the gray ball strikes the white ball. This is done by asserting that the asserting that the gray ball does not enter the region of the table where the white ball resides. Since the white ball resides at (60,300), if the gray ball enters the range of 58 to 62 in the x direction while it is also in the range of 298 to 302 in the y direction a failure results. The VHDL-AMS code is compiled into an LHPN with 15 places and 13 transitions shown in Figure 6.7. The example shows that the gray ball does not strike the white ball after exploring 154 zones in 0.06 seconds. If the position of the gray ball is changed to (20,20) the verification finds that the gray ball now strikes the white ball after exploring 20 zones in 0.05 seconds.


p_0
t_0 $\{t \geq 1100\}$ $[0,0]$
p_1
t_1 $\{\neg timer1 \geq 80 \wedge timer2 \geq 80\}$ $[0,0]\langle ratebool2 := T\rangle$
t_3 $\{timer1 \geq 80\}$ $[0,0]\langle ratebool1 := T\rangle$
t_5 $\{\neg timer1 \geq 80 \wedge \neg timer2 \geq 80\}$ $[0,0]\langle ratebool2 := T\rangle$
p_3
p_4
p_5
t_2 $\{\neg t \geq 250 \wedge \neg timer2 \geq 80\}$ $[0,0]\langle ratebool2 := F\rangle$
t_4 $\{\neg t \geq 250 \wedge \neg timer1 \geq 80\}$ $[0,0]\langle ratebool1 := F\rangle$
t_6 $[0,0]\langle ratebool1 := T\rangle$
(a)
p_6
t_7 $\{ratebool1 \wedge \neg t \geq 250\}$ $[0,0]\langle timer1 := 0\rangle$
(b)
p_7
t_8 $\{ratebool2 \wedge \neg t \geq 250\}$ $[0,0]\langle timer2 := 0\rangle$
(c)
p_8
t_9 $\{ratebool1 \wedge ratebool2\}$ $[0,0]\langle fail := T\rangle$
(d)
t_{10} $\{\neg ratebool1 \wedge \neg ratebool2 \wedge \neg b_{t32}\}$ $[0,0]$ $\langle t := 32, b_{t32} := T, b_{t0} := F, b_{t-10} := F, b_{t-25} := F\rangle$
t_{13} $\{ratebool1 \wedge ratebool2 \wedge \neg b_{t0}\}$ $[0,0]$ $\langle t := 0, b_{t32} := F, b_{t0} := T, b_{t-10} := F, b_{t-25} := F\rangle$
p_9
t_{11} $\{\neg ratebool1 \wedge ratebool2 \wedge \neg b_{t-10}\}$ $[0,0]$ $\langle t := -10, b_{t32} := F, b_{t0} := F, b_{t-10} := T, b_{t-25} := F\rangle$
t_{12} $\{ratebool1 \wedge \neg ratebool2 \wedge \neg b_{t-25}\}$ $[0,0]$ $\langle t := -25, b_{t32} := F, b_{t0} := F, b_{t-10} := F, b_{t-25} := T\rangle$
(e)

$Q_0 = \{t = 0, timer1 = 80, timer2 = 80\}$
$R_0 = \{t = 32, timer1 = 1, timer2 = 1\}$
$S_0 = \{\neg ratebool1, \neg ratebool2, b_{t32}, \neg b_{t0}, \neg b_{t-10}, \neg b_{t-25}\}$

Figure 6.4. An LHPN model of the temperature controller compiled from the VHDL-AMS description in Figure 6.3.

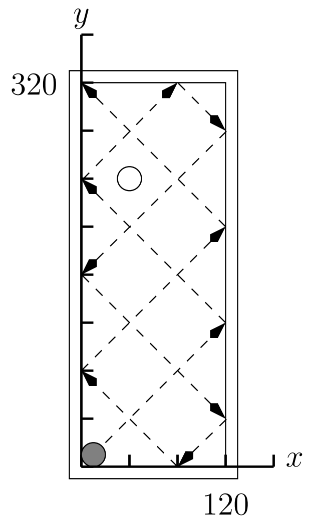


Figure 6.5. A possible trajectory for the gray ball in the billiards game example.

6.2 Tunnel Diode Oscillator

Figure 6.8 is a schematic diagram of a tunnel diode oscillator presented in [75]. The numerical parameters used for this example are from [71]. The verification goal is to ensure that I_l oscillates for specific circuit parameters and initial conditions. This circuit can be described with two differential equations:

$$\frac{dV_\text{c}}{dt} = \frac{1}{C}(-h(V_\text{c}) + I_\text{l})$$

$$\frac{dIl}{dt} = \frac{1}{L}(-V_\text{c} - R \cdot I_\text{l} + V_\text{in})$$

where h is a piecewise model of the tunnel diode behavior:

$$h(V_\text{d}) = \begin{cases} 6.0105V_\text{d}^3 - 0.9917V_\text{d}^2 + 0.0545V_\text{d} \\ \quad \text{where } 0 \leq V_\text{d} \leq 0.055 \\ 0.0692V_\text{d}^3 - 0.0421V_\text{d}^2 + 0.004V_\text{d} + 8.9579 \cdot 10^{-4} \\ \quad \text{where } 0.055 \leq V_\text{d} \leq 0.35 \\ 0.2634V_\text{d}^3 - 0.2765V_\text{d}^2 + 0.0968V_\text{d} - 0.0112 \\ \quad \text{where } 0.35 \leq V_\text{d} \leq 0.50 \end{cases}$$

The system is analyzed using a discretized model of the differential equations. Continuous variables in LHPNs can only change at ranges of rates. Therefore, to analyze more complicated systems, the continuous operating ranges must be decomposed into regions

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.handshake.all;
use work.nondeterminism.all;
entity billiards is
end billiards;
architecture balls of billiards is
  quantity x, y:real;
  signal xInc, yInc:std_logic := '1';
begin
  break x => 20.0;
  break y => 40.0;
  if xInc = '1' use
    x'dot == 2.0;
  else
    x'dot == -2.0;
  end use;
  if yInc = '1' use
    y'dot == 2.0;
  else
    y'dot == -2.0;
  end use;
  xcoord:process begin
    wait until x'above(120.0);
    assign(xInc,'0',0,0);
    wait until not x'above(0.0);
    assign(xInc,'1',0,0);
  end process xcoord;
  ycoord:process begin
    wait until y'above(320.0);
    assign(yInc,'0',0,0);
    wait until not y'above(0.0);
    assign(xInc,'1',0,0);
  end process ycoord;
  assert not x'above(58.0) or x'above(62.0) or
    not y'above(298.0) or y'above(302.0)
    report "Error:  Reached white ball."
    severity failure;
end balls;
```

Figure 6.6. VHDL-AMS code for the billiards game.

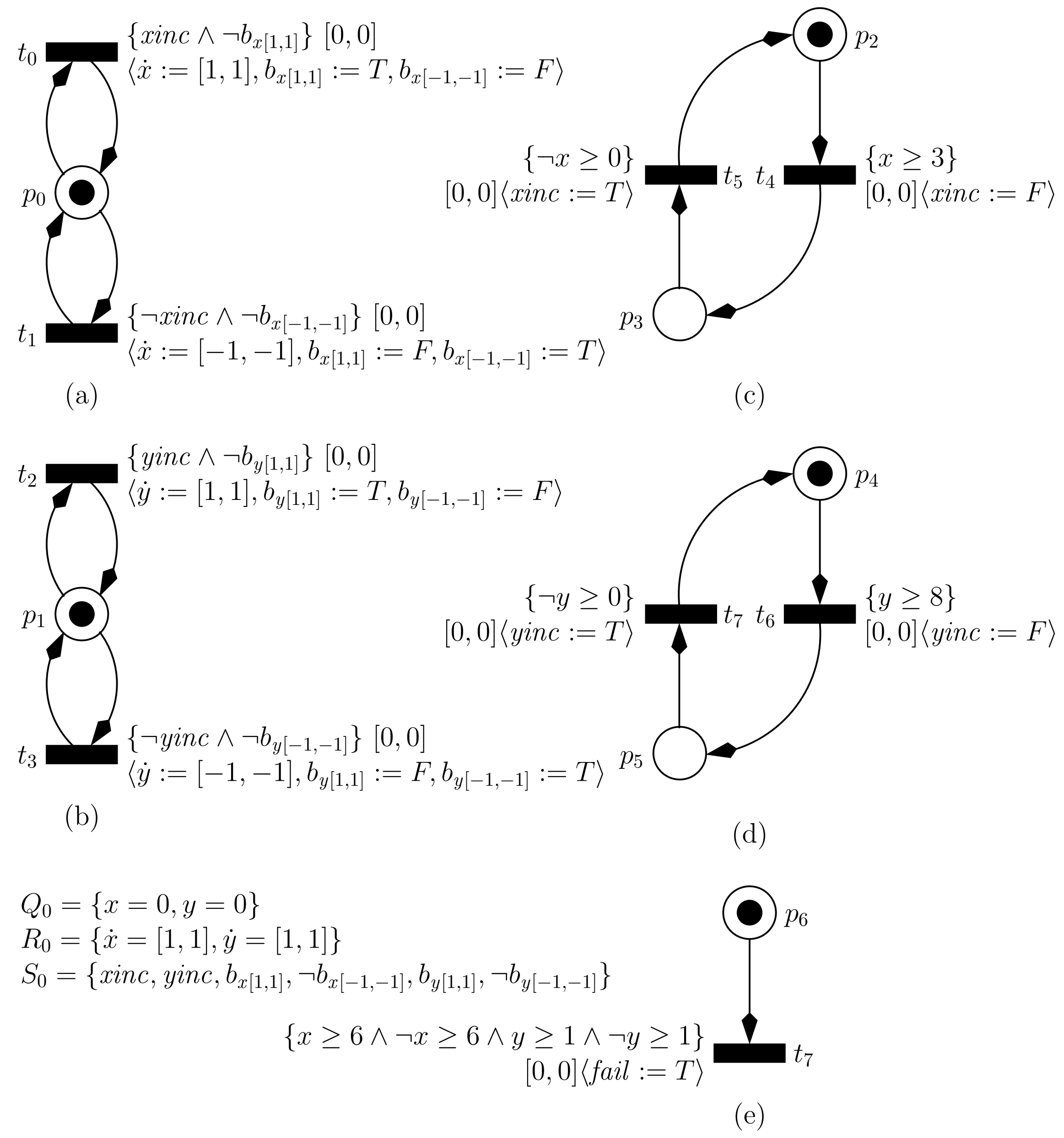

t_0
$\{xinc \wedge \neg b_{x[1,1]}\}$ $[0,0]$
$\langle \dot{x} := [1,1], b_{x[1,1]} := T, b_{x[-1,-1]} := F\rangle$
p_0
t_1
$\{\neg xinc \wedge \neg b_{x[-1,-1]}\}$ $[0,0]$
$\langle \dot{x} := [-1,-1], b_{x[1,1]} := F, b_{x[-1,-1]} := T\rangle$
(a)
p_2
$\{\neg x \geq 0\}$
$[0,0]\langle xinc := T\rangle$
t_5
t_4
$\{x \geq 3\}$
$[0,0]\langle xinc := F\rangle$
p_3
(c)
t_2
$\{yinc \wedge \neg b_{y[1,1]}\}$ $[0,0]$
$\langle \dot{y} := [1,1], b_{y[1,1]} := T, b_{y[-1,-1]} := F\rangle$
p_1
t_3
$\{\neg yinc \wedge \neg b_{y[-1,-1]}\}$ $[0,0]$
$\langle \dot{y} := [-1,-1], b_{y[1,1]} := F, b_{y[-1,-1]} := T\rangle$
(b)
p_4
$\{\neg y \geq 0\}$
$[0,0]\langle yinc := T\rangle$
t_7
t_6
$\{y \geq 8\}$
$[0,0]\langle yinc := F\rangle$
p_5
(d)
$Q_0 = \{x = 0, y = 0\}$
$R_0 = \{\dot{x} = [1,1], \dot{y} = [1,1]\}$
$S_0 = \{xinc, yinc, b_{x[1,1]}, \neg b_{x[-1,-1]}, b_{y[1,1]}, \neg b_{y[-1,-1]}\}$
p_6
$\{x \geq 6 \wedge \neg x \geq 6 \wedge y \geq 1 \wedge \neg y \geq 1\}$
$[0,0]\langle fail := T\rangle$
t_7
(e)

Figure 6.7. An LHPN model of the billiards game compiled from the VHDL-AMS description in Figure 6.6.

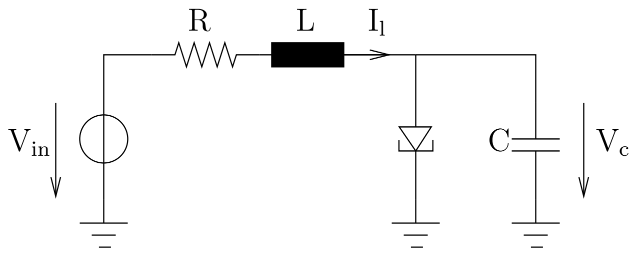
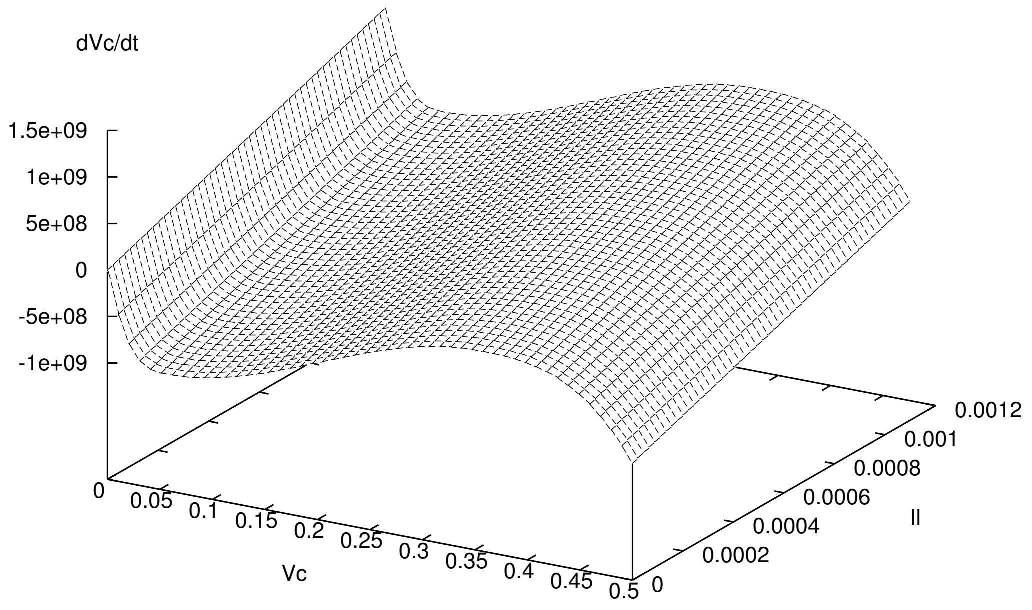


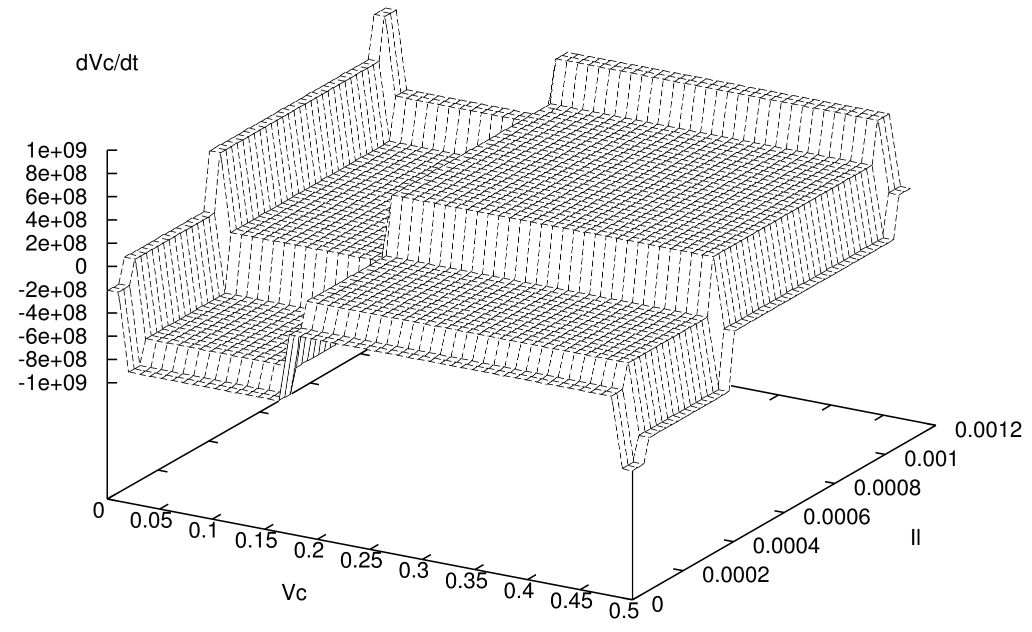
Figure 6.8. Tunnel diode oscillator circuit (V_{in} = 0.3 V, L = 1 μH, and C = 1 pF).

in which a range of rates can be calculated. This is illustrated as the continuous region in Figure 6.9a is decomposed into the regions in Figure 6.9b using the differential equation approximation method proposed by Walter in [157]. Any discretization method with the goal of minimizing the resulting number of regions may be utilized; however, we require that, regions must be rectangular in shape and have only a single neighboring region on each side in each dimension. We use a discretization approach similar to that proposed in [74, 75]. After decomposing the continuous domain into regions, a VHDL-AMS model is created by hand using an architecture similar to that produced by `LEMA`'s model generator as shown in Figure 6.10. The oscillation property is verified using an assert statement on a watchdog timer. Each time I_l goes below 0.3 mA (30000 in the VHDL-AMS code) or goes above 0.7 mA (70000 in the VHDL-AMS code) the watchdog timer is reset. The assert statement ensures that the value of the watchdog timer does not exceed the expected period of oscillation.

In the model for the tunnel diode oscillator, sixteen discrete regions are required to model the oscillatory/nonoscillatory behavior of the circuit resulting in a compiled LHPN for the oscillatory version with 19 places, 30 transitions, and 32 unique rates with up to 3 digits of precision. Table 6.2 shows results from `LEMA`'s DBM-based analyzer, HyTech, and `PHAVer` for two versions of the tunnel diode oscillator circuit. The property is verified for a range of initial conditions in which I_l is between 0.4 to 0.5 mA and V_c is between 0.4 and 0.47 V. As expected, the property verifies with $R = 200\Omega$ in 14.62 s after finding 17703 zones, and the property did not verify with $R = 242\Omega$ in 0.34 s after finding 1826 zones. HyTech [81] is unable to complete verification of the tunnel diode oscillator due to arithmetic overflow errors. HyTech can complete analysis with less precision on the rates, but the model of the circuit no longer produces oscillation. Therefore,



(a)



(b)

Figure 6.9. Decomposing continuous flows into discrete regions. (a) Continuous flow for $\frac{dV_C}{dt}$. (b) Discretized flow for $\frac{dV_C}{dt}$.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.handshake.all;
use work.nondeterminism.all;
entity diode is
end diode;
architecture diode_osc of diode is
  quantity Vc, Il, watchdog, l:real;
begin
  break Vc => span(400000.0,470000.0);
  break Il => span(45000.0,55000.0);
  break l => 1.0, watchdogA => 0.0 when not l'above(0.0) and
    Il'above(70000.0);
  break l => -1.0, watchdogA => 0.0 when l'above(0.0) and not
    Il'above(30000.0);
  watchdog'dot == 1.0;
  if not Vc'above(10200.0) use
    if not Il'above(4500.0) use
      Vc'dot == -207.0;
      Il'dot == 29.0;
    elsif Il'above(6700.0) and not Il'above(38200.0) use
      Vc'dot == -5.0;
      Il'dot == 25.0;
    elsif Il'above(40400.0) and not Il'above(103300.0) use
      Vc'dot == 489.0;
      Il'dot == 15.0;
    elsif Il'above(105500.0) use
      Vc'dot == 848.0;
      Il'dot == 8.0;
    end use;
  elsif Vc'above(20400.0) and not Vc'above(163300.0) use
    if not Il'above(4500.0) use
      Vc'dot == -875.0;
      Il'dot == 20.0;
    elsif Il'above(6700.0) and not Il'above(38200.0) use
      Vc'dot == -673.0;
      Il'dot == 16.0;
    elsif Il'above(40400.0) and not Il'above(103300.0) use
      Vc'dot == -179.0;
      Il'dot == 6.0;
    elsif Il'above(105500.0) use
      Vc'dot == 180.0;
      Il'dot == -1.0;
    end use;
```

Figure 6.10. VHDL-AMS code for the oscillating version of the tunnel diode oscillator.

```
  elsif Vc'above(173500.0) and not Vc'above(479600.0) use
    if not Il'above(4500.0) use
      Vc'dot == -287.0;
      Il'dot == -3.0;
    elsif Il'above(6700.0) and not Il'above(38200.0) use
      Vc'dot == -85.0;
      Il'dot == -7.0;
    elsif Il'above(40400.0) and not Il'above(103300.0) use
      Vc'dot == 409.0;
      Il'dot == -17.0;
    elsif Il'above(105500.0) use
      Vc'dot == 768.0;
      Il'dot == -24.0;
    end use;
  elsif Vc'above(489800.0) use
    if not Il'above(4500.0) use
      Vc'dot == -893.0;
      Il'dot == -20.0;
    elsif Il'above(6700.0) and not Il'above(38200.0) use
      Vc'dot == -690.0;
      Il'dot == -24.0;
    elsif Il'above(40400.0) and not Il'above(103300.0) use
      Vc'dot == -197.0;
      Il'dot == -34.0;
    elsif Il'above(105500.0) use
      Vc'dot == 163.0;
      Il'dot == -41.0;
    end use;
  end use;
  assert not watchdog'above(18000.0)
    report "Error:  Current not oscillating."
    severity failure;
end diode_osc;
```

Figure 6.10 continued.

Table 6.2. Tunnel diode oscillator circuit verification results.

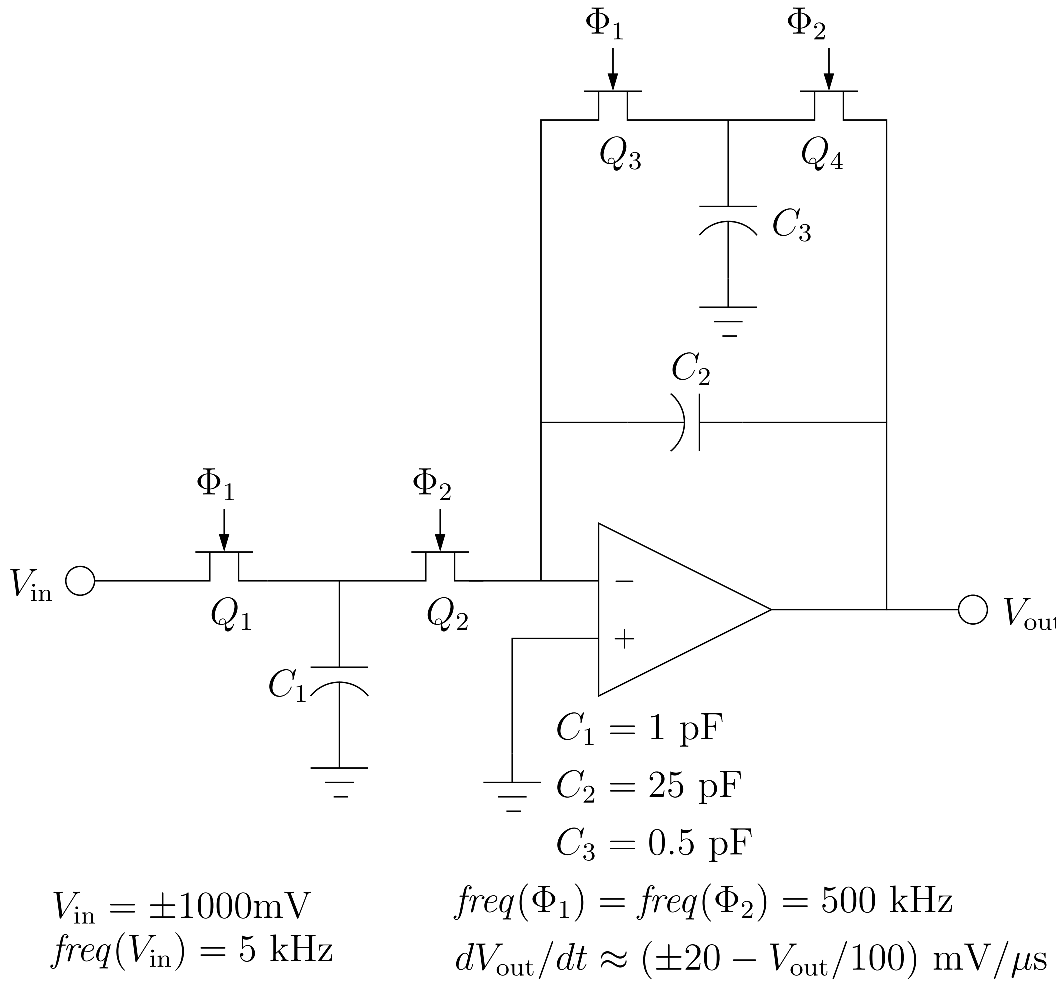
	LEMA DBM-based			HyTech		PHAVer	
Example	Zones	Time(s)	Ok?	Time(s)	Ok?	Time(s)	Ok?
Tunnel diode (osc.)	17703	14.62	Yes	overflow	N/A	72.8	Yes
Tunnel diode (non-osc.)	1826	0.34	No	overflow	N/A	n.r.	N/A

the verification results are incorrect. In [56], it is reported that PHAVer requires 72.8 seconds to verify the oscillating version of the tunnel diode oscillator, and results are not reported for the nonoscillating version. This result shows the efficiency of LEMA's DBM-based model checker compared to a model checker that finds the exact state space. Though runtimes are not reported for the Boolean mapping approach in [75], we believe our analysis method is competitive in runtime, and that it is more accurate since the continuous variables are modeled explicitly.

6.3 Switched Capacitor Integrator

Using LEMA's model generator, two simulation traces of the switched capacitor integrator for C_2=23 pF and C_2=27 pF result in the LHPN shown in Figure 3.6. Although neither of the simulation traces indicates a problem with saturation of the integrator, a state space analysis using the DBM-based model checker finds in less than a second that there is a potential for the circuit to fail. This failure can occur when the integrator charges the capacitor, C_2, at a rate that is on average faster than the rate of discharge. This situation causes charge to build up on the capacitor and eventually results in V_{out} reaching a voltage above 2000 mV. The reason that this method can find this failure is that the LHPN model represents not only each simulation trace, but also the union of the traces. It is this behavior explored by unioning the traces that allows the analyzer to find the flaw in the circuit.

Saturation of the integrator can be prevented using the circuit shown in Figure 6.11. In this circuit, a resistor in the form of a switched capacitor is inserted in parallel with the feedback capacitor. This causes V_{out} to drift back to 0 V. In other words, if V_{out} is increasing, it increases faster when it is far below 0 V than when it is near or above 0 V. This behavior can be seen in the simulation traces in Figure 6.12 as a "bending"



Φ_1
Φ_2
Q_3
Q_4
C_3
C_2
Φ_1
Φ_2
V_{in}
Q_1
Q_2
C_1
V_{out}
$C_1 = 1$ pF
$C_2 = 25$ pF
$C_3 = 0.5$ pF
$V_{\text{in}} = \pm 1000$mV
$freq(V_{\text{in}}) = 5$ kHz
$freq(\Phi_1) = freq(\Phi_2) = 500$ kHz
$dV_{\text{out}}/dt \approx (\pm 20 - V_{\text{out}}/100)$ mV/μs

Figure 6.11. Schematic diagram of a corrected switched capacitor integrator circuit.

of the triangle wave as it moves away from zero. Using the same simulation parameters and thresholds for this circuit, `LEMA`'s model generator obtains an LHPN with the same structure as the one shown in Figure 3.6, but the ranges of rates for each region are different as shown in Table 6.3. These differences mean that the slew rate changes as V_{out} changes as evidenced by the rate change for each place as shown in Figure 6.13. This LHPN also fails the property as the thresholds are too simple to capture the effect of the additional switched capacitor. Due to the addition of this switched capacitor resistor, the rate of change is now dependent on the value of V_{out}. In particular, this variation slows the rate of the voltage change as it approaches the power supply rail. This prevents saturation of the integrator. Based on this knowledge, the thresholds on V_{out} are changed to -500 mV, 0 V, and 500 mV. These new thresholds result in the rates

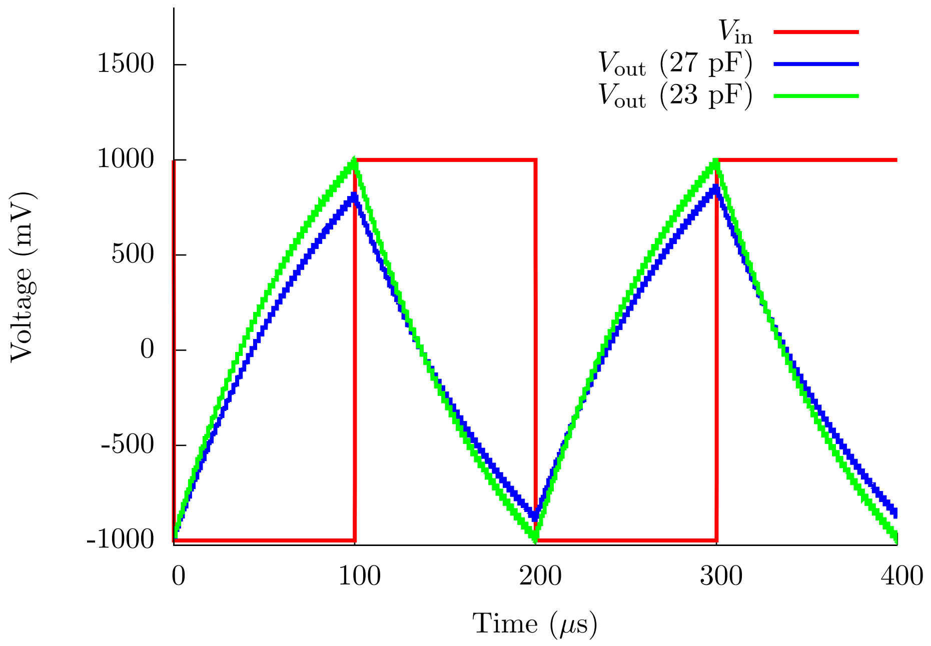


Figure 6.12. Simulation traces for the corrected switched capacitor integrator circuit.

Table 6.3. Rates for V_{out} from the corrected switched capacitor integrator circuit using a threshold of 0 V.

Bin	Place	Range of rates	Comment
00	p_6	[18,32]	$V_{\text{in}} < 0$ V; $V_{\text{out}} < 0$ V
01	p_3	[9,22]	$V_{\text{in}} < 0$ V; $V_{\text{out}} \geq 0$ V
11	p_4	[-22,-9]	$V_{\text{in}} \geq 0$ V; $V_{\text{out}} \geq 0$ V
10	p_5	[-32,-18]	$V_{\text{in}} \geq 0$ V; $V_{\text{out}} < 0$ V

shown in Table 6.4. Figure 6.14 shows the LHPN for this table, and this LHPN is found to satisfy the property in less than a second using `LEMA`'s DBM-based model checker.

Table 6.5 summarizes the verification results for the switched capacitor integrator circuit using `LEMA`'s model checking engines. All of the models verify as expected in a short time using the DBM-based model checker. The BDD-based and SMT-based engines require significantly more time to complete the verification. Also, it should be noted that due to a conservative approximation, the BDD-based model checker produces a false negative result on the VHDL-AMS version of the corrected switched capacitor

Initial values = $\{V_{\text{out}} = -1000 \text{ mV}, V_{\text{in}} = -1000 \text{ mV}\}$; Initial rates = $\{\dot{V}_{\text{in}} = 0, \dot{V}_{\text{out}} = [18, 32]\}$

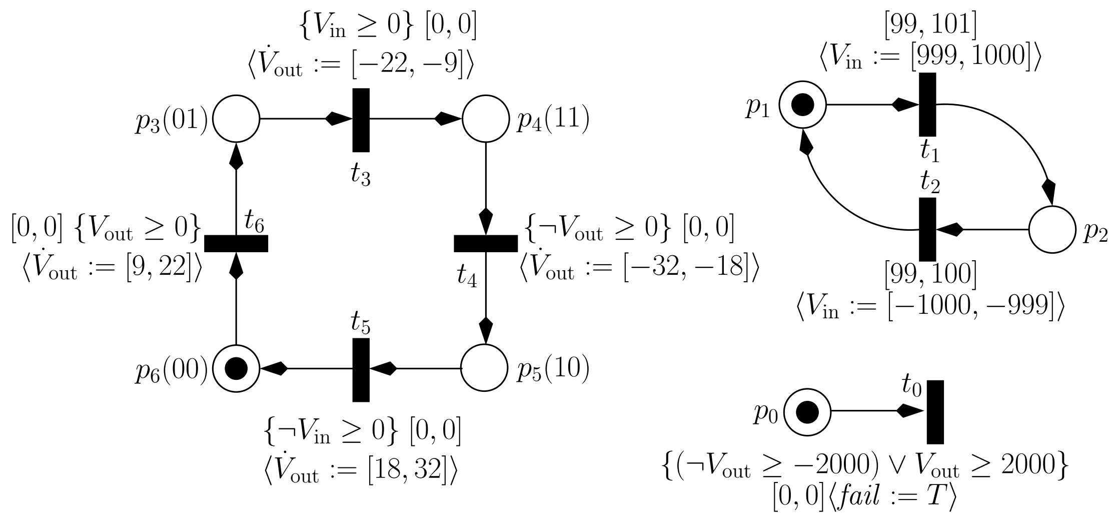
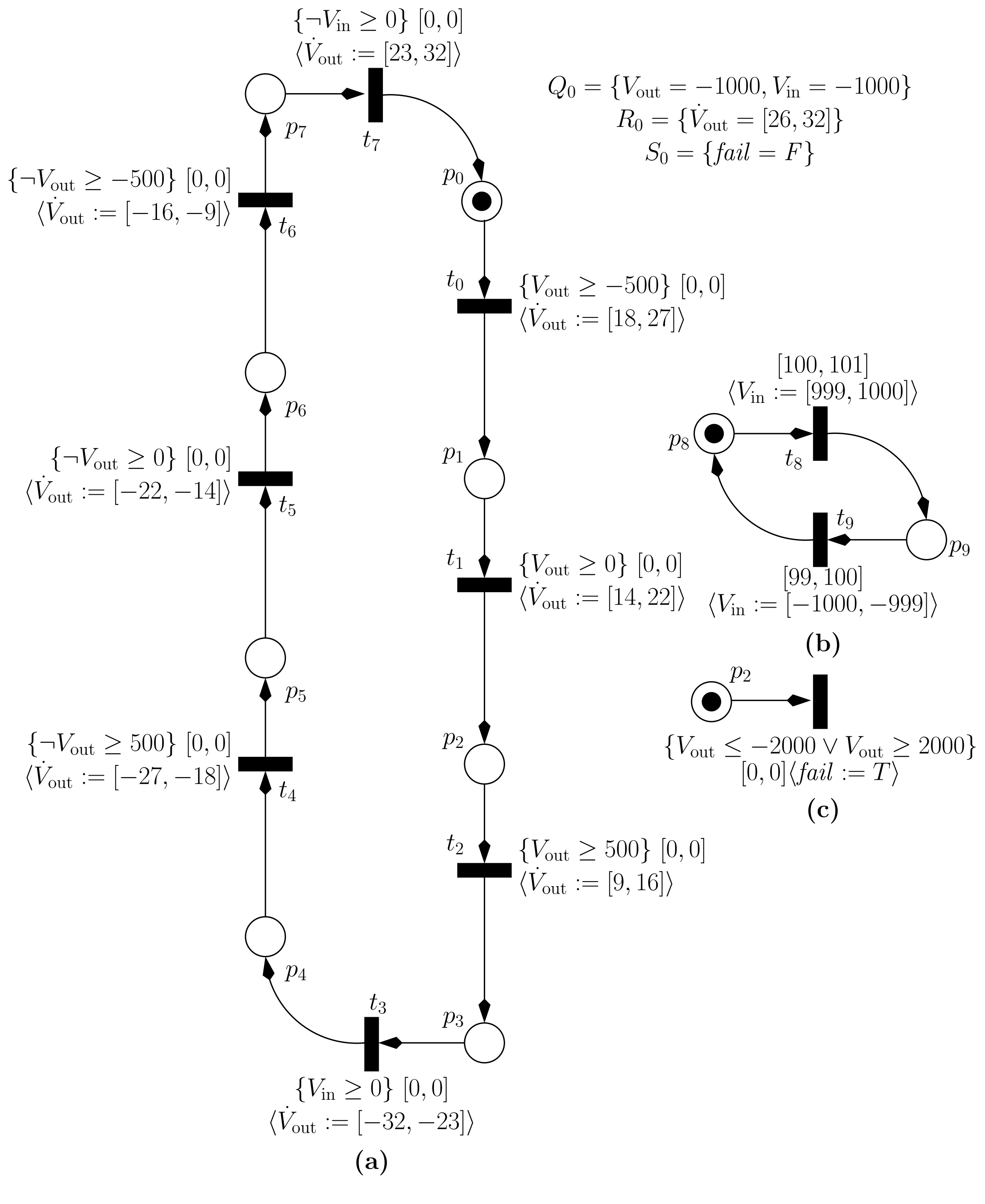

$\{V_{\text{in}} \geq 0\}$ $[0, 0]$
$\langle \dot{V}_{\text{out}} := [-22, -9] \rangle$
$p_3(01)$
t_3
$p_4(11)$
$[0, 0]$ $\{V_{\text{out}} \geq 0\}$
$\langle \dot{V}_{\text{out}} := [9, 22] \rangle$
t_6
$\{\neg V_{\text{out}} \geq 0\}$ $[0, 0]$
$\langle \dot{V}_{\text{out}} := [-32, -18] \rangle$
t_4
t_5
$p_6(00)$
$p_5(10)$
$\{\neg V_{\text{in}} \geq 0\}$ $[0, 0]$
$\langle \dot{V}_{\text{out}} := [18, 32] \rangle$
$[99, 101]$
$\langle V_{\text{in}} := [999, 1000] \rangle$
p_1
t_1
t_2
p_2
$[99, 100]$
$\langle V_{\text{in}} := [-1000, -999] \rangle$
t_0
p_0
$\{(\neg V_{\text{out}} \geq -2000) \vee V_{\text{out}} \geq 2000\}$
$[0, 0] \langle fail := T \rangle$

Figure 6.13. An LHPN for the corrected switched capacitor integrator circuit using a single threshold of 0 V for V_{in} and V_{out}.

Table 6.4. Rates for V_{out} from the corrected switched capacitor integrator circuit using thresholds of -500 mV, 0 V, and 500 mV.

Bin	Place	Range of rates	Comment
00	p_9	[23,32]	$V_{\text{in}} < 0$ V; $V_{\text{out}} < -500$ mV
01	p_7	[18,27]	$V_{\text{in}} < 0$ V; -500 mV $\leq V_{\text{out}} < 0$ V
02	p_5	[14,22]	$V_{\text{in}} < 0$ V; 0 V $\leq V_{\text{out}} < 500$ mV
03	p_3	[9,16]	$V_{\text{in}} < 0$ V; $V_{\text{out}} \geq 500$ mV
10	p_{10}	[-16,-9]	$V_{\text{in}} \geq 0$ V; $V_{\text{out}} < -500$ mV
11	p_8	[-22,-14]	$V_{\text{in}} \geq 0$ V; -500 mV $\leq V_{\text{out}} < 0$ V
12	p_6	[-27,-18]	$V_{\text{in}} \geq 0$ V; 0 V $\leq V_{\text{out}} < 500$ mV
13	p_4	[-32,-23]	$V_{\text{in}} \geq 0$ V; $V_{\text{out}} \geq 500$ mV



$\{\neg V_{\text{in}} \geq 0\}$ $[0,0]$
$\langle \dot{V}_{\text{out}} := [23,32] \rangle$
p_7
t_7
$Q_0 = \{V_{\text{out}} = -1000, V_{\text{in}} = -1000\}$
$R_0 = \{\dot{V}_{\text{out}} = [26,32]\}$
$S_0 = \{fail = F\}$
$\{\neg V_{\text{out}} \geq -500\}$ $[0,0]$
$\langle \dot{V}_{\text{out}} := [-16,-9] \rangle$
t_6
p_0
t_0
$\{V_{\text{out}} \geq -500\}$ $[0,0]$
$\langle \dot{V}_{\text{out}} := [18,27] \rangle$
p_6
$\{\neg V_{\text{out}} \geq 0\}$ $[0,0]$
$\langle \dot{V}_{\text{out}} := [-22,-14] \rangle$
t_5
p_1
t_1
$\{V_{\text{out}} \geq 0\}$ $[0,0]$
$\langle \dot{V}_{\text{out}} := [14,22] \rangle$
p_5
$\{\neg V_{\text{out}} \geq 500\}$ $[0,0]$
$\langle \dot{V}_{\text{out}} := [-27,-18] \rangle$
t_4
p_2
t_2
$\{V_{\text{out}} \geq 500\}$ $[0,0]$
$\langle \dot{V}_{\text{out}} := [9,16] \rangle$
p_4
t_3
p_3
$\{V_{\text{in}} \geq 0\}$ $[0,0]$
$\langle \dot{V}_{\text{out}} := [-32,-23] \rangle$
(a)
$[100,101]$
$\langle V_{\text{in}} := [999,1000] \rangle$
p_8
t_8
t_9
p_9
$[99,100]$
$\langle V_{\text{in}} := [-1000,-999] \rangle$
(b)
p_2
$\{V_{\text{out}} \leq -2000 \vee V_{\text{out}} \geq 2000\}$
$[0,0]\langle fail := T \rangle$
(c)

Figure 6.14. An LHPN for the corrected switched capacitor integrator circuit example using thresholds of −500 mV, 0 V, and 500 mV.

integrator. Since the SMT-based model checker is a bounded model checker, the number of iterations used to find an error or verify the property are listed. Finally, to explore the scalability of our algorithms, Table 6.6 shows how the size of the LHPN and model checking time scales as the number of thresholds increases. While runtime is increasing quickly, it is still very small even for nine thresholds. The state space exploration engine's ability to scale is based upon the complexity of the state space. The complexity of the state space is related to the number of continuous variables and number and complexity of the rates of the continuous variables.

6.4 PLL Phase Detector

PLLs are notoriously difficult circuits to design and validate. There are a small number of major components to a PLL which traditionally include a phase detector, low pass filter, VCO, and a frequency divider. `LEMA` analyzes a phase detector as shown in the schematic diagram of Figure 6.15. The phase detector is used to measure the phase

Table 6.5. Switched capacitor integrator circuit verification results for versions compiled form VHDL-AMS and generated from SPICE simulations. OOM indicates that the exploration ran out of memory.

		DBM-based		BDD-based	SMT-based	
Example	Verifies?	Zones	Time(s)	Time(s)	Iterations	Time(s)
VHDL-AMS [20,20]	Yes	4	0.03	0.3	30	7.3
VHDL-AMS [18,22]	No	11	0.04	1.2	15	0.85
Cor. VHDL-AMS	Yes	266	0.35	7.7*	30	589.2
Original SPICE	No	20	0.03	20.2	15	4.2
Cor. SPICE	Yes	73	0.05	OOM	30	1406

* Verification result does not match the expected result.

Table 6.6. Scalability of `LEMA`'s DBM-based model checker as the number of thresholds increase.

No. thresholds	Places	Transitions	Model Checking Time
1	7	7	0.03s
3	11	11	0.06s
5	14	14	0.19s
7	18	19	0.31s
9	22	27	0.62s

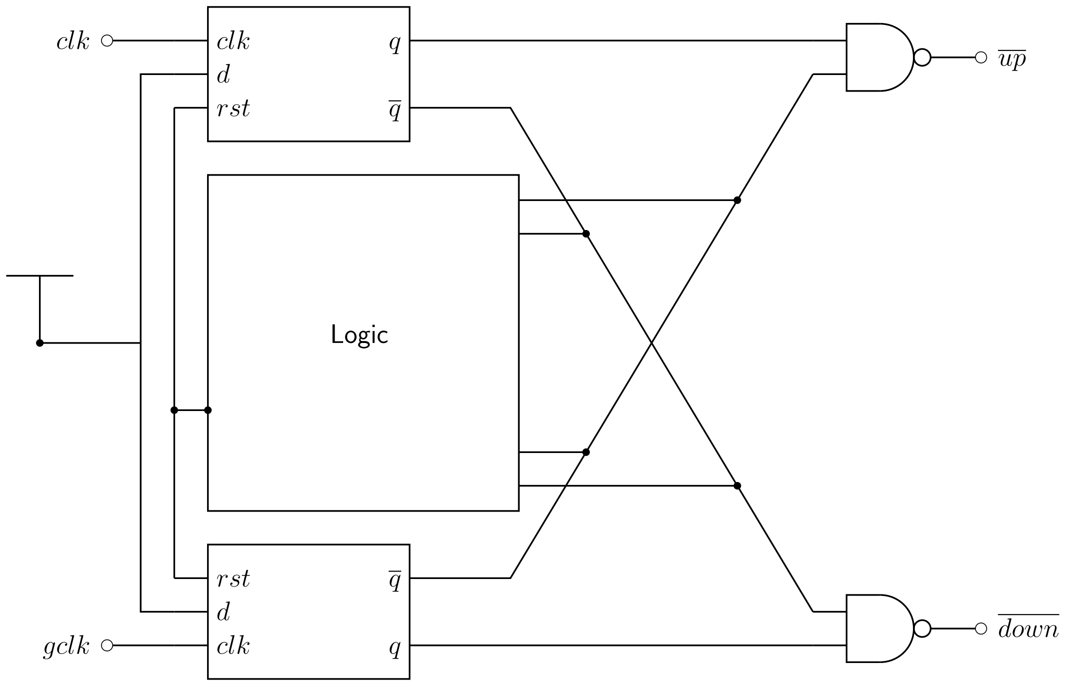


Figure 6.15. A schematic diagram for a PLL phase detector.

difference between two input signals and provide this information to the VCO. The VCO uses this information to adjust the phase of the internal PLL clock in order to align the phase of the two clock signals. The inputs to the phase detector are *clk* and *gclk*. The output signals, $\overline{up}$ and $\overline{down}$, are asserted to provide instruction on how to adjust the VCO frequency.

The phase detector is simulated using a piecewise linear simulation input 1 μs long representing reasonable clock skew for the *clk* input and a periodic clock signal for the *gclk* input. The input waveforms are reversed and a second simulation is run. The phase detector is simulated two more times using two periodic signals of fixed but different periods. In one simulation, *clk* is leading and *gclk* is lagging. In the next simulation that condition is reversed. These four simulations are used to build the LHPN and Verilog-AMS models of the phase detector.

The Verilog-AMS and LHPN model are generated in approximately 20 seconds for the phase detector example using the four simulations described previously. Eight variables are required for model generation to accurately capture the state of the phase detector. Four of the signals are the inputs, *clk* and *gclk*, and outputs, $\overline{up}$ and $\overline{down}$, while the remaining four signals are chosen from the internal signals of the latches. It is logical that signals from the state holding latches are required to delineate the states of the phase detector. As all eight variables are digital signals, each variable is assigned a single threshold equal to $\frac{V_{\mathrm{dd}}}{2}$.

Comparison between simulation times for the transistor-level design and the Verilog-AMS model are performed using the same simulation inputs and simulator. Table 6.7 presents the results of these simulations. The first four simulations use one piecewise linear input and one periodic input. The final table entry is a result for two periodic inputs. Figure 6.16 shows a comparison of the waveforms produced by the two simulations. There is a slight timing difference between the two waveforms, but the abstracted model is accurate enough to be used in a system-level simulation.

The LHPN model of the PLL phase detector is composed of 69 places and 87 transitions. The property $\neg(\overline{down} \wedge \overline{up})$ verifies in 0.3 seconds. This property is a sanity check on the outputs of the phase detector ensuring that $\overline{down}$ and $\overline{up}$ are not asserted at the same time. A more complex property for the PLL phase detector is shown as pseudocode in Figure 6.17. This pseudocode is a behavioral description of the correct input/output operation of the phase detector. This property cannot be specified directly in LEMA's property language. To verify the property it is necessary to convert the property to an LHPN with transitions used to indicate a failure of the property. The resulting property LHPN is composed of 20 transitions and 14 places. The half of the property LHPN that checks the first three lines of the property is shown in Figure 6.18. The LHPN representing the bottom three lines of the property is similar. Composing these property LHPNs with the LHPN model for the PLL phase detector results in an LHPN that verifies correctly in 2.12 seconds.

Table 6.7. A comparison of simulation times between the transistor-level model and the Verilog-AMS model of the PLL phase detector.

Sim	Verilog-AMS (s)	Transistor (s)	Speed-up
0.5 μs	0.54	18.28	33.8
0.5 μs	0.54	17.92	33.2
1.0 μs	0.81	36.67	45.3
1.0 μs	0.81	40.46	49.9
2.0 μs	0.38	9.47	24.9

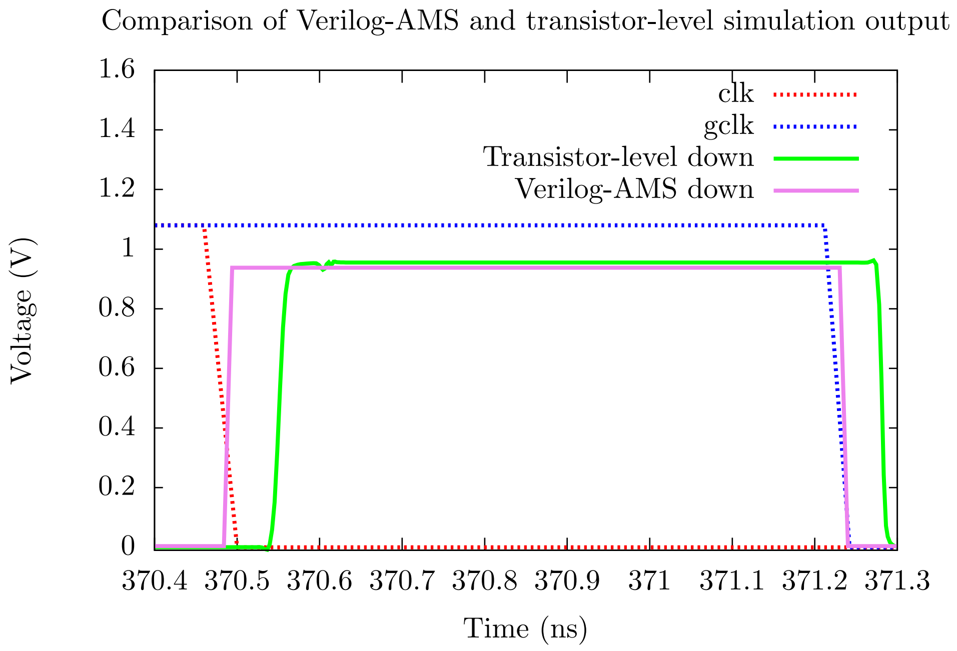


Figure 6.16. A comparison of simulation waveforms from the transistor-level and Verilog-AMS PLL phase detector models.

```
if gclk 1 → 0 then
  if up = 1 then up = 0 within 5 ns
  elsif down = 0 then down = 1 within 5 ns
if clk 1 → 0 then
  if down = 1 then down = 0 within 5 ns
  elsif up = 0 then up = 1 within 5 ns
```

Figure 6.17. A property to verify for the PLL phase detector.

6.5 CMOS Ring Oscillator with Feedforward Inverters

Engineers at Rambus recently suggested investigating the modeling and verification of the ring oscillator in Figure 6.19 [87]. Traditional CMOS ring oscillators are created using an odd number of inverters in the ring. These oscillators have an oscillation period of 2Nτ_d, where τ_d is an inverter delay. The period of oscillation can be reduced to Nτ_d by using *feedforward inverters* to send the signal ahead in the ring [69]. An advantage of using feedforward inverters is that the inverter ring can be designed to oscillate using an even number of inverters. This allows for *quadrature outputs* (i.e., four outputs where each

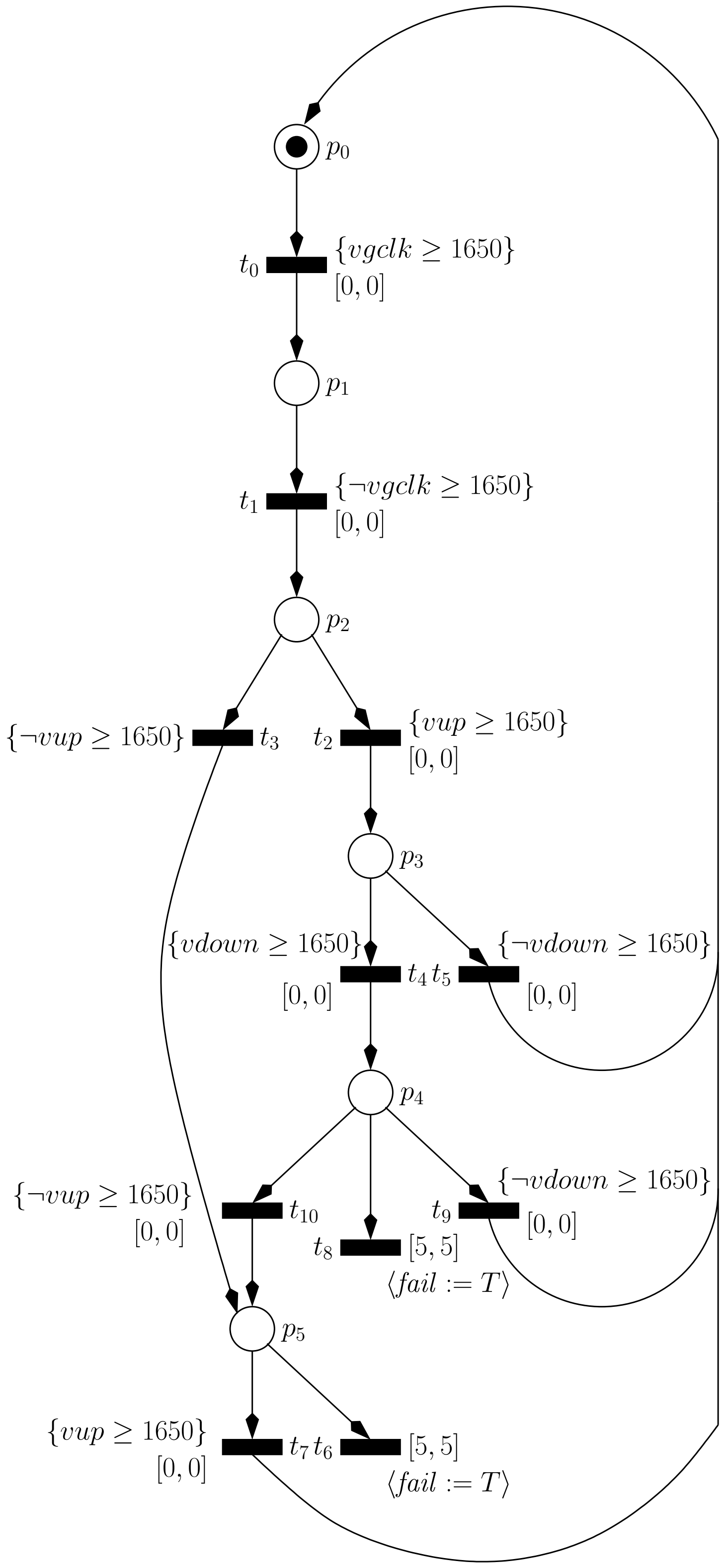


Figure 6.18. An LHPN for the first three lines of the property in Figure 6.17.

output is 90° degrees out of phase with the previous output). This is not possible with a ring oscillator employing an odd number of inverters. Quadrature outputs are becoming increasingly important in clock recovery and multiphase processor clocking. For these reasons, Rambus uses this circuit in many of their VCO designs. Despite having an essentially digital specification, this circuit requires an analog analysis, since its behavior cannot be reasonably analyzed at the switch-level.

This circuit only oscillates for a particular range of ratios in transistor sizes for the chain inverters to feedforward inverters. If the feedforward inverters ($F_1 - F_4$) are much larger than the chain inverters ($C_1 - C_4$), the feedforward inverters overpower the chain inverters and latch the values X and Y at opposite logical values. A simulation using a ratio of chain to feedforward inverters of 0.39 exhibits this behavior as shown in Figure 6.20. If the ratio is changed to 0.40, the circuit begins to oscillate slowly with X leading Y as demonstrated by the simulation in Figure 6.21. As the ratio increases, the speed of oscillation increases. Approximately in the middle of the range of oscillating ratios, a ratio of 1.60, the simulation in Figure 6.22 demonstrates the stability of the oscillator. As the ratio increases to 2.275, the chain inverters begin to overpower the feedforward inverters, and the circuit struggles to start oscillating. It does eventually start oscillating and is stable after the initial startup period as demonstrated in Figure 6.23.

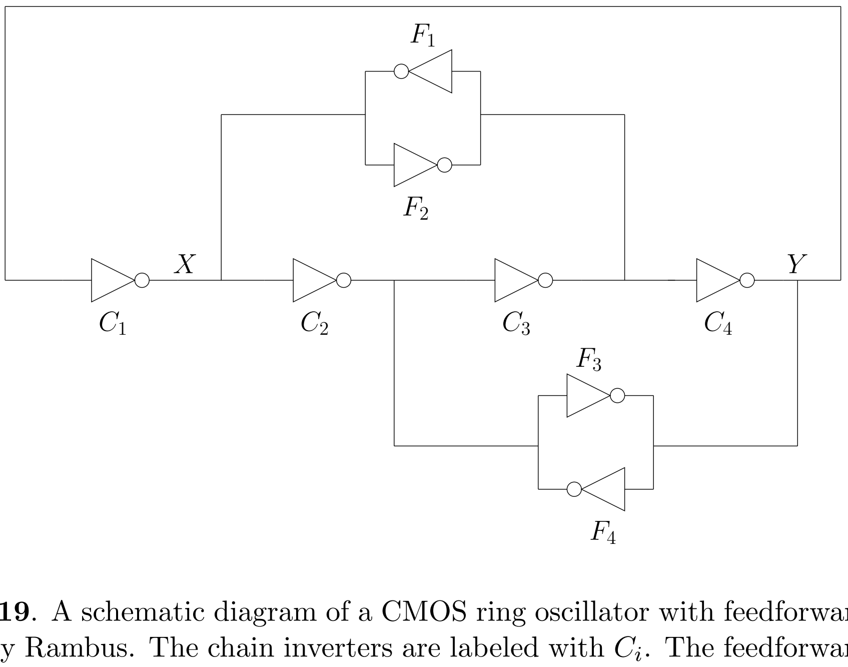


Figure 6.19. A schematic diagram of a CMOS ring oscillator with feedforward inverters provided by Rambus. The chain inverters are labeled with C_i. The feedforward inverters are labeled with F_i.

Finally, Figure 6.24 shows the simulation results for a ratio of 2.30 where the chain inverters overpower the feedforward inverters and produce the expected result for a ring oscillator with an even number of inverters. In other words, the oscillator begins to oscillate but reaches equilibrium with X and Y at opposite values and stops oscillating.

The behavior described above provides an interesting modeling and verification challenge. Using a single simulation, a designer may be convinced that the circuit oscillates. However, if the circuit is on the edge of the oscillating ratio, it is possible that process variation could cause regular failures of the circuit. Therefore, a model created from simulations near the center of the oscillating transistor ratios should upon analysis find that the circuit always oscillates. In contrast, a model generated from simulations near the edge of the acceptable region should show the possibility of this failure. This result makes the designer aware of the potential failure for the inverter sizes being used. The designer could then do a set of simulations to characterize the oscillator and center the circuit within the acceptable operating range.

`LEMA`'s model generator produces a model from the simulation data for the ring oscillator. For example, the simulation data for the ratio of 1.60 produces the model shown in Figure 6.25. The simulation data only include the variables X and Y because

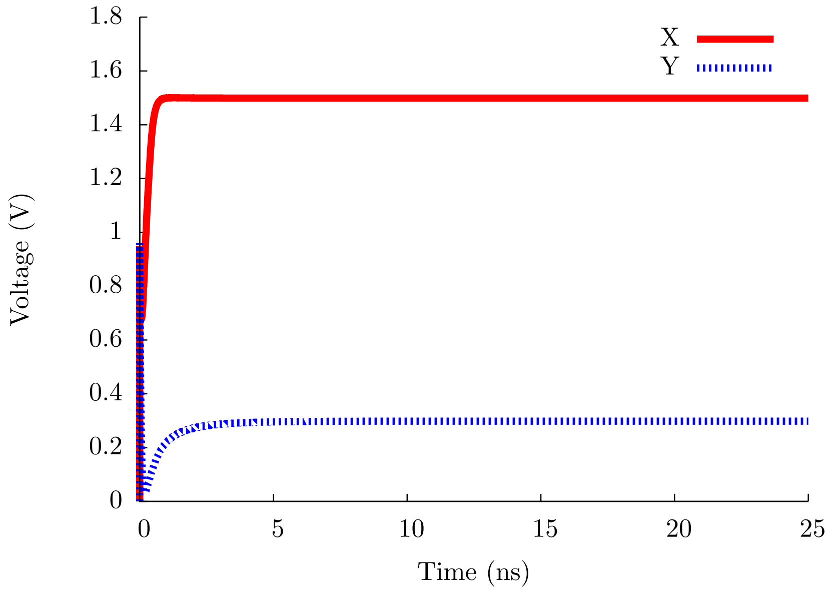


Figure 6.20. A CMOS ring oscillator simulation trace with a ratio of chain to feedforward inverters of 0.39.

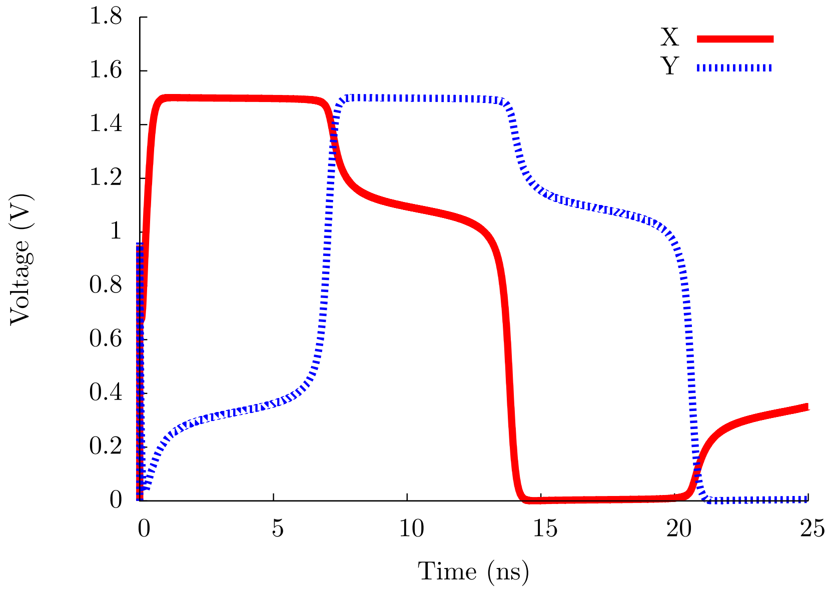


Figure 6.21. A CMOS ring oscillator simulation trace with a ratio of chain to feedforward inverters of 0.40.

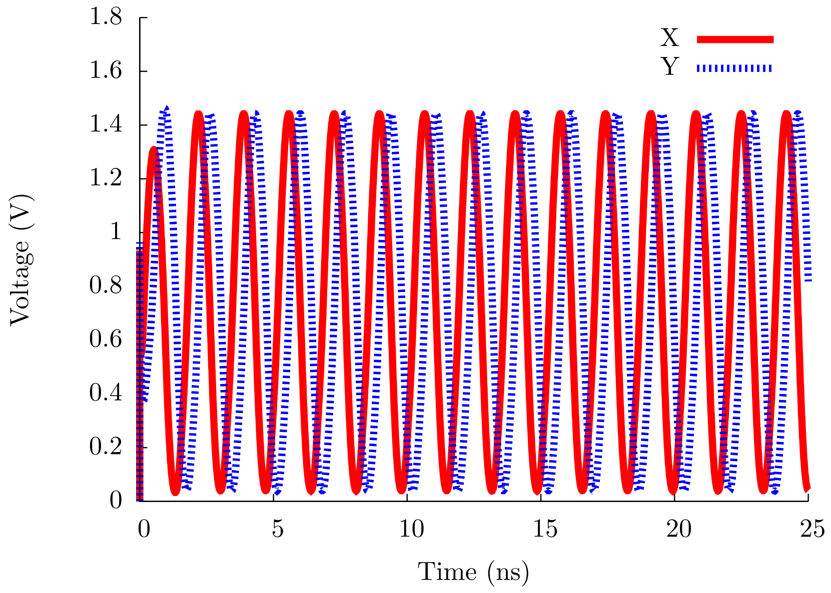


Figure 6.22. A CMOS ring oscillator simulation trace with a ratio of chain to feedforward inverters of 1.60.

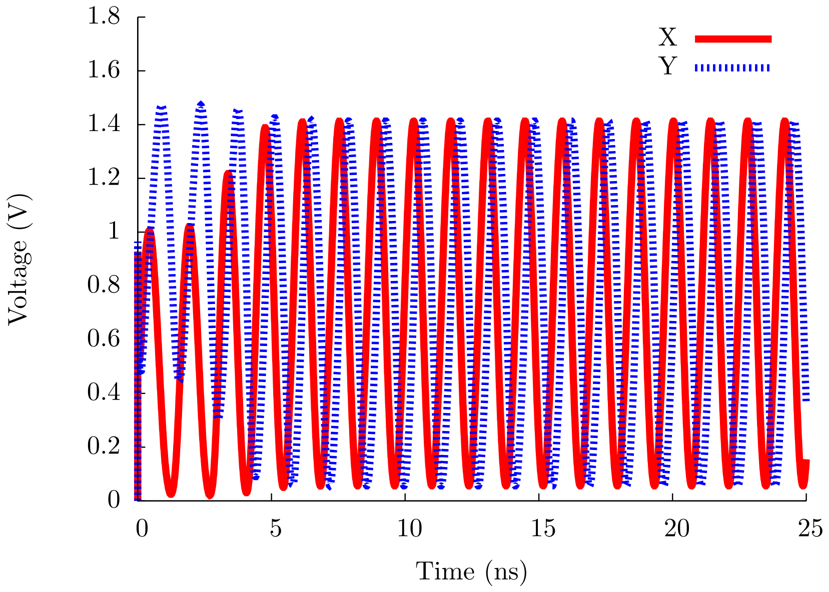


Figure 6.23. A CMOS ring oscillator simulation trace with a ratio of chain to feedforward inverters of 2.275.

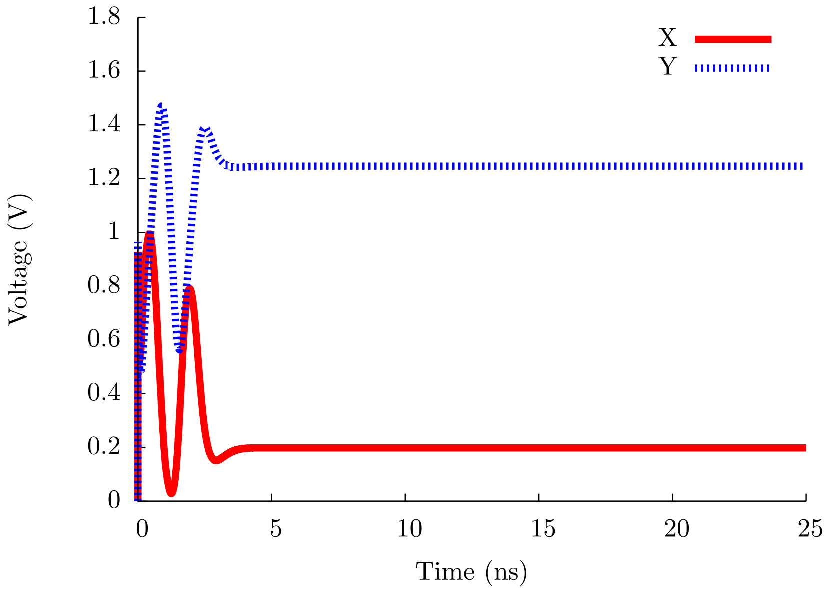


Figure 6.24. A CMOS ring oscillator simulation trace with a ratio of chain to feedforward inverters of 2.30.

other nodes in the inverter chain are correlated to X and Y. Correlated variables do not add any useful information to the generated model. These correlation relationships are determined manually through designer knowledge and experimentation. Following the solid arrows in this model sequentially from p0 to p15 shows that X and Y oscillate with X leading Y. Starting from p0 and moving to p4, X is decreasing as Y remains largely stable. Because Y lags X, in the progression from p4 to p8, Y decreases while X stabilizes. In the path from p8 to p12, X increases while Y is stable. To finish the cycle, Y increases while X remains stable. This is the only path through the region space found during the model generation. This single path indicates stable oscillation. The dashed arrows have been added as described above to allow additional behaviors that may be encountered during formal analysis of this model.

The simulation data for the ratio of 2.275 produce the model shown in Figure 6.26. While this model still includes an oscillating path similar to the one in the previous model, it also includes numerous other paths that represent potentially nonoscillating or truncated oscillations. Since the regions are numbered in the order they are encountered, the initial path through the regions from p0 to p11 shows truncated oscillations. A path with nontruncated oscillations follows the following sequence: p0, p14, p2, p3, p4, p5, p19, p15, p6, p12, p18, p16, p17, p9, p13, p11, and back to p0. The presence of truncated and nontruncated oscillations should indicate to the designer that this set of transistor ratios is potentially on the margin of correct operation, and that the ratios likely need to be modified.

The major verification concern with the ring oscillator circuit is stability of oscillation. From the models produced by `LEMA`'s model generator, it is possible to provide the designer with a measure of stability for the circuit. Based on a stability assurance measure provided by the verification tool for a given simulation, the designer would either be satisfied or use additional simulations to characterize the design space. The model built from the simulation with a ratio of 1.60 has a very high stability score as there is only a single oscillatory path through the state space. The model built from the simulation with a ratio of 2.275 would have a lower stability score as there are multiple oscillatory paths through the state space. The stability score for the second model is still reasonably high as all paths do oscillate. A model generated from a ratio of 2.30 would include paths through the state space that terminate without full oscillation, so it would have the lowest score.

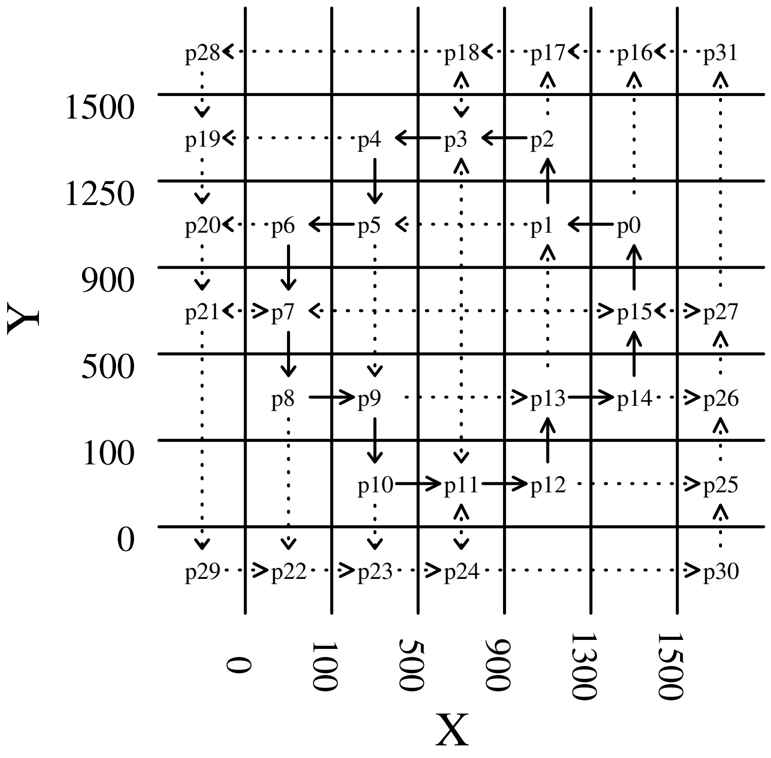


Figure 6.25. An LHPN model of the ring oscillator for the simulation in Figure 6.22.

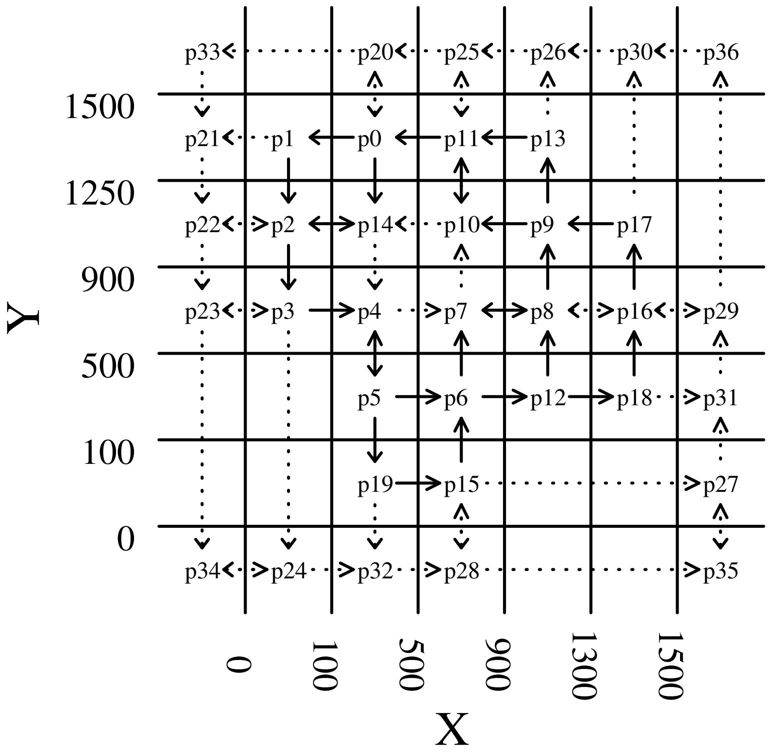


Figure 6.26. An LHPN model of the ring oscillator for the simulation in Figure 6.23.

CHAPTER 7

CONCLUSIONS

The growing complexity of AMS circuits has led to a situation where traditional verification techniques are allowing an unacceptable number of escaped bugs. Improved verification tools and methodologies are needed for AMS circuits. This dissertation describes tools and methodologies that improve upon the current state of the art for AMS verification. A new modeling formalism for AMS circuits, LHPNs, can be compiled from VHDL-AMS or automatically generated from simulation traces. Improved abstract modeling is key to improving system-level simulation and verification. An automatic model generation methodology to create abstract VHDL-AMS and Verilog-AMS models from simulation traces provides improved abstract system-level models for simulation. Formal methods can provide a higher level of verification assurance. DBM-based model checking algorithms for LHPNs complete the path from simulation traces to formal verification. The methods presented in this dissertation show promise to improve AMS circuit verification on industrial examples.

7.1 Summary

This dissertation describes a new SAV methodology, implemented in `LEMA`, to verify AMS circuits. The development of a formal AMS circuit model, LHPNs, amenable to compilation and automatic generation is key. Previously developed modeling formalisms can model AMS circuits but are difficult to generate. The ease of generating LHPNs results in a compiler from VHDL-AMS to LHPNs as well as model generation techniques from simulation traces to system-level VHDL-AMS models, system-level Verilog-AMS models, and LHPNs.

One of the primary challenges facing formal methods for AMS circuits is the difficulty of creating formal models from the circuit descriptions used by AMS designers. In the simulation centric AMS design methodology, simulation data are not a commodity in short supply. Using simulation data that are readily available, `LEMA`'s model generator creates formal models capable of verifying the AMS circuit under simulation. Furthermore, simulation of entire AMS systems is not practical due to the lack of abstract models available for AMS circuits. Using the same simulations, `LEMA`'s model generator creates abstract VHDL-AMS and Verilog-AMS models. These models are composable into full system models that can be simulated in a fraction of the time of the unabstracted transistor-level model.

Another of the primary challenges facing formal methods for AMS circuits is the complexity of performing state space exploration on AMS circuit models. `LEMA`'s DBM-based model checking engine is an efficient model checker for LHPNs, representing AMS circuits. By extending work in the timing verification community, the highly efficient DBM representation can conservatively represent the continuous state space of LHPNs. The conservative state space may lead to errors being reported that are not present in the LHPN model. To help address these false negatives, `LEMA` produces error traces representing the failure traces that can be inspected by the designer for validity.

The effectiveness of the SAV methodology supported by `LEMA` is evidenced through the verification results on several benchmark examples. The ease of model generation and efficiency of the DBM-based model checker make it possible for `LEMA` to successfully analyze a variety of benchmark and industrial examples. The results of these examples show the promise of this new SAV methodology as well as highlighting areas of future work.

7.2 Future Work

While this dissertation describes progress in several areas of AMS circuit verification, there is still significant work that needs to be done in order to make automated AMS circuit verification practical for industrial use. This section proposes a number of areas in need of further research.

7.2.1 Automated Model Abstraction

Our initial study of the ring oscillator in Section 6.5 is promising, but it has highlighted several areas of future work for `LEMA`'s model generator. While the model generation process itself is automated, there are several user decisions required to configure the model generation process. The user must select a good set of design variables, thresholds for these design variables, and a window size for rate generation. Ideally, the user would select the minimal set of uncorrelated variables required to model the system. Modeling correlated variables increases the complexity of the LHPN model without adding any additional information to the model. Support can be added to the model generator to detect correlated variables and notify the user of this condition. Finding suitable thresholds for each selected design variable can also be difficult. We have begun work to automatically detect suitable thresholds given a user desired number of thresholds for each selected design variable. This method uses an optimization algorithm to search for the optimal set of thresholds in which the ranges on the rates are minimized. However, improved optimization algorithms and cost functions are needed to determine the most effective method for generating good thresholds. Finally, setting the window size for rate generation affects both the rates and modeled regions. A window size should be large enough to smooth out transitory pulses and other waveform artifacts, but it should be small enough to include regions that are traversed quickly. This requirement may be difficult to meet and modifying the model generation algorithm to employ a variable window size may prove useful to address this problem.

A good abstract model only captures the behavior of the unabstracted model necessary for the given application of the model. This concept is often discussed from the standpoint of including the proper behaviors. For AMS circuits, there is a second measure of abstraction which is the accuracy of the model. `LEMA`'s model generator can be modified to provide a user configurable model accuracy. This adjustable accuracy could be achieved through simplification of the rates and delays or by safe transformations of the model that reduce the number of rates or delays through the combination of similar quantities.

One difficult issue facing the model generation community is the quantification of model quality. For some of the abstraction methods for LTI systems, explicit error bounds for the abstracted model can be automatically derived. For the majority of abstraction methods there are no easily derivable error bounds. This results in model quality comparisons like the one presented in Figure 6.16 where a simulation of the abstracted and unabstracted model are compared qualitatively. This method of quality comparison negates much of the benefit of the abstract model because both the abstracted and unabstracted model must be simulated for every new use of the abstract model to accurately understand the model behavior. There are several potential directions for this work, but initially, the most fruitful direction may be to develop a standardized set of simulations to characterize a model for a given circuit type or topology similar to the method proposed in the Model QA Specification by the Compact Model Council [108].

7.2.2 Coverage Metrics for AMS Circuits

The coverage metric proposed in Section 3.4 gives guidance as to whether a recently added simulation provides additional information to the model generator, but it does not suggest if further simulations should be done or which simulations should be done. More sophisticated coverage metrics are needed to serve this purpose. Looking to digital coverage metrics for inspiration has not proved to be a profitable approach. Most notions of coverage for digital systems assume that each input combination may produce a unique and interesting output combination or are related to the RTL code [150]. This digital assumption does not hold for analog circuits whose output is often linearly related to the input. AMS designers also spend significant effort characterizing their designs for potential sources of variation. Coverage metrics for analog circuits need to account for these characteristics of analog design and then quantify the value of a set of simulations to explore the operating range of the circuit as well as the variation space. These metrics could then be used to suggest or even automatically run simulations to improve the model. For example, if a region is not visited during the current set of simulations, a new simulation can be run that has initial values to place it within the previously unexplored region.

Another approach to coverage metrics for AMS circuits would be to develop specialized coverage metrics for specific types of circuits. The number of building blocks for analog circuits is relatively small. It is reasonable to believe that a coverage metric could be developed for each of the major building blocks. This metric could be tweaked based on the topology of the specific circuit as well as the specific semiconductor fabrication process being used. Development of these types of metrics would a be a nice first step toward useful analog coverage metrics. These coverage metrics could then be analyzed for common elements in an effort to build a generalized AMS coverage metric.

7.2.3 Counter-Example Guided Abstraction-Refinement

Another way to help improve the quality of the model generation by leveraging information from the verification engine would be to use a counter-example guided abstraction-refinement methodology. When the verification engine returns an invalid error trace, the trace could be analyzed to determine the continuous variable and its value that creates the error. New thresholds could then be added to this variable around this value refining the model. More sophisticated approaches would likely also consider adding more continuous variables to the model or adjusting the window size used for rate generation. This abstraction-refinement loop could then be continued until a valid error trace is found or an adequately abstract model is verified.

7.2.4 False Negative Detection

Automatically determining if an error trace produced by `LEMA` is an actual error or a false negative is important to increasing the utility of `LEMA` by designers. This is difficult to do because `LEMA` explores the design space over ranges of parameters and initial conditions while current SPICE simulators only support exploring traces for a single set of initial conditions and parameters. This restriction by current SPICE simulators often makes it impossible to correlate `LEMA`'s error trace back to the original model.

Recently, Tiwary et al. proposed an interval-based SPICE simulator, `iSpice` [152]. `iSpice` provides support for transient simulation over intervals for initial conditions and parameters. This simulator provides a promising framework to check `LEMA`'s error traces for false negative results. `iSpice` is not publicly available, but integrating `iSpice` with `LEMA` should be explored.

7.2.5 AMS Property Specification Language

One key to verification is providing the verification engineer with a property specification language expressive enough to capture the properties that need to be verified yet simple enough to be easily learned and used. There are several property specification languages in use for digital circuits such as PSL and SVA. These languages are not well suited to AMS verification because AMS circuit verification involves continuous quantities as well as frequency domain analysis.

The two primary efforts to develop an AMS circuit property specification language are the development of Ana CTL [46] and STL/PSL [121]. Ana CTL provides operators for capturing constant signals, delta changes between signals, finite values for a signal, variables related by a given function, and variables related by a given waveform. STL/PSL provides operators for capturing signals that are time shifted and relation of signals and constants using $+$, $-$, and $\times$. It has been shown that these types of languages are more useful than a standard digital property specification language, but they are still not able to verify all necessary types of analog properties [87]. Therefore, extensions like those suggested in [87] as well as the ability to verify properties in the frequency domain need to be made to produce an industrial grade AMS property specification language.

7.2.6 Assertion Based Verification for AMS Circuits

Assertion based verification (ABV) has become popular for digital circuits. This technology can also be applied to AMS circuits with low overhead. In fact, there are already industrial tools in this direction such as Mentor's Eldo simulator which provides user configurable monitoring to detect potential reliability failures. These features show that ABV for AMS circuits should be capable of checking for functional failures as well as reliability failures.

There are several bits of technology needed to provide an ABV infrastructure for AMS circuits. One of the items is a property specification language for AMS circuits as discussed in Section 7.2.5. Another item is simulator support for online detection of assertion violations. There has been some work in this area [121], but further work is needed. Finally, language support is needed for Verilog-AMS. VHDL-AMS provides a facility to write assertions, but Verilog-AMS does not.

7.2.7 Automatic Stability Verification

Stability is a critical property for many complex AMS circuits such as PLLs and oscillators. It is very difficult to verify the stability of these circuits using simulation due to the number of simulations required to characterize their behavior over large ranges of initial conditions and for long transient simulation times. The stability of these circuits can be analyzed using phase portraits [94]. It is feasible to create an accurate model of the circuit and then do state space exploration in the phase domain to prove the stability or instability of these types of circuits [17, 18].

7.2.8 Embedded Software Verification

Another prominent form of hybrid systems are embedded systems, systems that contain both AMS circuits and software. Preliminary efforts have shown that LHPNs can be extended to model software at the assembly language level. Using LHPNs, properties could be verified about the entire state space of embedded system. Further work is needed in the mechanisms required to translate and abstract software to LHPNs.

Another method to verify embedded systems would be to develop compositional verification techniques. The verification process would take verification results from the AMS circuits using LHPNs and embedded software using current software verification methodologies. After defining the software/hardware interface, the verification results from the software and the hardware would be combined to verify the entire system. It is possible that due to the low-level nature of embedded software some of the current software verification methods would need to be adapted to handle assembly language software constructs.

REFERENCES

[1] Accellera. *Property Specification Language Reference Manual*, 1.1 ed., June 2004.

[2] Accellera. *Verilog-AMS Language Reference Manual: Analog & Mixed-Signal Extensions to Verilog HDL*, version 2.2 ed., Nov. 2004.

[3] Al-Sammane, G., Zaki, M. H., Dong, Z. J., and Tahar, S. Towards assertion based verification of analog and mixed signal designs using PSL. In *Proc. Languages for Formal Specification and Verification, Forum on Specification and Design Languages (FDL)* (2007), pp. 293–298.

[4] Al-Sammane, G., Zaki, M. H., and Tahar, S. A symbolic methodology for the verification of analog and mixed signal designs. In *Proc. Design, Automation and Test in Europe (DATE)* (2007), R. Lauwereins and J. Madsen, Eds., ACM Press, pp. 249–254.

[5] Alur, R. Timed automata. In *Proc. International Conference on Computer Aided Verification (CAV)* (1999), N. Halbwachs and D. Peled, Eds., vol. 1633 of *Lecture Notes in Computer Science*, Springer, pp. 8–22.

[6] Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138*, 1 (1995), 3–34.

[7] Alur, R., Courcoubetis, C., Henzinger, T. A., and Ho, P.-H. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems* (1992), R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds., vol. 736 of *Lecture Notes in Computer Science*, Springer, pp. 209–229.

[8] Alur, R., and Dill, D. L. A theory of timed automata. *Theoretical Computer Science 126*, 2 (1994), 183–235.

[9] Alur, R., Grosu, R., Hur, Y., Kumar, V., and Lee, I. Modular specification of hybrid systems in CHARON. In *Hybrid Systems: Computation and Control (HSCC)* (2000), N. A. Lynch and B. H. Krogh, Eds., vol. 1790 of *Lecture Notes in Computer Science*, Springer, pp. 6–19.

[10] Alur, R., Henzinger, T. A., and Ho, P.-H. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering 22*, 3 (1996), 181–201.

[11] Annichini, A., Asarin, E., and Bouajjani, A. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. International Conference on Computer Aided Verification (CAV)* (2000), E. A. Emerson and A. P. Sistla, Eds., vol. 1855 of *Lecture Notes in Computer Science*, pp. 419–434.

[12] Annichini, A., Bouajjani, A., and Sighireanu, M. TReX: A tool for reachability analysis of complex systems. In *Proc. International Conference on Computer Aided Verification (CAV)* (2001), G. Berry, H. Comon, and A. Finkel, Eds., vol. 2102 of *Lecture Notes in Computer Science*, Springer, pp. 368–372.

[13] Antoulas, A., Sorensen, D., and Gugercin, S. A survey of model reduction methods for large-scale systems. *Contemporary Mathematics 280* (2001), 193–219.

[14] Asarin, E., Bournez, O., Dang, T., and Maler, O. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control (HSCC)* (2000), N. A. Lynch and B. H. Krogh, Eds., vol. 1790 of *Lecture Notes in Computer Science*, Springer, pp. 20–31.

[15] Asarin, E., Dang, T., and Girard, A. Reachability analysis of nonlinear systems using conservative approximation. In *Hybrid Systems: Computation and Control (HSCC)* (2003), O. Maler and A. Pnueli, Eds., vol. 2623 of *Lecture Notes in Computer Science*, Springer, pp. 20–35.

[16] Asarin, E., Dang, T., and Maler, O. The d/dt tool for verification of hybrid systems. In *Proc. International Conference on Computer Aided Verification (CAV)* (2002), E. Brinksma and K. G. Larsen, Eds., vol. 2404 of *Lecture Notes in Computer Science*, Springer, pp. 365–370.

[17] Asarin, E., Schneider, G., and Yovine, S. On the decidability of the reachability problem for planar differential inclusions. In *Hybrid Systems: Computation and Control (HSCC)* (Rome, Italy, 2001), M. di Benedetto and A. Sangiovanni-Vincentelli, Eds., no. 2034 in Lecture Notes in Computer Science, Springer, pp. 89–104.

[18] Asarin, E., Schneider, G., and Yovine, S. Towards computing phase portraits of polygonal differential inclusions. In *Hybrid Systems: Computation and Control (HSCC)* (Stanford, USA, 2002), C. J. Tomlin and M. R. Greenstreet, Eds., no. 2289 in Lecture Notes in Computer Science, Springer, pp. 49–61.

[19] Aziz, P. M., Sorensen, H. V., and Spiegel, J. V. D. An overview of sigma-delta converters. *IEEE Signal Processing Magazine 13*, 1 (Jan. 1996), 61–84.

[20] Bagnara, R., Ricci, E., Zaffanella, E., and Hill, P. M. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *Proc. International Symposium on Static Analysis (SAS)* (Madrid, Spain, 2002), M. V. Hermenegildo and G. Puebla, Eds., vol. 2477 of *Lecture Notes in Computer Science*, Springer, pp. 213–229.

[21] Bai, Z., Feldmann, P., and Freund, R. W. How to make theoretically passive reduced-order models passive in practice. In *Proc. Custom Integrated Circuit Conference (CICC)* (1998), IEEE Press, pp. 207–210.

[22] Baird, R. T., and Fiez, T. S. Stability analysis of high-order delta-sigma modulation for ADC's. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 41*, 1 (Jan. 1994), 59–62.

[23] Balduzzi, F., Giua, A., and Menga, G. First-order hybrid petri nets: A model for optimization and control. *IEEE Transactions on Robotics and Automation 16*, 4 (Aug. 2000), 382–399.

[24] Balivada, A., Hoskote, Y., and Abraham, J. A. Verification of transient response of linear analog circuits. *Proc. IEEE VLSI Test Symposium* (1995), 42–47.

[25] Belluomini, W., and Myers, C. J. Timed state space exploration using POSETS. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 19*, 5 (May 2000), 501–520.

[26] Belluomini, W., Myers, C. J., and Hofstee, H. P. Timed circuit verification using TEL structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20*, 1 (Jan. 2001), 129–146.

[27] Berthomieu, B., and Diaz, M. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering 17*, 3 (1991), 259–273.

[28] Berthomieu, B., and Vernadat, F. Time petri nets analysis with TINA. In *Proc. International Conference on Quantitative Evaluation of Systems (QEST)* (2006), pp. 123–124.

[29] Boser, B. E., and Wooley, B. A. The design of sigma-delta modulation analog-to-digital converters. *IEEE Journal of Solid-State Circuits 23*, 6 (Dec. 1988), 1298–1308.

[30] Botchkarev, O., and Tripakis, S. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *Hybrid Systems: Computation and Control (HSCC)* (2000), N. A. Lynch and B. H. Krogh, Eds., vol. 1790 of *Lecture Notes in Computer Science*, Springer, pp. 73–88.

[31] Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers 35*, 8 (1986), 677–691.

[32] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. Symbolic model checking: 10^{20} states and beyond. In *IEEE Symposium on Logic in Computer Science* (June 1990), IEEE Computer Society Press, pp. 428–439.

[33] Chang, H., and Kundert, K. Verification of complex analog and RF IC designs. *Proceedings of the IEEE 95*, 3 (Mar. 2007), 622–639.

[34] Chen, H., and Hanisch, H.-M. Analysis of hybrid systems based on hybrid net condition/event system model. *Discrete Event Dynamic Systems: Theory and Applications 11*, 1–2 (Jan. 2001), 163–185.

[35] Chen, Y., and White, J. A quadratic method for nonlinear model order reduction. In *Modeling and Simulation of Microsystems, Semiconductors, Sensors, and Actuators* (2000).

[36] Chiola, G., Marsan, M. A., Balbo, G., and Conte, G. Generalized stochastic petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering 19*, 2 (Feb. 1993), 89–107.

[37] Chutinan, A., and Krogh, B. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control 48*, 1 (Jan. 2003), 64–75.

[38] Chutinan, A., and Krogh, B. H. Computing polyhedral approximations to flow pipes for dynamic systems. In *Proc. IEEE Conference on Decision and Control (CDC)* (1998), vol. 2, pp. 2089–2094.

[39] Chutinan, A., and Krogh, B. H. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control (HSCC)* (1999), F. W. Vaandrager and J. H. van Schuppen, Eds., vol. 1569 of *Lecture Notes in Computer Science*, Springer, pp. 76–90.

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

[41] Clarke, E. M., and Kurshan, R. P. Computer-aided verification. *IEEE Spectrum 33*, 6 (June 1996), 61–67.

[42] Cury, J. E., Krogh, B. H., and Niinomi, T. Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Transactions on Automatic Control 43*, 4 (Apr. 1998), 564–568.

[43] Dang, T., Donzé, A., and Maler, O. Verification of analog and mixed-signal circuits using hybrid systems techniques. In *Formal Methods for Computer Aided Design (FMCAD)* (2004), A. J. Hu and A. K. Martin, Eds., vol. 3312 of *Lecture Notes in Computer Science*, Springer, pp. 21–36.

[44] Dang, T., and Maler, O. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control (HSCC)* (1998), T. A. Henzinger and S. Sastry, Eds., vol. 1386 of *Lecture Notes in Computer Science*, Springer, pp. 96–109.

[45] Daniel, L., Siong, O. C., Chay, L. S., Lee, K. H., and White, J. A multiparameter moment matching model reduction approach for generating geometrically parameterized interconnect performance models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23*, 5 (May 2004), 678–693.

[46] Dastidar, T. R., and Chakrabarti, P. P. A verification system for transient response of analog circuits using model checking. In *VLSI Design (VLSID)* (2005), IEEE Computer Society Press, pp. 195–200.

[47] David, R., and Alla, H. Autonomous and timed continuous petri nets. In *Applications and Theory of Petri Nets* (1991), G. Rozenberg, Ed., vol. 674, Springer, pp. 71–90.

[48] David, R., and Alla, H. On hybrid petri nets. *Discrete Event Dynamic Systems: Theory and Applications 11*, 1–2 (Jan. 2001), 9–40.

[49] Davis, M., Logemann, G., and Loveland, D. A machine program for theorem-proving. *Communications of the ACM 5*, 7 (July 1962), 394–397.

[50] Davis, M., and Putnam, H. A computing procedure for quantification theory. *Journal of the ACM (JACM) 7*, 3 (July 1960), 201–215.

[51] Dill, D. L. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Automatic Verification Methods for Finite-State Systems* (1989), J. Sifakis, Ed., vol. 407 of *Lecture Notes in Computer Science*, Springer, pp. 197–212.

[52] Dong, N., and Roychowdhury, J. Piecewise polynomial nonlinear model reduction. In *Proc. Design Automation Conference (DAC)* (New York, NY, USA, 2003), ACM Press, pp. 484–489.

[53] Donzé, A., and Maler, O. Systematic simulation using sensitivity analysis. In *Hybrid Systems: Computation and Control (HSCC)* (2007), A. Bemporad, A. Bicchi, and G. Buttazzo, Eds., vol. 4416 of *Lecture Notes in Computer Science*, Springer, pp. 174–189.

[54] Fainekos, G. E., Girard, A., and Pappas, G. J. Temporal logic verification using simulation. In *Formal Modelling and Analysis of Timed Systems (FORMATS)* (2006), E. Asarin and P. Bouyer, Eds., vol. 4202 of *Lecture Notes in Computer Science*, Springer, pp. 171–186.

[55] Frehse, G. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *Hybrid Systems: Computation and Control (HSCC)* (2005), M. Morari and L. Thiele, Eds., vol. 3414 of *Lecture Notes in Computer Science*, Springer, pp. 258–273.

[56] Frehse, G., Krogh, B. H., and Rutenbar, R. A. Verifying analog oscillator circuits using forward/backward refinement. In *Proc. Design, Automation and Test in Europe (DATE)* (2006), IEEE Computer Society Press, pp. 257–262.

[57] Frehse, G., Krogh, B. H., Rutenbar, R. A., and Maler, O. Time domain verification of oscillator circuit properties. *Electronic Notes Theoretical Computer Science 153*, 3 (2006), 9–22.

[58] Frehse, G., and Maler, O. Reachability analysis of a switched buffer network. In *Hybrid Systems: Computation and Control (HSCC)* (2007), A. Bemporad, A. Bicchi, and G. C. Buttazzo, Eds., vol. 4416 of *Lecture Notes in Computer Science*, Springer, pp. 698–701.

[59] Freund, R. W. Passive reduced-order models for interconnect simulation and their computation via krylov-subspace algorithms. In *Proc. Design Automation Conference (DAC)* (1999), pp. 195–200.

[60] Ghosh, A., and Vemuri, R. Formal verification of synthesized analog designs. In *Proc. International Conference on Computer Design (ICCD)* (1999), IEEE Computer Society Press, pp. 40–45.

[61] Girard, A. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control (HSCC)* (2005), M. Morari and L. Thiele, Eds., vol. 3414 of *Lecture Notes in Computer Science*, Springer, pp. 291–305.

[62] Girard, A., and Pappas, G. J. Approximate bisimulations for constrained linear systems. In *IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)* (2005), pp. 4700–4705.

[63] Girard, A., and Pappas, G. J. Approximate bisimulations for nonlinear dynamical systems. In *IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)* (2005), pp. 684–689.

[64] Girard, A., and Pappas, G. J. Verification using simulation. In *Hybrid Systems: Computation and Control (HSCC)* (2006), J. P. Hespanha and A. Tiwari, Eds., vol. 3927 of *Lecture Notes in Computer Science*, Springer, pp. 272–286.

[65] Greenstreet, M. R. Verifying safety properties of differential equations. In *Proc. International Conference on Computer Aided Verification (CAV)* (July 1996), R. Alur and T. A. Henzinger, Eds., vol. 1102 of *Lecture Notes in Computer Science*, Springer, pp. 277–287.

[66] Greenstreet, M. R., and Mitchell, I. Integrating projections. In *Hybrid Systems: Computation and Control (HSCC)* (1998), T. A. Henzinger and S. Sastry, Eds., vol. 1386 of *Lecture Notes in Computer Science*, Springer, pp. 159–174.

[67] Greenstreet, M. R., and Mitchell, I. Reachability analysis using polygonal projections. In *Hybrid Systems: Computation and Control (HSCC)* (1999), F. W. Vaandrager and J. H. van Schuppen, Eds., vol. 1569 of *Lecture Notes in Computer Science*, Springer, pp. 103–116.

[68] Grimme, E. J. *Krylov Projection Methods for Model Reduction.* PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[69] Grözing, M., Phillip, B., and Berroth, M. CMOS ring oscillator with quadrature outputs and 100 MHz to 3.5 GHz tuning range. In *Proc. European Solid-State Circuits Conference* (2003), pp. 679–682.

[70] Gugercin, S., and Antoulas, A. C. A survey of model reduction by balanced truncation and some new results. *International Journal of Control 77*, 8 (2004), 748–766.

[71] Gupta, S., Krogh, B. H., and Rutenbar, R. A. Towards formal verification of analog designs. In *Proc. International Conference on Computer Aided Design (ICCAD)* (2004), IEEE Computer Society Press, pp. 210–217.

[72] Hanna, K. Reasoning about real circuits. In *Proc. Higher Order Logic Theorem Proving and Its Applications (TPHOLs)* (1994), T. F. Melham and J. Camilleri, Eds., vol. 859 of *Lecture Notes in Computer Science*, Springer, pp. 235–253.

[73] Hanna, K. Automatic verification of mixed-level logic circuits. In *Formal Methods for Computer Aided Design (FMCAD)* (1998), G. Gopalakrishnan and P. J. Windley, Eds., vol. 1522 of *Lecture Notes in Computer Science*, Springer, pp. 133–166.

[74] Hartong, W., Hedrich, L., and Barke, E. Model checking algorithms for analog verification. In *Proc. Design Automation Conference (DAC)* (2002), ACM Press, pp. 542–547.

[75] Hartong, W., Hedrich, L., and Barke, E. On discrete modeling and model checking for nonlinear analog systems. In *Proc. International Conference on Computer Aided Verification (CAV)* (2002), E. Brinksma and K. G. Larsen, Eds., vol. 2404 of *Lecture Notes in Computer Science*, Springer, pp. 401–413.

[76] Hartong, W., Klausen, R., and Hedrich, L. *Advanced Formal Verification.* Springer, 2004, ch. Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking, pp. 205–245.

[77] Hedrich, L., and Barke, E. A formal approach to nonlinear analog circuit verification. In *Proc. International Conference on Computer Aided Design (ICCAD)* (Nov. 1995), IEEE Computer Society Press, pp. 123–127.

[78] Hedrich, L., and Barke, E. A formal approach to verification of linear analog circuits with parameter tolerances. In *Proc. Design, Automation and Test in Europe (DATE)* (Feb. 1998), IEEE Computer Society Press, pp. 649–654.

[79] Hein, S., and Zakhor, A. On the stabililty of sigma delta modulators. *IEEE Transactions on Signal Processing 41*, 7 (July 1993), 2322–2348.

[80] Hendricx, S., and Claesen, L. A symbolic core approach to the formal verification of integrated mixed-mode applications. In *Proc. European Design and Test Conference* (1997), IEEE Computer Society Press, pp. 432–436.

[81] Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer 1*, 1–2 (1997), 110–122.

[82] Henzinger, T. A., Kopke, P. W., Puri, A., and Varaiya, P. What's decidable about hybrid automata? *Journal of Computer and System Sciences 57*, 1 (1998), 94–124.

[83] Horton, G., Kulkarni, V. G., Nicol, D. M., and Trivedi, K. S. Fluid stochastic petri nets: Theory, applications, and solution techniques. *European Journal of Operational Research 105*, 1 (1998), 184–201.

[84] IEEE Computer Society. *IEEE Standard for Verilog Hardware Description Language (1364-2005)*, Apr. 2006.

[85] IEEE-SA Standards Board. *IEEE Standard VHDL Analog and Mixed-Signal Extensions*, Mar. 1999.

[86] IEEE-SA Standards Board. *IEEE Standard VHDL Language Reference Manual*, Jan. 2000.

[87] Jones, K. D., Kim, J., and Konrad, V. Some "real world" problems in the analog and mixed signal domains. In *Proc. Designing Correct Circuits* (2008).

[88] Jones, K. D., Konrad, V., and Nickovic, D. Analog property checkers: A DDR2 case study. In *Formal Verification of Analog Circuits (FAC)* (2008).

[89] Julius, A. A., Fainekos, G. E., Anand, M., Lee, I., and Pappas, G. J. Robust test generation and coverage for hybrid systems. In *Hybrid Systems: Computation and Control (HSCC)* (2007), A. Bemporad, A. Bicchi, and G. C. Buttazzo, Eds., vol. 4416 of *Lecture Notes in Computer Science*, Springer, pp. 329–342.

[90] Kamon, M., Wang, F., and White, J. Generating nearly optimally compact models from krylov-subspace based reduced-order models. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 47*, 4 (Apr. 2000), 239–248.

[91] Kapinski, J., Krogh, B. H., Maler, O., and Stursberg, O. On systematic simulation of open continuous systems. In *Hybrid Systems: Computation and Control (HSCC)* (2003), O. Maler and A. Pnueli, Eds., vol. 2623 of *Lecture Notes in Computer Science*, Springer, pp. 283–297.

[92] Kaufmann, M., Manolios, P., and Moore, J. S. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, June 2000.

[93] Kern, C., and Greenstreet, M. R. Formal verification in hardware design: A survey. *ACM Transactions on Design Automation of Electronic Systems 4*, 2 (Apr. 1999), 123–193.

[94] Kourjanski, M., and Varaiya, P. Stability of hybrid systems. In *Hybrid Systems: Computation and Control (HSCC)* (1995), R. Alur, T. A. Henzinger, and E. D. Sontag, Eds., vol. 1066 of *Lecture Notes in Computer Science*, Springer, pp. 413–423.

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

[96] Kurshan, R. P., and McMillan, K. L. Analysis of digital circuits through symbolic reduction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 10*, 11 (Nov. 1991), 1356–1371.

[97] Kurzhanski, A. B., and Varaiya, P. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems: Computation and Control (HSCC)* (2000), N. A. Lynch and B. H. Krogh, Eds., vol. 1790 of *Lecture Notes in Computer Science*, Springer, pp. 202–214.

[98] Lai, X., and Roychowdhury, J. TP-PPV: Piecewise nonlinear, time-shifted oscillator macromodel extraction for fast, accurate PLL simulation. In *Proc. International Conference on Computer Aided Design (ICCAD)* (2006), S. Hassoun, Ed., ACM Press, pp. 269–274.

[99] Li, P., and Pileggi, L. T. NORM: compact model order reduction of weakly nonlinear systems. In *Proc. Design Automation Conference (DAC)* (2003), pp. 472–477.

[100] Little, S., and Myers, C. Abstract modeling and simulation aided verification of analog/mixed-signal circuits. In *Formal Verification of Analog Circuits (FAC)* (2008).

[101] Little, S., Seegmiller, N., Walter, D., Myers, C., and Yoneda, T. Verification of analog/mixed-signal circuits using labeled hybrid petri nets. In *Proc. International Conference on Computer Aided Design (ICCAD)* (2006), IEEE Computer Society Press, pp. 275–282.

[102] Little, S., Sen, A., and Myers, C. Application of automated model generation techniques to analog/mixed-signal circuits. In *International Workshop on Microprocessor Test and Verification* (Dec. 2007).

[103] Little, S., Walter, D., and Myers, C. Analog/mixed-signal circuit verification using models generated from simulation traces. In *Automated Technology for Verification and Analysis (ATVA)* (2007), K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, Eds., vol. 4762 of *Lecture Notes in Computer Science*, Springer, pp. 114–128.

[104] Little, S., Walter, D., Seegmiller, N., Myers, C., and Yoneda, T. Verification of analog and mixed-signal circuits using timed hybrid petri nets. In *Automated Technology for Verification and Analysis (ATVA)* (Nov. 2004), vol. 3299 of *Lecture Notes in Computer Science*, Springer, pp. 426–440.

[105] Maler, O., and Nickovic, D. Monitoring temporal properties of continuous signals. In *Formal Modelling and Analysis of Timed Systems (FORMATS)* (2004), Y. Laknech and S. Yovine, Eds., vol. 3253 of *Lecture Notes in Computer Science*, Springer, pp. 152–166.

[106] Maler, O., Nickovic, D., and Pnueli, A. From MITL to timed automata. In *Formal Modelling and Analysis of Timed Systems (FORMATS)* (2006), E. Asarin and P. Bouyer, Eds., vol. 4202 of *Lecture Notes in Computer Science*, Springer, pp. 274–289.

[107] Maler, O., Nickovic, D., and Pnueli, A. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of Computer Science* (2008), A. Avron, N. Dershowitz, and A. Rabinovich, Eds., vol. 4800 of *Lecture Notes in Computer Science*, Springer, pp. 475–505.

[108] McAndrew, C. C. *CMC Compact Model QA Specification*, 1.2 ed. Compact Model Council, 2006.

[109] Merlin, P. M., and Farber, D. J. Recoverability of communication protocols. *IEEE Transactions on Communications 24*, 9 (Sept. 1976), 1036–1043.

[110] Metropolis, N., and Ulam, S. The monte carlo method. *Journal of the American Statistical Association 44*, 247 (Sept. 1949), 335–341.

[111] Miné, A. The octagon abstract domain. In *Analyzing, Slicing, and Transformation* (Oct. 2001), IEEE Computer Society Press, pp. 310–319.

[112] Miné, A. *Weakly Relational Numerical Abstract Domains.* PhD thesis, École Normale Supérieure de Paris, France, 2004.

[113] Mitchell, I., and Tomlin, C. Level set methods for computation in hybrid systems. In *Hybrid Systems: Computation and Control (HSCC)* (2000), N. A.

Lynch and B. H. Krogh, Eds., vol. 1790 of *Lecture Notes in Computer Science*, Springer, pp. 310–323.

[114] Myers, C. *Asynchronous Circuit Design.* Wiley, 2001.

[115] Myers, C. J., Belluomini, W., Killpack, K., Mercer, E., Peskin, E., and Zheng, H. Timed circuits: A new paradigm for high-speed design. In *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC)* (Feb. 2001), ACM Press, pp. 335–340.

[116] Myers, C. J., Harrison, R. R., Walter, D., Seegmiller, N., and Little, S. The case for analog circuit verification. *Electronic Notes Theoretical Computer Science 153*, 3 (2006), 53–63.

[117] Nagel, L. *SPICE2: A Computer Program to Simulate Semiconductor Circuits.* PhD thesis, University of California Berkeley, 1975.

[118] Nagel, L., and Rohrer, R. Computer analysis of nonlinear circuits, excluding radiation (cancer). *IEEE Journal of Solid-State Circuits 6*, 4 (Aug. 1971), 166–182.

[119] Nahhal, T., and Dang, T. Guided randomized simulation. In *Hybrid Systems: Computation and Control (HSCC)* (2007), A. Bemporad, A. Bicchi, and G. C. Buttazzo, Eds., vol. 4416 of *Lecture Notes in Computer Science*, Springer, pp. 731–735.

[120] Nahhal, T., and Dang, T. Test coverage for continuous and hybrid systems. In *Proc. International Conference on Computer Aided Verification (CAV)* (2007), W. Damm and H. Hermanns, Eds., vol. 4590 of *Lecture Notes in Computer Science*, Springer, pp. 449–462.

[121] Nickovic, D., and Maler, O. AMT: A property-based monitoring tool for analog systems. In *Formal Modelling and Analysis of Timed Systems (FORMATS)* (2007).

[122] Nieuwenhuis, R., Oliveras, A., and Tinelli, C. Solving SAT and SAT modulo theories: from an abstract davis-putnam-logemann-loveland procedure to DPLL(T). *Journal of the ACM (JACM) 53*, 6 (Nov. 2006), 937–977.

[123] Odabasioglu, A., Celik, M., and Pileggi, L. T. PRIMA: passive reduced-order interconnect macromodeling algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17*, 8 (1998), 645–654.

[124] Petri, C. A. Communication with automata. Tech. Rep. RADC-TR-65-377, Vol. 1, Suppl 1, Applied Data Research, Princeton, NJ, 1966.

[125] Pettersson, P., and Larsen., K. G. Uppaal2k. *Bulletin of the European Association for Theoretical Computer Science 70* (Feb. 2000), 40–44.

[126] Phillips, J. R. Model reduction of time-varying linear systems using approximate multipoint krylov-subspace projectors. In *Proc. International Conference on Computer Aided Design (ICCAD)* (1998), ACM Press, pp. 96–102.

[127] Phillips, J. R. Projection frameworks for model reduction of weakly nonlinear systems. In *Proc. Design Automation Conference (DAC)* (2000), pp. 184–189.

[128] Phillips, J. R. Projection-based approaches for model reduction of weakly nonlinear, time-varying systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22*, 2 (2003), 171–187.

[129] Phillips, J. R., and Silveira, L. M. Poor man's TBR: a simple model reduction scheme. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24*, 1 (2005), 43–55.

[130] Pillage, L. T., and Rohrer, R. A. Asymptotic waveform evaluation for timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 9*, 4 (1990), 352–366.

[131] Prasad, M. R., Biere, A., and Gupta, A. A survey of recent advances in SAT-based formal verification. *International Journal on Software Tools for Technology Transfer 7*, 2 (Apr. 2005), 156–173.

[132] Quarles, T. L. *Analysis of Performance and Convergence Issues for Circuit Simulation.* PhD thesis, University of California, Berkeley, Apr. 1989.

[133] Ray, S., and Bhadra, J. A mechanized refinement framework for analysis of custom memories. In *Formal Methods for Computer Aided Design (FMCAD)* (2007), IEEE Computer Society Press, pp. 239–242.

[134] Rewieński, M., and White, J. A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22*, 2 (Feb. 2003), 155–170.

[135] Rewieński, M., and White, J. Model order reduction for nonlinear dynamical systems based on trajectory piecewise-linear approximations. *Linear Algebra and its Applications 415*, 2–3 (jun 2006), 426–454.

[136] Rewieński, M. J. *A Trajectory Piecewise-Linear Approach to Model Order Reduction of Nonlinear Dynamical Systems.* PhD thesis, Massachusetts Institute of Technology, June 2003.

[137] Rokicki, T. *Representing and Modeling Digital Circuits.* PhD thesis, Stanford University, Dec. 1993.

[138] Roychowdhury, J. MPDE methods for efficient analysis of wireless systems. In *Proc. Custom Integrated Circuit Conference (CICC)* (1998), IEEE Press, pp. 451–454.

[139] Roychowdhury, J. Reduced-order modeling of time-varying systems. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 46*, 10 (Oct. 1999), 1273–1288.

[140] Rutenbar, R. A., Gielen, G. G. E., and Roychowdhury, J. Hierarchical modeling, optimization, and synthesis for system-level analog and RF designs. *Proceedings of the IEEE 95*, 3 (Mar. 2007), 640–669.

[141] Salem, A. Semi-formal verification of VHDL-AMS descriptions. In *Proc. International Symposium on Circuits and Systems (ISCAS)* (2002), vol. 5, IEEE Press, pp. 333–336.

[142] Seshadri, S., and Abraham, J. A. Frequency response verification of analog circuits using global optimization techniques. *Journal of Electronic Testing 17*, 5 (Oct. 2001), 395–408.

[143] Sha, Y.-B., Lee, M.-S., and Liu, C.-N. J. On code coverage measurement for Verilog-A. In *Proc. High-Level Design Validation and Test Workshop* (2004), IEEE Press, pp. 115–120.

[144] Silva, B. I., and Krogh, B. H. Formal verification of hybrid systems using CheckMate: A case study. In *Proc. American Control Conference* (June 2000), vol. 3, IEEE Press, pp. 1679–1683.

[145] Silva, B. I., Richeson, K., Krogh, B., and Chutinan, A. Modeling and verifying hybrid dynamic systems using CheckMate. In *Automation of Mixed Processes: Hybrid Dynamic Systems* (Sept. 2000).

[146] Silva, B. I., Stursberg, O., Krogh, B. H., and Engell, S. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *Proc. IEEE Conference on Decision and Control (CDC)* (Dec. 2003), vol. 3, IEEE Press, pp. 2867–2874.

[147] Sou, K. C., Megretski, A., and Daniel, L. A quasi-convex optimization approach to parameterized model order reduction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27*, 3 (Mar. 2008), 456–469.

[148] Sperling, E. Why is analog so difficult? *DACeZine 3*, 5 (Jan. 2008).

[149] Stursberg, O., and Krogh, B. H. Efficient representation and computation of reachable sets for hybrid systems. In *Hybrid Systems: Computation and Control (HSCC)* (2003), O. Maler and A. Pnueli, Eds., vol. 2623 of *Lecture Notes in Computer Science*, Springer, pp. 482–497.

[150] Tasiran, S., and Keutzer, K. Coverage metrics for functional validation of hardware designs. *IEEE Design & Test of Computers 18*, 4 (2001), 36–45.

[151] Tiwari, A. Approximate reachability for linear systems. In *Hybrid Systems: Computation and Control (HSCC)* (Apr. 2003), O. Maler and A. Pnueli, Eds., vol. 2623 of *Lecture Notes in Computer Science*, Springer, pp. 514–525.

[152] Tiwary, S. K., Gupta, A., Phillips, J. R., Pinello, C., and Zlatanovici, R. iSpice: A boolean satisfiability based approach to formally verifying analog circuits. In *Formal Verification of Analog Circuits (FAC)* (2008).

[153] Tiwary, S. K., and Rutenbar, R. A. Scalable trajectory methods for on-demand analog macromodel extraction. In *Proc. Design Automation Conference (DAC)* (2005), W. H. J. Jr., G. Martin, and A. B. Kahng, Eds., ACM Press, pp. 403–408.

[154] Tiwary, S. K., and Rutenbar, R. A. Faster, parametric trajectory-based macromodels via localized linear reductions. In *Proc. International Conference on Computer Aided Design (ICCAD)* (2006), S. Hassoun, Ed., ACM Press, pp. 876–883.

[155] Walter, D., Little, S., and Myers, C. Bounded model checking of analog and mixed-signal circuits using an SMT solver. In *Automated Technology for Verification and Analysis (ATVA)* (2007), K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, Eds., vol. 4762 of *Lecture Notes in Computer Science*, Springer, pp. 66–81.

[156] Walter, D., Little, S., Seegmiller, N., Myers, C. J., and Yoneda, T. Symbolic model checking of analog/mixed-signal circuits. In *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC)* (2007), pp. 316–323.

[157] Walter, D. C. *Verification of analog and mixed-signal circuits using symbolic methods*. PhD thesis, University of Utah, May 2007.

[158] Wang, F. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. *IEEE Transactions on Software Engineering 31*, 1 (Jan. 2005), 38–51.

[159] Wang, Z., Lai, X., and Roychowdhury, J. PV-PPV: Parameter variability aware, automatically extracted, nonlinear time-shifted oscillator macromodels. In *Proc. Design Automation Conference (DAC)* (2007), IEEE Press, pp. 142–147.

[160] Yan, C., and Greenstreet, M. R. Circuit level verification of a high-speed toggle. In *Formal Methods for Computer Aided Design (FMCAD)* (2007), IEEE Press, pp. 199–206.

[161] Yoneda, T. VINAS-P: A tool for trace theoretic verification of timed asynchronous circuits. In *Proc. International Conference on Computer Aided Verification (CAV)* (2000), E. A. Emerson and A. P. Sistla, Eds., vol. 1855 of *Lecture Notes in Computer Science*, Springer, pp. 572–575.

[162] Yovine, S. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer 1*, 1–2 (Oct. 1997), 123–133.

[163] Zaki, M. H., Al-Sammane, G., Tahar, S., and Bois, G. Combining symbolic simulation and interval arithmetic for the verification of AMS designs. In *Formal Methods for Computer Aided Design (FMCAD)* (2007), IEEE Computer Society Press, pp. 207–215.

[164] Zaki, M. H., Tahar, S., and Bois, G. Formal verification of analog and mixed signal designs: Survey and comparison. In *Proc. IEEE Northeast Workshop on Circuits and Systems (NEWCAS)* (2006).

[165] Zaki, M. H., Tahar, S., and Bois, G. A practical approach for monitoring analog circuits. In *ACM Great Lakes Symposium on VLSI (GLS-VLSI)* (2006), pp. 330–335.

[166] Zheng, H. Specification and compilation of timed systems. Master's thesis, University of Utah, 1998.