Formal Verification Methodologies for Nonlinear Analog Circuits

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von

Sebastian Steinhorst

aus Mainz

Frankfurt (2011)
(D 30)

vom Fachbereich Informatik und Mathematik der Johann Wolfgang Goethe-Universität als Dissertation angenommen

Dekan:	Prof. Dr. Tobias Weth
Gutachter:	Prof. Dr. Lars Hedrich
	Priv.-Doz. Dr. Helmut Gräb
Datum der Disputation:	3. Februar 2011

Acknowledgments

This thesis is the result of five years of my research in the Electronic Design Methodology Group at the Institute of Computer Science of the Goethe-University of Frankfurt am Main. It would not have been possible without the support of many people.

First of all, I would like to express my deepest gratitude to my doctoral advisor, Lars Hedrich, for his encouragement, guidance and support from the beginnings of my research until the finalization of this thesis. His keen mind and overwhelming interest in my work have been an invaluable help.

I am especially indebted to Helmut Gräb for fruitful discussions, his continuous interest in this thesis and for acting as a referee.

Moreover, I owe a big thank you to my examiners Georg Schnitger and Uwe Brinkschulte for their interesting questions and valuable remarks.

I am grateful to Angelika Schifignano for all of her assistance, being the nicest and most caring secretary I could imagine.

I would like to thank Oliver Mitea, with whom I shared the office for 4 years, for the great time we had together as colleagues and most of all for becoming a friend. Furthermore, I would like to thank my colleagues Mingyu Ma, Markus Meissner, Julius von Rosen and Felix Salfelder for the great working atmosphere in our group. Thank you also to Ronja Düffel, Andreas Hofmann, Conrad Rau, David Sabel and Manfred Schmidt-Schauß for the good times we had.

Finally, I would like to thank my wonderful wife Christina Steinhorst, my parents Doris and Gerhard Steinhorst, my parents-in-law Melitta and Fred Bayer and grandma Ottilia Gutermann for their patience, continuous support and for allowing me to concentrate on my research work by managing my private affairs.

Abstract

The objective of this thesis is to develop new methodologies for formal verification of nonlinear analog circuits. Therefore, new approaches to discrete modeling of analog circuits, specification of analog circuit properties and formal verification algorithms are introduced.

While the design flow for digital circuits is mostly automated and formalized, the analog design flow still contains several manual steps. This particularly applies to the area of verification, which is the process of systematically assuring specification conformance of all design steps. There are two global classes of verification approaches, represented by conventional non-formal verification and formal verification. Non-formal test bench-based simulation is the state of the art in the area of analog verification in industrial design flows. Due to the experimental character of this approach, critical specification-violating corner-case behavior can remain unreached by the simulation runs and therewith undiscovered by the designer.

Formal approaches to verification of analog circuits are not yet introduced into industrial design flows and still subject to research. Formal verification proves specification conformance for all possible input conditions and all possible internal states of a circuit. Automatically proving that a model of the circuit satisfies a declarative machine-readable property specification is referred to as model checking. Equivalence checking proves the equivalence of two circuit implementations.

Starting from the state of the art in modeling analog circuits for simulation-based verification, discrete modeling of analog circuits for state space-based formal verification methodologies is motivated in this thesis. The up to now most capable approach to discrete modeling partitions the state space into paraxial hyperboxes of homogeneous behavior of the state space dynamics. Due to the paraxial slicing, the non-paraxial vector field dynamics representing the state space dynamics cannot be sufficiently captured. Hence, the hyperbox-approach is not rotation invariant with respect to the structure of the state space dynamics which results in a massive over-approximation of the successor relation of the discrete transition model. In order to improve the discrete modeling of analog circuits, a new trajectory-directed partitioning algorithm was developed in the scope of this thesis. This new approach determines the partitioning of the state space parallel or orthogonal to the trajectories of the state space dynamics.

Therewith, a high accuracy of the successor relation is achieved in combination with a lower number of states necessary for a discrete model of equal accuracy compared to the hyperbox-approach. The mapping of the partitioning to a discrete analog transition structure (DATS) enables the application of formal verification algorithms.

Formal property specification for the initial approaches to model checking of analog circuits was strongly related to temporal logic specification approaches in the digital domain. However, specification of analog properties such as slew rate and oscillation is fundamentally different from digital properties such as fairness and liveness. Additionally, already in the digital domain, specification by using temporal logics such as the Computation Tree Logic (CTL) was considered not to be designer-friendly. Hence, specifying analog properties with CTL cannot be considered as suitable for analog designers that, in general, do not have a background in computer science. By analyzing digital specification concepts and the existing approaches to analog property specification, the requirements for a new specification language for analog properties have been discussed in this thesis. On the one hand, it shall meet the requirements for formal specification of verification approaches applied to DATS models. On the other hand, the language syntax shall be oriented on natural language phrases. By synthesis of these requirements, the analog specification language (ASL) was developed in the scope of this thesis. ASL includes a natural language encapsulation of temporal logic operations, advanced operations for determination of transition paths and oscillations, as well as arithmetic calculations on state space variable values. Hence, a combination of high expressiveness with a designer-oriented syntax was achieved. An extended variable concept, parameterized macros and an assertion-layer allow to develop reusable specifications for complex analog properties. The verification algorithms for model checking, that were developed in combination with ASL for application to DATS models generated with the new trajectory-directed approach, offer a significant enhancement compared to the state of the art.

In order to prepare a transition of signal-based to state space-based verification methodologies, an approach to transfer transient simulation results from non-formal test bench simulation flows into a partial state space representation in form of a DATS has been developed in the scope of this thesis. As has been demonstrated by examples, the same ASL specification that was developed for formal model checking on complete discrete models could be evaluated without modifications on transient simulation waveforms.

An approach to counterexample generation for the formal ASL model checking methodology offers to generate transition sequences from a defined starting state to a specification-violating state for inspection in transient simulation environments. Based on this counterexample generation, a new formal verification methodology using complete state space-covering input stimuli was developed. On a DATS model of the analog circuit, an input stimulus is determined such that all reachable states and transitions of the modeled circuit are visited at least once from a defined starting state. The generated sequence of tuples of value and time for the input variables represent piecewise linear input stimuli for each input of the circuit. By conducting a transient simulation with these complete state space-covering input stimuli, the circuit adopts every state and transition that were visited during stimulus generation. An alternative formal verification methodology is given by retransferring the transient simulation responses to a DATS model and by applying the ASL verification algorithms in combination with an ASL property specification.

Moreover, the complete state space-covering input stimuli can be applied to develop a formal equivalence checking methodology. The new approach introduced in the scope of this thesis replaces the user-defined input stimuli from conventional non-formal equivalence checking approaches with complete-coverage stimuli. Therewith, the equivalence of two implementations can be proven for every inner state of both systems by comparing the transient simulation responses to the complete-coverage stimuli of both circuits.

In order to visually inspect the results of the newly introduced verification methodologies, an approach to dynamic state space visualization using multi-parallel particle simulation was developed. Due to the particles being randomly distributed over the complete state space and moving corresponding to the state space dynamics, another perspective to the system's behavior is provided that covers the state space and hence offers formal results.

The prototypic implementations of the formal verification methodologies developed in the scope of this thesis have been applied to several example circuits. The acquired results for the new approaches to discrete modeling, specification and verification algorithms all demonstrate the capability of the new verification methodologies to be applied to complex circuit blocks and their properties.

Zusammenfassung (German Abstract)

Gegenstand dieser Dissertation ist die Entwicklung neuer Methodiken zur formalen Verifikation nichtlinearer analoger elektronischer Schaltungen. Dazu werden im Rahmen dieser Arbeit entstandene neue Ansätze in den Bereichen verifikationsgerechte diskrete Modellierung analoger Schaltungen, Spezifikation analoger Schaltungseigenschaften und formale Verifikationsalgorithmen vorgestellt.

Während der Entwurfsprozess digitaler Schaltungen weitgehend automatisiert und formalisiert ist, sind zum Entwurf analoger Schaltungen noch viele manuelle Schritte notwendig. Insbesondere im Bereich der Sicherstellung, dass ein Entwurf die in einem Lastenheft spezifizierten Eigenschaften zu jeder Zeit erfüllt, stehen wesentlich weniger Verfahren zur Verfügung als im Bereich digitaler Schaltungen.

Die systematische Sicherstellung der Spezifikationseinhaltung von Entwurfsschritten wird als Verifikation bezeichnet. Aufbauend auf Analysewerkzeugen, wie der Simulation von Schaltungsverhalten im Zeitbereich unter Berücksichtigung sich verändernder Eingangsgrößen, wird für die Verifikation eine Systematik der durchzuführenden Analysen benötigt. Es gibt zwei Klassen von Verifikationsverfahren, welche durch die Bereiche der konventionellen, nicht-formalen Verifikation und der formalen Verifikation gebildet werden.

Der Stand der Technik im Bereich der Analogverifikation in industriellen Entwurfsprozessen ist die Testbench-basierte Simulation. Dieser nicht-formale Ansatz charakterisiert das Schaltungsverhalten anhand von Simulationsläufen mit einer begrenzten Zahl von benutzerdefinierten Eingangssignalen. Abhängig von der Erfahrung des Schaltungsentwicklers decken diese Signale einen Teil der zukünftigen realen Eingangssignale der Schaltung nach ihrer Fertigung ab. Durch den experimentellen Charakter der Verifikation können kritische nicht-spezifikationsgerechte Verhaltenseigenschaften der Schaltung durch die Simulationen unerreicht und so durch den Schaltungsentwickler unentdeckt bleiben. Das Ausbleiben des Entdeckens weiterer Fehler wird im Bereich der nicht-formalen Verifikation als Erfüllen der Spezifikation betrachtet. Wie bereits erläutert, ist dies aber nicht hinreichend für den Nachweis, dass die Schaltung unter allen zukünftigen Umständen die Spezifikation erfüllt. Diese Problematik wird in dieser Arbeit anhand eines einführenden motivierenden Beispiels dargestellt, bei dem eine Oszillatorschaltung erst nach ihrer Fertigung kritisches Verhalten offenbart hat und so nicht einsetzbar war. Während die Simulationsläufe im Entwurfsprozess die Schaltung als voll funktionsfähig darstellten, zeigte sich später, dass bestimmte Startbedingungen, sogenannte „Initial Conditions“ diese Schaltung reproduzierbar daran hindern können, in eine Oszillation zu laufen. Die im Rahmen dieser Arbeit entwickelten formalen Verifikationsmethodiken können derartige Fehler identifizieren.

Formale Verifikationsverfahren für analoge Schaltungen sind noch Gegenstand der Forschung. Formale Verifikation beweist, dass für alle möglichen Eingangssignale und für alle möglichen internen Zustände einer Schaltung die Spezifikation eingehalten wird. Formale Verfahren, die ein Modell einer Schaltung automatisiert auf die Einhaltung einer deklarativen maschinenlesbaren Eigenschaftsspezifikation überprüfen, werden als „Model Checking“ bezeichnet. Vergleicht man formal die Äquivalenz zweier Implementierungen, stellt dieser Prozess das Verfahren des „Equivalence Checking“ dar.

Ausgehend vom Stand der Technik der Modellierung analoger Schaltungen für die simulationsbasierte Verifikation wird im Rahmen dieser Arbeit die diskrete Modellierung analoger Schaltungen für zustandsraumbasierte formale Verifikationsverfahren betrachtet. Der leistungsfähigste bestehende Ansatz zur diskreten zustandsraumbasierten Modellierung teilt den Zustandsraum in achsenparallele Hyperboxen homogenen Verhaltens der Zustandsraumdynamik auf. Hierbei besteht eine Problematik bei der Abbildung nicht-achsenparalleler Vektorfelddynamik, die die Zustandsraumdynamik repräsentiert. Da der bestehende Ansatz nicht rotationsinvariant im Bezug auf die Vektorfeldstruktur ist, findet eine massive Überabschätzung der Nachfolgerrelation des diskreten Transitionsmodells statt. Um dieser Problematik entgegenzutreten wurde im Rahmen dieser Arbeit ein neuer Ansatz zur diskreten Modellierung entwickelt, der die Aufteilungsstruktur anhand der Trajektorien der Vektorfelddynamik bestimmt. So wird eine hohe Genauigkeit der Nachfolgerrelation ermöglicht, woraus eine niedrigere Zahl an Zuständen für ein diskretes Modell gleicher Genauigkeit im Vergleich mit dem Hyperbox-Ansatz folgt.

Der neue Ansatz zur Trajektorien-gesteuerten Partitionierung basiert auf der Bestimmung eines initialen transienten Simulationsschritts von einem Startpunkt im Zustandsraum. Um den Transitionsvektor wird unter Einsatz des Gram-Schmidt-Orthogonalisierungsverfahrens ein Orthogonalsystem erzeugt, dessen Addition und Trajektorienflussrichtung-korrigierte Subtraktion mit dem initialen Startpunkt neue Startpunkte für transiente Schritte mit nachfolgender Orthogonalsystem-Erzeugung bildet. Eine Skalierung der Vektoren stellt die Homogenität eingeschlossener Zustandsraumpartitionen sicher. Mit diesem neuen Diskretisierungsverfahren werden die Eckpunkte geometrischer Objekte bestimmt, die den Zustandsraum bis zu nutzerdefinierten Ausdehnungsgrenzen partitionieren. Während die Topologie des Graphen der Ecken und Kanten der Partitionsobjekte isomorph zum entsprechend konstruierten Graph eines Hyperwürfels ist, gibt es sonst keine Regularität zur einfachen Beschreibung der Partitionen durch Objekte, wie z.B. Polytope. Eine vorgeschlagene Approximation der Grenzflächen der Partitionsobjekte durch eine gewichtete Kombination von Hyperebenen erlaubt dennoch die Bestimmung von Punkt-Einschlüssen in den Partitionsobjekten. Die Abbildung der Partitionierung auf eine diskrete analoge Transitionsstruktur (DATS) erfolgte auf einer dualen Darstellung der Partitionsobjekte, sodass die von transienten Simulationsschritten repräsentierten Kanten der Partitionen als Zentren von Zustandsraumgebieten betrachtet werden können. Dies erlaubt eine effiziente Bestimmung des diskreten Modells mit hoher Genauigkeit der Transitionsrelation, die so direkt durch transiente Simulationsschritte bestimmt wird.

Da automatisierte Verifikationsmethodiken im analogen Bereich noch nicht etabliert sind, ist in der Praxis die Formalisierung der Spezifikation ebenfalls noch nicht weit fortgeschritten. Als nächster Entwicklungsschritt ist eine verbreitete Anwendung von Verfahren zur Assertion-basierten Simulation im Analogbereich zu erwarten. Eigenschaftsspezifikationen für dieses Verfahren, bei dem Simulationsergebnisse automatisiert mit einer maschinenlesbaren Spezifikation verglichen werden, stellen einen ersten Schritt zur Formalisierung analoger Eigenschaftsspezifikation dar. Die hierbei eingesetzte signalbasierte Eigenschaftsformulierung ist allerdings nicht für die zustandsraumbasierte Verifikation einsetzbar.

Die formale Spezifikation von Eigenschaften in ersten Ansätzen zum Model Checking analoger Schaltungen hat sich stark an den bestehenden Verfahren aus dem Bereich digitaler Hardware orientiert. Eine Erweiterung der Temporallogik „Computation Tree Logic“ (CTL) um einen analogen Operator und die Spezifikation von Zeitbeschränkungen erlaubten nur eine sehr begrenzte Formulierung analoger Systemeigenschaften. Analoge Eigenschaften wie z.B. Flankensteilheit und Oszillation sind grundlegend anders zu spezifizieren als digitale Eigenschaften wie „Fairness“ und „Lebendigkeit“. Zudem ist die temporallogische Spezifikation bereits im digitalen Bereich als nicht anwenderfreundlich betrachtet worden. Analoge Eigenschaften mit CTL zu spezifizieren ist somit für die nicht aus dem Bereich der Informatik stammenden Analogentwickler nicht zielführend.

Ausgehend von einer Analyse digitaler Spezifikationskonzepte und der bestehenden Ansätze für analoge Eigenschaften wurden Anforderungen an eine neue Spezifikationssprache für analoge Eigenschaften abgeleitet. Sie soll den formalen Spezifikationsansprüchen für Verifikationsverfahren auf diskreten analogen Transitionsstrukturen genügen und dabei eine Sprachsyntax besitzen, die an natürlichsprachliche Formulierungen angelehnt ist. Die aus diesen Anforderungen im Rahmen dieser Arbeit entwickelte analoge Spezifikationssprache „Analog Specification Language“ (ASL) basiert auf einer natürlichsprachlichen Kapselung temporallogischer Operationen, die mit erweiterten Algorithmen zur Transitionspfadbestimmung, Durchführung von Berechnungen auf Zustandsparametern und Oszillationsbestimmung eine hohe Ausdrucksstärke analoger Eigenschaften mit einer anwenderfreundlichen Syntax kombinieren konnte. Ein erweitertes Variablenkonzept, Kapselung in parametrisierte Makros und eine Assertionen-Ebene erlauben es, wiederverwendbare Spezifikationen für komplexe Eigenschaften zu erzeugen. Die zusammen mit ASL entwickelten Model Checking-Verifikationsalgorithmen zur Auswertung von ASL-Spezifikationen auf einem mit dem Trajektorien-gesteuerten Diskretisierungsverfahren erzeugten DATS-Modell bilden eine wesentliche Erweiterung zum Stand der Technik. Die neuen Spezifikationsmöglichkeiten konnten anhand von neuen Spezifikationsmethodiken für Überschwingen, erweiterte Oszillationseigenschaften wie Eingangsspannungssensitivität von spannungsgesteuerten Oszillatoren und Startverhalten von autonomen Schaltungen demonstriert werden.

Um einen Übergang der Verifikation von signalbasierten zu zustandsraumbasierten Methodiken zu ermöglichen, wurde im Rahmen dieser Arbeit ein Ansatz entwickelt, der die Übertragung von transienten Simulationsergebnissen aus nicht-formalen Testbench-Simulationsumgebungen in eine partielle DATS-Zustandsraumdarstellung erlaubt. Damit kann, wie anhand von Beispielen gezeigt werden konnte, die gleiche ASL-Spezifikation für Eigenschaften eines vollständigen diskreten Modells ohne Modifikation auch auf Simulationsergebnissen ausgewertet werden.

Ein für das formale ASL-basierte Model Checking entwickelter Ansatz zur Erzeugung von Gegebenbeispielen für als spezifikationsverletzend identifizierte Zustandsraumgebiete erlaubt es, Transitionsfolgen von einem definierten Startzustand zu einem spezifikationsverletzenden Zustand zu ermitteln. Diese Transitionsfolgen entsprechen auf einer DATS stückweise-linearen analogen Signalverläufen, die das in den ungewünschten Zustand führende Schaltungsverhalten in Signalform repräsentieren. Neben der Möglichkeit der direkten Beurteilung aller Zustandsraumparameter auf Signalebene bietet dieser Ansatz die Möglichkeit, den Gegenbeispiel-Signalverlauf auch für die Eingangsvariablen der Schaltung zu exportieren. Diese stückweise-linearen Eingangsstimuli können in einer herkömmlichen Testbench-Umgebung verwendet werden, um das spezifikationsverletzende Verhalten in einer gewohnten Verifikationsumgebung per Simulation mit diesen Eingangsstimuli zu reproduzieren.

Auf Basis des Gegenbeispiel-Verfahrens wurde eine neue formale Verifikationsmethodik mittels vollständig den Zustandsraum einer Schaltung abdeckenden Eingangsstimuli entwickelt. Die zugrundeliegende Motivation war es, formale Verfahren in die bislang nicht-formalen Testbench-basierten Simulationsumgebungen zu integrieren. Dazu wird auf einem diskreten Modell der analogen Schaltung ein Eingangsstimulus so ermittelt, dass von einem Startzustand aus alle im Modell vorhandenen Zustände und Transitionen mindestens einmal besucht werden. Die dabei entstehende Folge von Wert-Zeit-Paaren für die Eingangssignale stellen wiederum stückweise-lineare Eingangsstimuli für jeden Eingang der Schaltung dar. Führt man eine transiente Simulation der Schaltung mit diesen vollständig den Zustandsraum abdeckenden Stimuli durch, wird die Schaltung während der Simulation alle Zustände und Transitionen annehmen, die bei der Traversierung des diskreten Modells aufgezeichnet wurden. Somit kann eine Simulation in einer konventionellen Testbench-Umgebung durchgeführt werden, die im Gegensatz zu anwenderdefinierten Stimuli jeden möglichen erreichbaren Zustand der Schaltung annimmt. Mit der Gültigkeit der Simulationsergebnisse für jeden Zustand der Schaltung ist somit ein effizientes Verfahren zur vollständig den Zustandsraum abdeckenden und somit formalen Simulation gegeben. Überträgt man die mit diesem Verfahren ermittelten Simulationsergebnisse wieder in ein DATS-Modell und führt darauf ASL-Verifikationsalgorithmen aus, ist eine formale Assertion-basierte Verifikation möglich, die eine alternative Verifikationsmethodik zum Model Checking darstellt.

Die vollständig den Zustandsraum abdeckenden Eingangsstimuli bieten noch eine weitere Anwendungsmöglichkeit im Bereich des Äquivalenzvergleichs. Nur wenige existierende Ansätze im Bereich der Forschung bieten die Möglichkeit, für nichtlineare analoge Schaltungen die vollständige Verhaltensäquivalenz bezüglich einer definierten Fehlergrenze zu beweisen. Aufgrund komplexer Algorithmen ist der Anwendungsbereich limitiert auf den Vergleich von Implementierungen, die keine wesentlichen Unterschiede in ihrem Abstraktionsgrad besitzen. Im industriellen Einsatz wird Äquivalenzvergleich nicht-formal durch den Vergleich von Simulationsergebnissen durchgeführt, die mittels anwenderdefinierten Eingangsstimuli berechnet wurden. Die im Rahmen dieser Arbeit entwickelte Methodik zum formalen Äquivalenzvergleich auf Basis der vollständig den Zustandsraum abdeckenden Eingangsstimuli ersetzt die anwenderdefinierten Eingangsstimuli durch die vollständig den Zustandsraum abdeckenden. So kann die Äquivalenz für jeden möglichen Zustand der zu vergleichenden Implementierungen anhand eines automatisierten Vergleichs der Simulationsergebnisse beider Implementierungen gezeigt werden. Eine vollständige Verifikationsaussage kann getroffen werden, wenn Stimuli für jede der zu vergleichenden Implementierungen generiert werden und die Simulation jeder Implementierung mit allen Stimuli der eigenen und der anderen Implementierung erfolgt. Der Abstraktionsgrad zwischen den Implementierungen ist hierbei irrelevant.

Um die Ergebnisse der neu eingeführten formalen Verifikationsmethodiken visuell zu untersuchen, wurde ein Verfahren entwickelt, das den Zustandsraum und seine Dynamik mittels eines Partikel-Simulationsansatzes visualisiert. Da die Partikel über den gesamten Zustandsraum randomisiert verteilt werden und sich dann gemäß der Vektorfelddynamik fortbewegen, kann auch hier ein Einblick in das Systemverhalten gewonnen werden, der eine weitestgehend vollständige und somit formale Repräsentation des Zustandsraums bietet.

Die prototypische Implementierung der im Rahmen dieser Arbeit entwickelten formalen Verifikationsmethodiken wurde auf zahlreiche Beispielschaltungen angewendet. Ein modifizierter Ringoszillator wurde mittels ASL-Model Checking und Partikelsimulation auf Startbedingungen untersucht, die eine Oszillation verhindern. Das Überschwingen eines aktiven Sallen-Key Tiefpassfilters wurde mittels ASL-Model Checking mit Gegenbeispielgenerierung untersucht. Eine transiente Simulation dieser Schaltung mit einem vollständig den Zustandsraum abdeckenden Eingangsstimulus wurde wiederum mit der ASL-Spezifikation verifiziert. Ein Vergleich der neuen diskreten Modellierung mittels des Trajektorien-gesteuerten Ansatzes mit dem Hyperbox-Diskretisierungsverfahren konnte eine wesentliche Steigerung der Modellierungsgenauigkeit des neuen Verfahrens dokumentieren. Für eine Charge-Pump-Schaltung wurde das Startverhalten per Model Checking verifiziert und die ASL-Spezifikation zudem auf die Ergebnisse einer konventionellen transienten Simulation angewendet. Das Model Checking der Eingangsspannungssensitivität eines spannungsgesteuerten Oszillators konnte sowohl die Fähigkeit des neuen Diskretisierungsverfahrens wie auch der Spezifikations- und Verifikationsmethodik zeigen, erfolgreich auf komplexen Zustandsraumstrukturen zu operieren. Schließlich wurde das neue Stimuli-basierte Äquivalenzvergleichsverfahren anhand einer Bandpass-Schaltung, eines Delta-Sigma-Modulators zweiter Ordnung und weiterer Schaltungen demonstriert und mit einem bestehenden Verfahren verglichen.

Trotz der in der Arbeit diskutierten bestehenden Herausforderungen, die bis zu einem industriellen Einsatz der neu vorgestellten Methodiken noch bearbeitet werden müssen, konnten mehrere neue Methodiken zur Formalisierung analoger Verifikation motiviert und erfolgreich prototypisch umgesetzt werden.

Contents

Acknowledgments i

Abstract iii

Zusammenfassung (German Abstract) vii

List of Tables xviii

List of Figures xxii

List of Algorithms xxiii

List of Symbols and Abbreviations xxv

1 Introduction 1
- 1.1 Analog Circuit Design Flow . . . 2
- 1.2 Analog Circuit Analysis and Verification . . . 4
- 1.3 Formal Verification . . . 4
- 1.4 Motivating Example . . . 5
- 1.5 Contributions . . . 7
- 1.6 Publications . . . 9
- 1.7 Overview . . . 9

2 System Representation for Verification 11
- 2.1 System Description . . . 11
- 2.2 Digital Systems . . . 14
- 2.3 Analog Systems . . . 16
  - 2.3.1 Device Models . . . 17
  - 2.3.2 Network Analysis using the Modified Nodal Approach . . . 18
  - 2.3.3 Numerical Simulation . . . 19
  - 2.3.4 DAE Index . . . 21

2.3.5 Analog Behavioral Modeling and Hardware Description Languages . . . 21
2.3.6 State Space Representation . . . 24
2.4 Discrete Modeling of Analog Systems . . . 24
2.4.1 Discrete Analog Transition Structure . . . 25
2.4.2 The Discretization Problem for Analog Circuits . . . 27
2.4.3 Hyperbox Discretization . . . 31
2.4.4 Trajectory-Directed Discretization . . . 32
2.4.4.1 Calculating the State Space Partitioning . . . 33
2.4.4.2 Geometric Structure of the State Space Partitions . . . 38
2.4.4.3 Transition Relation of the Hypercells . . . 43
2.4.4.4 Mapping the Trajectory-Directed Partitioning to a DATS 44
2.4.4.5 Duality of the Trajectory-Directed Partitioning . . . 45
2.4.4.6 Handling Input Variables . . . 47
2.4.4.7 Runtime Complexity . . . 48
2.4.4.8 Modeling Error Analysis . . . 50

3 Property Specification for Verification 55
3.1 Basic Definitions . . . 55
3.2 Operational and Declarative Specification . . . 56
3.3 Property Specification for Discontinuous Systems . . . 57
3.3.1 Linear Temporal Logic (LTL) . . . 58
3.3.2 Computation Tree Logic (CTL) . . . 59
3.3.3 Property Specification Language (PSL) . . . 61
3.4 Existing Approaches to Specification of Analog System Properties . . . 62
3.4.1 Specification of Assertions within Analog Hardware Description Languages . . . 63
3.4.2 PSL for Analog Signal Property Specification . . . 64
3.4.3 CTL Specification of Analog Properties in the State Space . . . 64
3.5 Analog Specification Language (ASL) . . . 65
3.5.1 Language Concept . . . 67
3.5.2 EBNF Grammar of ASL . . . 67
3.5.3 Semantics of ASL Operations . . . 69

4 Verification of Systems 75
4.1 Non-Formal Verification . . . 76
4.1.1 Simulation-Based Verification . . . 76
4.1.2 Assertion-Based Verification . . . 76
4.2 Verification Coverage . . . 77
4.2.1 Structural Coverage . . . 79
4.2.2 Functional Coverage . . . 80

4.3 Formal Verification 80
    4.3.1 Model Checking 81
    4.3.2 Equivalence Checking 82
4.4 Existing Approaches to Non-Formal Analog Circuit Verification 83
    4.4.1 Assertion-Based Approaches to Analog Verification 83
    4.4.2 Analog Verification Coverage 84
    4.4.3 Formalizing the Analog Verification Flow 85
4.5 Existing Formal Approaches to Analog Verification 86
    4.5.1 Reachability Analysis and Verification of Analog Circuits 86
    4.5.2 Analog Model Checking 87
    4.5.3 Analog Equivalence Checking 88
        4.5.3.1 The VERA Equivalence Checking Algorithm 90

5 Analog Formal Verification Methodologies **93**
5.1 New Verification Methodologies for the Analog Design Flow 93
    5.1.1 Verification Methodology Perspective 94
    5.1.2 Design Flow Perspective 95
5.2 ASL Property Specification and Verification Methodology 96
    5.2.1 Specification of Circuit Overshoot 96
    5.2.2 Specification of Oscillation and Voltage Controlled Oscillator Gain K_{VCO} 98
    5.2.3 Specification of the Startup Time of Autonomous Circuits 100
5.3 ASL Property Verification on Transient Simulation Waveforms 101
5.4 Counterexample Generation for Model Checking 103
    5.4.1 Counterexample Generation in the ASL Verification Flow 106
5.5 Complete-Coverage Input Stimuli Generation 106
    5.5.1 Verification Methodology 109
5.6 Equivalence Checking using Complete-Coverage Input Stimuli 111
    5.6.1 Error Measures for Waveform Comparison 112
    5.6.2 Automation in the ASL Verification Flow 113
5.7 Multi-Parallel State Space Particle Simulation 114

6 Experimental Results **117**
6.1 Implementation 117
6.2 Verification of Initial Conditions of a Ring Oscillator 118
6.3 Verification of Active Lowpass Filter Overshoot 120
    6.3.1 Model Checking 122
    6.3.2 Counterexample 123
    6.3.3 Complete-Coverage Input Stimulus 124
    6.3.4 Comparison to Hyperbox Discretization 125
    6.3.5 ASL Verification on Simulation Waveforms 126

6.4 Verification of CMOS Charge Pump Startup Time 128
    6.4.1 Model Checking 128
    6.4.2 ASL Verification on Simulation Waveforms 131
6.5 Model Checking of VCO Gain K_{VCO} 132
6.6 Equivalence Checking with Complete-Coverage Stimuli 134
    6.6.1 Biquad Bandpass Filter 135
    6.6.2 Second-Order Delta-Sigma Modulator 136
    6.6.3 Further Circuit Examples 138
6.7 Assessment 140

7 Conclusions **143**
7.1 Summary 143
7.2 Challenges and Future Work 145

A Appendix **147**
A.1 VCO State Space Slices 147

Bibliography **149**

List of Tables

2.1 State transition table for the two-bit down counter. 15
2.2 Comparison of the trajectory-directed discretization and the hyperbox discretization approaches. 53

3.1 Explanation of LTL syntax. 59
3.2 Explanation of CTL syntax. 60
3.3 Explanation of CTL-AT syntax. 65

6.1 Verification results for the modified ring oscillator with results obtained from ASL model checking (MC) and transient analysis (TRA). 119
6.2 Runtimes of the trajectory-directed discretization algorithm for the Sallen-Key lowpass filter. 121
6.3 Verification results calculated by the ASL model checking algorithms on the DATS for the Sallen-Key lowpass filter. 123
6.4 Verification results calculated by the ASL model checking (MC) algorithms on the DATS for the Sallen-Key lowpass filter compared to the ASL evaluation on the transient simulation results (Sim) using a complete-coverage input stimulus. 128
6.5 Runtimes of the trajectory-directed discretization algorithm for the CMOS charge pump. 129
6.6 Comparison between ASL property evaluation on a DATS generated by the trajectory-directed discretization (TDD) method and on simulation waveforms obtained from transient analysis (TRA). 132
6.7 Comparison of the results between ASL model checking and transient analysis for the oscillation period of the VCO at different input voltages. 134
6.8 Comparison of the results between ASL model checking and transient analysis for the gain K_{VCO} of the VCO. 134
6.9 Comparison of the results between equivalence checking by complete state space-covering input stimuli (stimEC) and the transformed state space comparison approach (VERA). 141

6.10 Runtimes of the stimEC equivalence checking approach compared to the approximated simulation runtimes of circuit comparison by systematic simulation. 142

List of Figures

1.1 Y-chart for hierarchical analog design with exemplary synthesis steps. . 3
1.2 The synthesis steps in the design flow transfer a specification to an implementation. Every implementation step needs to be analyzed for specification conformance by verification methods. 5
1.3 Modified ring oscillator with an even number of inverter stages and cross-coupling. 6
1.4 Transient responses of the ring oscillator for transistor ratio $\alpha/\beta = 1.95$. 7
1.5 State space trajectories of initial conditions leading into the non-oscillating steady states for transistor ratio $\alpha/\beta = 1.95$. 8

2.1 General illustration of a system. 12
2.2 Circuit schematic for the two-bit down counter (a) and the corresponding state transition graph (b). 16
2.3 Simple nonlinear analog circuit example with input voltage V_{in}, resistor R_1, diode D_1 and capacitor C_1. 19
2.4 Schematic of the CMOS implementation of the charge pump circuit (a) and simplified macro model (b). 22
2.5 Transient startup response of the transistor netlist charge pump (solid line) and the simplified model (dashed line). 23
2.6 Schematic illustration of a graph structure representing a DATS. 27
2.7 Illustration of a vector field with paraxial flow. 33
2.8 Illustration of a non-paraxial vector field flow with a trajectory-directed state space partitioning. 34
2.9 Schematic visualization of the process of determining the trajectory-orthogonal point sets $\mathbf{q}^{(a)}$ and $\mathbf{q}^{(b)}$ in a two-dimensional space. 37
2.10 Illustration of the orthogonal sets constructed around the transition vectors in a three-dimensional state space. 38
2.11 Illustration of a hypercell object in a three-dimensional space. 40
2.12 Basis vector sets for each vertex of a hypercell facet for determination of the position of point $\mathbf{p}$ with respect to the facet. 41
2.13 Curved surface in a three-dimensional space representing a facet of a cell. 42

2.14 Schematic illustration of contracting and expanding vector field flow, with the corresponding partitioning and the successor relation for the hypercells. 44
2.15 Illustration of a two-dimensional vector field with calculated transition endpoints as boxes and the quadrilateral enclosed regions of the state space. 46
2.16 Schematic illustration of a DATS with a possible input change-induced transition path (a). The corresponding input signal (b) and output signal (c) is assumed for this path. 48

3.1 Property space with specification and performances. 56
3.2 Illustration of the LTL operation semantics. 59
3.3 Illustration of the CTL operation semantics. 60
3.4 Illustration of ASL algorithms transition, oscillation and delta_compare. 73

4.1 Simulation-based verification flow. 77
4.2 Assertion-based verification flow. 78
4.3 Illustration of structural coverage. 80
4.4 Model checking flow. 81
4.5 Comparison of verification coverage in the state space by test bench-based transient simulation and formal verification. 85
4.6 Structure of the VERA equivalence checking methodology. 91

5.1 Analog verification framework for different verification methodologies. 94
5.2 New verification methodologies in the analog design flow. 96
5.3 Overshoot of the output voltage caused by a trajectory in the state space. 97
5.4 Oscillation in the time domain (a), in the continuous state space (b) and in the DATS (c). 98
5.5 Schematic illustration of oscillations for different input voltages V_1, V_2, V_3 and V_4. 99
5.6 Schematic illustration of startup transition paths. 101
5.7 Graph structure obtained by transient simulation waveforms. 103
5.8 Periodic transient signal waveforms $s_1(t)$ (a) and $s_2(t)$ (b). State space representation of periodic signals $s_1(t)$ and $s_2(t)$ (c). 103
5.9 ASL assertion-based verification flow for transient simulation results transferred to a state space representation. 104
5.10 Path generated by the stimuli generation algorithm (a) and the corresponding input/output behavior (b). 109
5.11 Complete-coverage input stimuli generation and verification flow. . . . 110
5.12 Equivalence checking flow using complete state space-covering input stimuli. 113

5.13 Determining nearest sample point $\mathbf{q}_2$ in state space for particle $\mathbf{p}_1$ within discrete vector field $\mathcal{V}_D$. 115
5.14 Particle simulation for the two-dimensional state space of an oscillator circuit with increasing time from (a) to (d). 116

6.1 Transition vectors between states of the detected oscillation set and non-periodic steady states of the modified ring oscillator detected by model checking for α/β-ratio 1.95 (a). Particle simulation visualizing the oscillation trajectory and the non-periodic steady states (b). 120
6.2 Circuit schematic of Sallen-Key biquad lowpass filter. 121
6.3 Reachable state space of the Sallen-Key biquad lowpass filter. 122
6.4 Transient response to the counterexample input stimulus of the Sallen-Key biquad lowpass filter. 124
6.5 Complete generated input stimulus and transient output response of the Sallen-Key biquad lowpass filter. 124
6.6 Transient response to the generated input stimulus of the Sallen-Key biquad lowpass filter plotted over V_{C_1} and V_{C_2}. 125
6.7 Excerpt of the transient response to the complete-coverage input stimulus of the Sallen-Key biquad lowpass filter with the expected response generated by the trajectory-directed discretization approach. 126
6.8 Transient response to the complete-coverage input stimulus of the Sallen-Key biquad lowpass filter with the expected response generated by the hyperbox discretization approach. 127
6.9 Circuit schematic of the CMOS charge pump. 128
6.10 Transient output waveform of the CMOS charge pump circuit. 129
6.11 Startup trajectory projected to V_{C_1}, $V_{C_{load}}$ and V_{clk}, identified by evaluating the ASL property specification on the DATS of the charge pump. . . 130
6.12 Transient simulation waveforms of the charge pump for V_{C_1}, $V_{C_{load}}$ and V_{clk} transferred into a DATS representation for application of ASL verification algorithms. 131
6.13 Circuit schematic of the voltage controlled oscillator. 132
6.14 Oscillation areas of the voltage controlled oscillator at different control voltages. 133
6.15 Circuit schematic of the biquad bandpass filter circuit netlist. 136
6.16 Complete generated input stimulus and transient output response of the biquad bandpass filter transistor netlist (Output_A) and VHDL-AMS behavioral model (Output_B). 137
6.17 Transient response to the generated input stimulus of the biquad bandpass filter plotted over V_{C_1} and V_{C_2}. 137
6.18 Circuit schematic of the second-order delta-sigma modulator. 138

6.19 Simple behavioral model for the second-order delta-sigma modulator using an allpass filter for signal delay modeling. 138
6.20 Transient response to the generated input stimulus of the delta-sigma modulator transistor netlist plotted over V_{C_1} and V_{C_2}. 139

A.1 State space slices with all state transitions for input voltages 0.66 V (a), 0.83 V (b), 1.00 V (c) and 1.17 V (d) showing the adaption of the trajectory-directed discretization to the changed transition structure. . . 148

List of Algorithms

1 Trajectory-Directed Partitioning Algorithm. 39

2 ASL Transition Algorithm. 71
3 ASL Oscillation Algorithm. 71
4 ASL Delta_compare Algorithm. 72

5 Counterexample Generation Algorithm. 105
6 Complete-Coverage Input Stimuli Generation Algorithm. 108
7 Particle Simulation Algorithm. 115

List of Symbols and Abbreviations

Notation	
x	Scalar value $x \in \mathbb{R}$
$\mathbf{x}$	Vector $\mathbf{x} = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$
$\mathbf{A}$	Matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
$\lvert x \rvert$	Absolute value of x
$[x]$	Interval $[x] = [\underline{x}, \overline{x}]$
$\lVert \mathbf{x} \rVert$	Vector norm of $\mathbf{x}$
$\dot{\mathbf{x}}$	Temporal derivative of $\mathbf{x}(t)$
$f()$	Function returning a scalar
$\mathbf{f}()$	Function returning a vector

Symbols	
$\mathbf{a}$	Vertices constraining a partition of the trajectory-directed discretization
α, β	Scalar factors
$\mathbf{B}$	Orthogonal basis
$\mathcal{C}$	Runtime complexity
deg	Out-degree
Δ_{rs}	Length difference ratio between two vectors $\mathbf{v}_r$ and $\mathbf{v}_s$
ϵ	Error function
ϵ_Δ	Mean direction error
ϵ_θ	Mean length error
ϵ_{suc}	Mean successor relation error
$\mathbf{J}$	Jacobian matrix
k	Number of state space partitions (Section 2.4)

κ	Approximated number of sample points of the trajectory-directed discretization in one dimension
L	Labeling function
L_A	Labeling function of the DATS for the atomic propositions
L_T	Labeling function of the DATS for the atomic transition propositions
L_V	Labeling function of the DATS for the extended state space variable values
L_X	Labeling function of the DATS for the DAE system's inner variable values
M	Model
n_d	Number of dimensions of the extended state space $\mathbf{z}^{(e)}$
n_u	Number of input variables in $\mathbf{u}$
n_y	Number of output variables in $\mathbf{y}$
n_z	Number of linear independent state space variables in $\mathbf{z}$
$\mathcal{O}$	Asymptotic upper complexity bound
P	Property space
P_{spec}	Specification
π	Ordered sequence of states $\pi = \sigma_1, \sigma_2, ..., \sigma_n$ representing a directed state path
Π	Set of paths $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$
ϕ, ψ	Formula in a specification logic
$\mathbf{p}, \mathbf{q}$	Points in $\mathcal{Z}$
R	Transition relation
$\mathcal{R}$	Partition of $\mathcal{Z}$
$\mathbb{R}$	Set of all real numbers
r	User-defined scalar value
S	System
Σ	Unordered set $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_n\}$ of states
T	Temporal labeling function for the edges of a DATS
θ_{rs}	Angle between two vectors $\mathbf{v}_r$ and $\mathbf{v}_s$
$\mathbf{u}$	Vector $\mathbf{u} = (u_1, u_2, ..., u_{n_u})^T$ of input variables
$\mathcal{V}$	Continuous vector field in $\mathbb{R}^{n_d}$
$\mathbf{v}$	Direction vector

$\mathbf{x}$	Vector $\mathbf{x} = (x_1, x_2, ..., x_{n_x})^T$ of inner variables of a system
$\mathbf{y}$	Vector $\mathbf{y} = (y_1, y_2, ..., y_{n_y})^T$ of output variables
$\mathbf{z}$	Vector $\mathbf{z} = (z_1, z_2, ..., z_{n_z})^T$ of linear independent state space variables
$\mathbf{z}^{(e)}$	Vector $\mathbf{z}^{(e)}$ of extended state space variables $(\mathbf{z}^T, \mathbf{u}^T)^T$
$\mathcal{Z}$	Infinite point set in $\mathbb{R}^{n_d}$
$\mathbb{Z}$	Set of all integer numbers

Abbreviations

ABV	Assertion-Based Verification
AC	Alternating Current
AMS	Analog/Mixed-Signal
AP	Atomic Proposition
ASL	Analog Specification Language
ATPG	Automatic Test Pattern Generation
AVF	Analog Verification Framework
BCE	Branch Constitutive Equation
BDD	Binary Decision Diagram
BE	Backward Euler
BMD	Binary Moment Diagram
CCIS	Complete-Coverage Input Stimuli
CMOS	Complementary Metal Oxide Semiconductor
CTL	Computation Tree Logic
DAE	Differential Algebraic Equation
DATS	Discrete Analog Transition Structure
DC	Direct Current
DUV	Design Under Verification
EBNF	Extended Backus-Naur-Form
EC	Equivalence Checking
EDA	Electronic Design Automation
FE	Forward Euler
HDL	Hardware Description Language
KCL	Kirchhoff's Current Law

KVL	Kirchhoff's Voltage Law
LIC	Line Integral Convolution
LTE	Local Truncation Error
LTL	Linear Temporal Logic
MC	Model Checking
MNA	Modified Nodal Analysis
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
NP	Nondeterministic Polynomial Time
NR	Newton-Raphson
ODE	Ordinary Differential Equation
OVL	Open Verification Library
PL	List of accepted partitioning points
PSL	Property Specification Language
RF	Radio Frequency
RTCTL	Real Time Computation Tree Logic
SAT	Satisfiability Problem of Boolean Formulas
SERE	Sequential Extended Regular Expression
SMV	Symbolic Model Verifier
SPICE	Simulation Program with Integrated Circuit Emphasis
STL	Signal Temporal Logic
SVA	System Verilog Assertions
TDD	Trajectory-Directed Discretization
TR	Transient
TRA	Transient Analysis
VCO	Voltage Controlled Oscillator
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WL	Waiting List

1

Introduction

The impact of electronic devices on our everyday life is inevitable. With the dependency on electronics increasing continuously, the consequences of errors in such electronic systems are increasing just as well. There are several levels of severity from just being disconnected during a phone call to possible airplane crashes due to errors in the electronic components. Even for non-safety-critical cases, errors in electronic systems have an economic dimension, where the cost of missed design flaws is determining whether a company can stay competitive or not.

Due to the increasing system complexity and decreasing time to market, design verification has become a more and more crucial part of the electronic circuit design flow. While formal verification methods are established in the digital domain, industrial analog circuit design flows are lacking formal or at least formalized verification methodologies. Analog circuit verification still depends on the designer's experience and expertise to manually define appropriate test benches for simulation-based design flows and to select the right input signals in order to detect possible design errors.

In contrast to the common perception of digital circuits dominating today's electronic devices, the importance of analog circuits and especially of analog circuit design is increasing. This is due to the fact that most electronic circuits are nowadays mixed-signal systems, using analog interfaces to the external environment in combination with a digital core. Moreover, with decreasing feature sizes, not only the relative percentage of the not equally-scaling analog part of mixed-signal designs increases, but also the analog behavior of digital circuit components becomes more and more critical.

Finally, important parts of digital systems such as clock generators have always been designed on the analog level.

Hence, driven by the perennial demand for higher design efficiency, new methodologies offering more automation of the analog verification process are of vital importance.

While approaches to assertion-based simulation are emerging, which are mainly automating previously manual efforts, they are not targeting the fundamental problem of analog circuit verification: verification coverage. Today's established common verification methodology is analyzing the circuit's behavior by simulation using test benches. Specification conformance is checked by performing several transient simulations with input signals which are considered to be representative for the future operating conditions of the circuit. Although this approach to discover design errors has been working for decades, redesigns have occurred frequently due to missing some critical behavior of the circuit during simulation.

There has been significant progress in several areas of electronic design automation (EDA) for analog circuits. Some complex tasks such as sizing, placement and design centering have been addressed by EDA-vendors, now being available as automated tools which are fully integrated into the design flow. These tools are exploiting algorithmic concepts which by far outperform manual approaches. By contrast, the area of analog design verification is not yet systematically covered by existing tools.

Therefore, the goal of this thesis is to advance the field of formal verification methodologies for analog circuits in order to contribute to the development and future productive application of analog formal verification methodologies.

1.1 Analog Circuit Design Flow

The objective of the design flow for analog circuits is to transfer a functional specification of the design into a physical implementation satisfying the specification. Between the initial specification and the final implementation, several design steps are required in order to hierarchically partition the complex tasks into solvable portions.

For a classification of the design abstraction levels and the design domains, the application of the Y-chart to the analog design task offers a structured approach to this hierarchy [GDWL92, HBKK94]. The Y-chart consists of three views which build the functional domain, the structural domain and the physical domain. Different levels of abstraction range from the top-level concept layer to the low-level component layer. Descending from higher abstraction levels to lower levels by increasing the overall complexity of the design is regarded as synthesis. Comparing the conformance of lower levels of abstraction with higher levels or different domains is regarded as analysis. Figure 1.1 shows the Y-chart for the analog design flow with a possible top-down

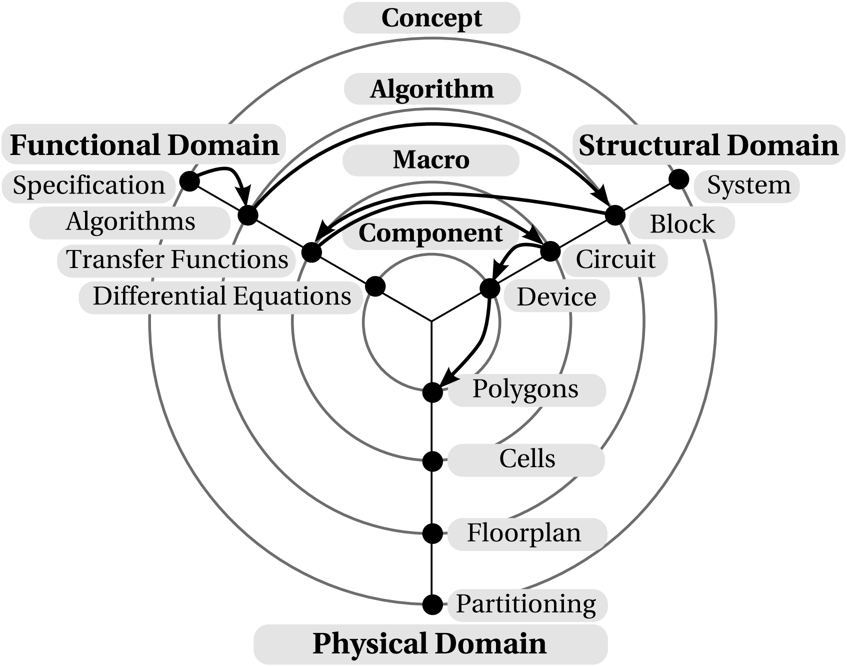


Figure 1.1: *Y-chart for hierarchical analog design with exemplary synthesis steps.*

design flow from the functional specification to the physical implementation, which will be explained in the following.

Starting in the functional domain with a specification on the concept layer, the algorithms describing the circuit's behavior are determined by descending to the algorithm layer in the functional domain. For the selected algorithms, the top level building blocks are allocated by a transition to the structural domain and the topology of these analog blocks has to be generated or is selected from existing libraries. The functional design of the analog blocks is determined by the transfer functions on the macro layer, altogether forming the desired behavior of the previously selected analog system algorithms. The transition back to the structural domain maps the transfer functions to circuit structures, which are then represented by devices on the component layer. The topology of the circuit structures in conjunction with the parameters of the component devices determine the behavior of the circuit. The selection of device parameters in order to meet the specification for a given topology is referred to as circuit sizing. Finally, the devices and their topology have to be transferred to polygons in the physical domain. This process is the layout generation, determining the geometry of the physical structures in silicon for production of the resulting integrated circuit.

From the design synthesis point of view, three main steps characterize the design flow: topology selection or generation, circuit sizing and layout generation. On the other hand, for each synthesis step, it must be assured that the design still satisfies the specification. Hence, a functional equivalence between abstraction level changes or domain changes is mandatory for a successful design flow, which by definition is provided by analysis. Therefore, analysis is the central tool for assuring the validity of every design step.

1.2 Analog Circuit Analysis and Verification

Within the terminology of the Y-chart design flow, the term analysis describes the checking of the synthesis steps to retain functional equivalence with higher abstraction levels and hence with the specification. While in abstract theory this definition is sufficient, in practice, circuit analysis is a multi-faceted issue.

The central tool for circuit analysis is simulation. By simulation, for a given circuit in form of its mathematical representation, the reaction to input excitations is calculated. Different types of simulations offer direct current (DC), alternating current (AC) and transient (TR) analysis.

For DC-analysis the system reaction in its steady state $(t \to \infty)$ is observed. Hence, the steady state characteristics of a circuit can be analyzed by a DC-analysis sweep over an input interval. AC-analysis considers the frequency response of linear systems at the linearized operating points of nonlinear systems. Being the most complex and therewith computationally intensive analysis, TR-simulation calculates the dynamic transient response of the circuit to a piecewise linear input stimulus over time. For this purpose, the implicit nonlinear differential equation system describing the circuit is transferred to a system of difference equations by numerical integration. This system of difference equations in correlation with the input values at the specific time point can then be solved by the Newton-Raphson method. Implementing the described analysis methods, the ancestor of most modern analog circuit simulators is SPICE (Simulation Program with Integrated Circuit Emphasis) [Nag75].

While the capability of analyzing the behavior of analog circuits is the basis of analog circuit verification, methodologies that systematically apply simulations are necessary in order to meet the verification tasks arising from the transitions within the design flow. Thus, in this context, verification can be considered as the systematic application of simulations in order to detect behavioral differences of the abstraction levels or design domains. Figure 1.2 illustrates the relationship between synthesis and analysis in the design flow.

1.3 Formal Verification

In order to detect design errors, non-formal verification procedures try to analyze the design under verification (DUV) whether it corresponds to a specification or a refer-

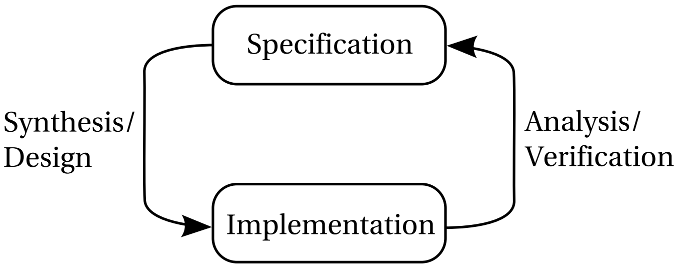


Figure 1.2: *The synthesis steps in the design flow transfer a specification to an implementation. Every implementation step needs to be analyzed for specification conformance by verification methods.*

ence design with a finite number of verification test cases with external conditions of the DUV defined in a test bench.

In contrast, formal verification does not search for deviations between DUV and specification but proves the absence of any deviations for all possible states and input conditions of the DUV. Depending on the verification task, there are two major approaches to formal verification: model checking (MC) and equivalence checking (EC). Model checking algorithms prove that the model M of the DUV is correct for every possible input stimulus and internal state of the system with respect to a property specification P_{spec} specified in a machine-readable specification language. In MC terminology, the model then satisfies the property specification: $M \models P_{spec}$. If the model does not satisfy the specification, all erroneous states are detected and counterexamples in form of state transitions to these erroneous states can be returned.

Equivalence checking proves the general functional equality of two implementations of a design. The implementations can be of different abstraction levels and different description methods such as transistor netlists and hardware description languages.

As will be motivated in this thesis, the concept of formal verification can be generalized from specific algorithmic approaches such as model checking or equivalence checking to the idea of verification coverage. Therewith, any verification approach that obtains results that hold for any input signal and any state of the DUV are considered as formal verification, as they cover the complete possible behavior a DUV can exhibit.

1.4 Motivating Example

The following motivating example will illustrate that conventional analog circuit simulation within a test bench setup can lead to wrong verification assumptions and how, in contrast, a complete formal verification using some of the methodologies developed in this thesis can identify hidden design errors.

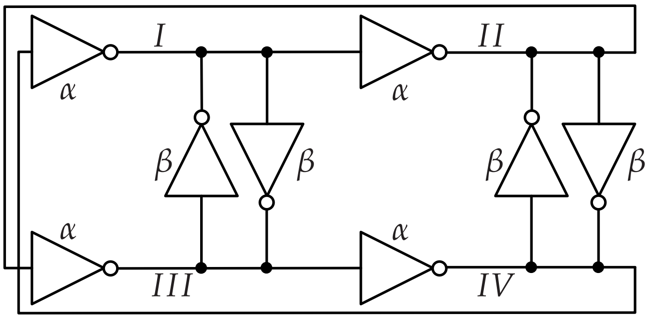


Figure 1.3: *Modified ring oscillator with an even number of inverter stages and cross-coupling.*

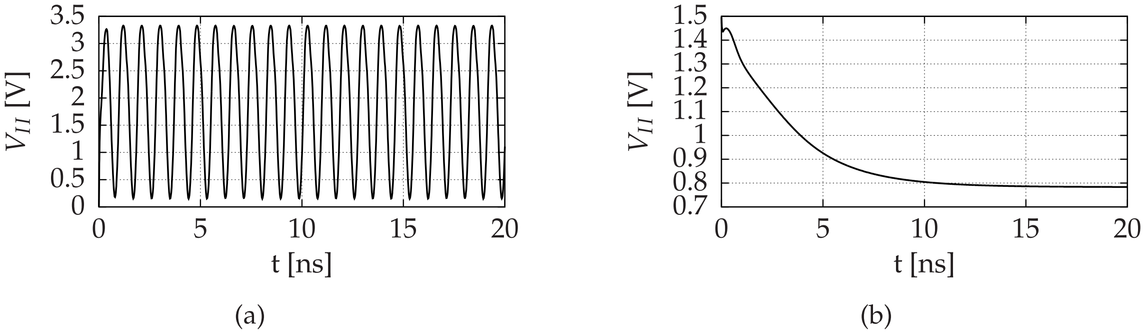
When a finite number of simulations with input stimuli or initial conditions is conducted and the expected behavior of the circuit is validated by the simulations, in today's industrial verification applications the circuit is assumed to be successfully verified due to the lack of formal verification tools. However, as pointed out in the previous section, not finding a specification violation with simulation runs cannot prove that the specification is satisfied under any circumstances.

The example circuit illustrated in Figure 1.3 is a modified ring oscillator with an even number of inverter stages and cross-coupling [JKK08]. Due to the bridges β, the circuit oscillates if there is a ratio α/β of the transistor sizes in the feedback chain to those of the bridges within the interval [0.4, 2.0].

This circuit has in fact been considered as successfully verified by transient simulation with a set of predefined initial conditions and went into production. What was discovered only after the tapeout of the circuit is its crucial property of being prone to certain initial conditions that prevent it from oscillating when the α/β ratio reaches or exceeds the interval boundaries. These particular initial conditions have not been covered by the simulation runs during verification. For two different initial conditions, the simulation runs are illustrated in Figure 1.4.

The critical behavior with certain initial conditions could have been detected by applying the formal verification approaches developed in the scope of this thesis. In order to demonstrate this, a formal property verification of the circuit was performed using the ASL verification methodology (introduced in Section 5.2) on a discrete state space model generated with the trajectory-directed approach (introduced in Section 2.4.4). Therewith, for transistor ratio 1.05, 1.35 and 1.65, no initial conditions violating the oscillation behavior are reported by the verification algorithms.

In contrast to the oscillation starting from every initial condition with transistor ratios 1.05 to 1.65, for transistor ratio 1.95, initial conditions are detected by the formal verification method for which the circuit will not run into an oscillation. This area of nontrivial bad initial conditions is illustrated in Figure 1.5, projected to the state space



(a) (b)

Figure 1.4: *Transient responses for the node voltage* V_{II} *of the ring oscillator for transistor ratio* $\alpha/\beta = 1.95$. *With initial conditions* $V_I, V_{II}, V_{III} = 0$ *V and* $V_{IV} = 0.5$ *V, the circuit oscillates (a). With the initial conditions* $V_I = 3.33444$ *V,* $V_{II} = 1.49605$ *V,* $V_{III} = 3.16195$ *V,* $V_{IV} = 0.207917$ *V detected by formal verification, the circuit does not oscillate (b).*

variables V_I, V_{II} and V_{III}. The non-oscillating behavior shown in Figure 1.4(b) was identified by transient simulations starting from the set of these initial conditions.

1.5 Contributions

With the objective of advancing the state of the art in the field of analog formal verification, this thesis presents a framework of new formal verification methodologies for the analog circuit design flow. The contributions will be outlined in the following.

In order to formalize specifications of analog circuit properties for automated verification approaches, a new Analog Specification Language (ASL) is introduced with a designer-oriented syntax but also with semantics satisfying the demands of formal property verification algorithms. Therefore, the model checking tool ASL-MCT (Analog Specification Language model checking tool), implementing the ASL verification algorithms, was developed to build an ASL-based model checking framework for analog complex property verification based on models obtained by a discrete modeling approach. ASL model checking offers a syntax much easier understandable than CTL-based temporal logic specification and the expressiveness of ASL for analog circuit properties is significantly higher. A state space-based specification methodology developed in conjunction with ASL offers specification reuse by building up a library of parameterizable macros. Regression verification is offered by a high level assertion layer, allowing the combination of several property verifications to an automated verification run with a detailed verification report. For identified design errors, counterexamples in form of piecewise linear input stimuli can be automatically generated in

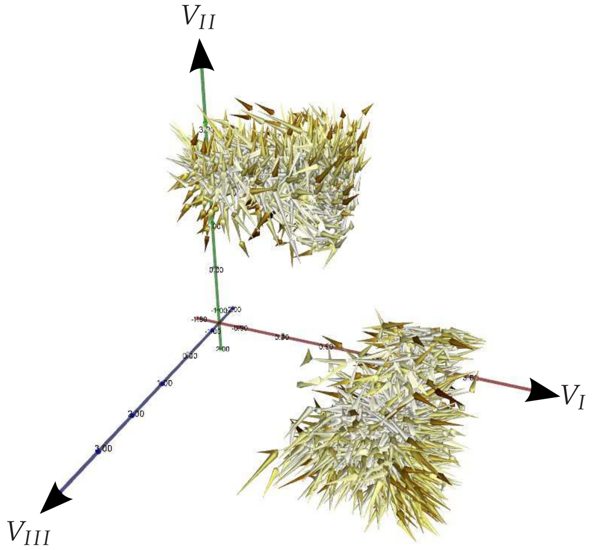


Figure 1.5: *State space trajectories of initial conditions leading into the non-oscillating steady states for transistor ratio* $\alpha/\beta = 1.95$.

order to give the circuit designer the possibility of conducting a conventional transient simulation that leads to the error state.

ASL model checking increases the possible analog properties that can be specified and verified on discrete state space models. Nevertheless, the state-of-the-art discrete modeling approach [HHB02a] based on hyperbox binary space partitioning of the modeled circuit's state space limits the complexity of the circuits that can be modeled. Therefore, a new approach for improving the discrete model generation is developed in the scope of this thesis, increasing modeling precision and therewith reducing the number of state computations necessary for the generated model. The new approach is based on trajectory partitioning of the state space, letting the flow of the state space dynamics define the boundaries of the state space partitions forming the system states.

In order to apply ASL-based specifications to today's non-formal test bench-based simulation flows, a method to transfer conventional simulation waveforms to a state space representation was developed. Therewith, profiting from formalized property specification and verification is possible within the established non-formal industrial design flows, transferring the ASL specification methodology to assertion-based verification. As will be presented, ASL property specifications can be exchanged without modifications between formal ASL model checking and transient simulation waveform evaluation.

In conjunction with the counterexample generation algorithms for ASL verification, a new method of complete state space-covering input stimuli generation was developed. On the discrete state space, a traversal algorithm efficiently visits every reachable state of the state space, recording the piecewise linear input stimulus that is necessary to bring the circuit into this state during a transient simulation. Therewith, a transient simulation with complete coverage of the circuit's state space is obtained and the benefits of a complete, hence formal, verification approach can be provided within a conventional transient simulation-based verification flow. By evaluating these transient simulation results with the aforementioned ASL verification on transient simulation results, an alternative to the model checking approach for formal analog property verification is given.

Moreover, based on the complete state space-covering input stimuli generation, an analog equivalence checking methodology has been developed. By generating such stimuli for each of the two systems under verification A and B and simulating each system with the stimuli generated for A and B, the level of equivalence of both systems can be determined by the deviation of the transient responses. Due to the complete state space-covering input stimuli bringing each of the systems to all of its reachable states, a formal equivalence checking methodology is given. As an advantage over other approaches, the level of abstraction between the two systems under verification is not restricted.

A tool for visualization of the state space and its dynamics by application of a particle simulation algorithm was developed for visually exploring the state space properties verified with the aforementioned formal verification approaches.

1.6 Publications

Parts of this thesis have been published in [SJH06, SH08a, SH08b, SPH09, SH09, SH10b]. The analog model checking approach based on the development of the analog specification language (ASL) is detailed in [SJH06, SH08b]. The applicability of ASL specifications to transient simulation waveforms transformed to a state space representation has been demonstrated in [SH09]. The foundations of state space-directed transient simulation for verification with complete-coverage input stimuli have been introduced in [SH08a] and their extension to an analog equivalence checking approach is presented in [SH10b]. Supporting the verification insight, state space visualization and visualized particle simulation is introduced in [SPH09].

1.7 Overview

The remainder of this thesis is organized as follows. Chapter 2 discusses approaches to system representation for verification with emphasis on discrete modeling of analog circuits which is essential for application of the verification algorithms developed in this thesis. In Chapter 3, approaches to property specification are analyzed with the goal of finally developing an analog specification language. The existing approaches to non-formal and formal system verification are described in Chapter 4 in order to introduce new methodologies for formal verification of analog circuits in Chapter 5. The experimental results obtained by application of the new verification methodologies to example circuits are discussed in Chapter 6. In Chapter 7, conclusions and suggestions for future work are given.

2

System Representation for Verification

Electronic systems can be hierarchically divided into several classes. The most general distinction is made between discontinuous and continuous systems, commonly referred to as digital and analog systems. While digital systems are characterized by discrete time steps with a clocked or event-based time domain, analog system classes are distinguished by the type of differential equation system required for describing their continuous behavior in time and values.

In contrast to experiment-based static or transient dynamic network analysis, analog formal verification techniques require a state space representation of the system dynamics which cannot be acquired in an analytical way. Hence, discrete modeling approaches for nonlinear analog systems are mandatory.

This chapter introduces the basic representations of electronic systems and the modeling methods for applying analysis and verification techniques. Emphasis is put on developing a new trajectory-directed discrete modeling approach for nonlinear analog circuits.

2.1 System Description

Starting with a more general system concept, the definition of analog circuits and their relevant characteristics is developed based on the classification of general system characteristics. Therefore, fundamentals such as the definition of signals and signal types, system types and the state space of a system will be introduced.

Definition 2.1.1 (System)
A system S maps a vector of inputs $\mathbf{u}$ and inner variables $\mathbf{x}$ to a vector of output variables $\mathbf{y}$ with the operator relation

$$\mathbf{y} = S(\mathbf{u}, \mathbf{x}) \tag{2.1}$$

as illustrated in Figure 2.1. The inputs represent the system excitation and the outputs represent the system reaction.

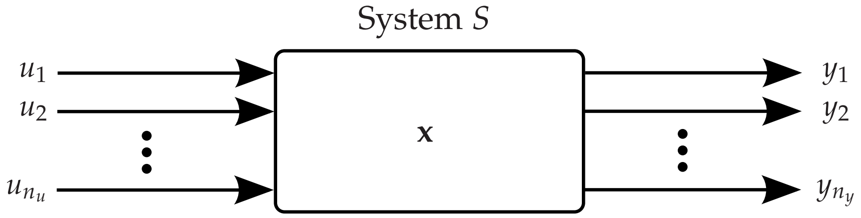


Figure 2.1: *General illustration of a system S with input variable vector $\mathbf{u} = (u_1, u_2, \ldots, u_{n_u})^T$, output variable vector $\mathbf{y} = (y_1, y_2, \ldots, y_{n_y})^T$ and the vector of inner variables $\mathbf{x} = (x_1, x_2, \ldots, x_{n_x})^T$.*

While this definition of a system defines the basic relation of system variables, a more detailed definition is needed for quantifying their characteristics in form of information. The information of the inputs and outputs of a system, as well as the inner variables, are in the following defined by signals.

Definition 2.1.2 (Signal)
A signal is the variation of information over another quantity such as time. Signals can be of time-continuous or time-discontinuous form, which means that the time domain of the signal function f is either defined as $f(t)$ with $t \in \mathbb{R}$ or $f[n]$ with integer-valued $n \in \mathbb{Z}$. An analog electrical signal is the continuous variation of an electrical quantity over time.

An important characteristic of a system is whether there is a dependency between the time when an input signal occurs and the reaction the system exhibits. For this purpose, the class of time-invariant systems has to be defined.

Definition 2.1.3 (Time-Invariant System)
The output of a time-invariant system does not depend on the absolute time of the occurrence of an input signal. If the input is shifted by time δ, a time shift of δ of the output occurs:

$$\mathbf{y}(t) = S(\mathbf{u}(t)) \Leftrightarrow \mathbf{y}(t+\delta) = S(\mathbf{u}(t+\delta)) \tag{2.2}$$

With the previous definition of time-invariant systems, it is now possible to distinguish two major classes of systems: static and dynamical systems.

Definition 2.1.4 (Static / Dynamical System)
If the output of a time-invariant system at each point in time is only depending on the current input at that time, it is a static (memoryless) system:

$$\mathbf{y}(t) = S(\mathbf{u}(t)) \tag{2.3}$$

If a system is depending on internal states that are part of the vector of inner variables $\mathbf{x}$ *determined by previous input values, it is a dynamical system:*

$$\mathbf{y}(t) = S(\mathbf{u}(t), \mathbf{x}(t)) \text{ with } \mathbf{x}(t) = \mathbf{f}(\mathbf{u}(\tau), \mathbf{x}(\tau)) \text{ for } 0 \leq \tau \leq t \tag{2.4}$$

The concept of a dynamical system is closely related to the idea of internal states of a system that will be further detailed in Section 2.3.6. Hence, there is a set of inner variables which are controllable by the input variables but depending on the previous evolvement of the inputs over time. The system reaction is therefore a function of the input variables and the configuration of these inner variables determining the state of the system within a state space as defined in the following.

Definition 2.1.5 ((Extended) State Space)
The state space of a system is spanned by a subset of its inner variables. This subset is the vector $\mathbf{z} \subseteq_\zeta \mathbf{x}$ *of* n_z *linear independent state space variables. The subset relation "*$\subseteq_\zeta$*" is defined by:*

$$\mathbf{z} \subseteq_\zeta \mathbf{x} \;\Leftrightarrow\; \forall\, i \in \{1, ..., n_z\} : z_i = x_{\zeta(i)} \tag{2.5}$$

A valid assignment of values to the state space variables represents a state of the system. The extended state space $\mathbf{z}^{(e)}$ *of a system is spanned by its state space variables* $\mathbf{z}$ *and the input variables* $\mathbf{u}$.

Preparing the terminology of analog circuits, the distinction between linear and nonlinear systems has a great impact on the problem complexity of circuit analysis techniques. Hence, the definition is as follows:

Definition 2.1.6 (Linear / Nonlinear System)
A system is linear if the linear combination of the system's reaction to input signals $u_1(t)$ *and* $u_2(t)$ *equals the system's reaction to the linear combination of the input signals:*

$$S(k_1 u_1(t) + k_2 u_2(t)) = k_1 S(u_1(t)) + k_2 S(u_2(t)) \tag{2.6}$$

hence

$$S(k\mathbf{u}(t)) = kS(\mathbf{u}(t)) \tag{2.7}$$

If this property does not apply, the system is considered as nonlinear.

2.2 Digital Systems

Although digital systems are not in the focus of this thesis, the state of the art in discrete modeling and specification for verification of analog circuits is based on the ideas developed for digital circuits. Therefore, a brief introduction into digital systems and their representation is given in order motivate the transfer of digital verification methods to the analog domain at a later stage.

A digital system is operating on discrete values internally and on its inputs and outputs. Digital refers to a finite number of input and output values and in the following digital systems are considered as binary digital systems with two signal levels abstracted by 0 and 1. The fundamental benefit of this binary approach is the possibility of a rigorous formulation of all digital system behavior by mathematical logic [Men01] for reasoning about the truth of a formula and Boolean algebra [Whi95] for combining and manipulating logic statements.

Every Boolean formula can be transferred into a representation of the basic Boolean operators AND, OR and INVERT. Hence, an arbitrary Boolean logic formula can be implemented as a digital circuit consisting of the hardware implementation of this basic operators called logic gates. Digital circuits can be distinguished into combinatorial and sequential circuits. A combinatorial circuit has a direct mapping of the input values to the outputs by the logic function implemented by the circuit. Hence, it represents a static system. A sequential circuit contains internal memories called flip-flops which in the most basic form can store one Boolean signal level and propagate it to its output until a new Boolean value is stored. Moreover, a feedback of the internal states through logic functions causes the internal states being dependent on the input signals and the previous internal state. Therewith, sequential circuits represent dynamic systems. Clocked sequential circuits only propagate signals when an external trigger called clock signal event occurs. Due to the internal states and the time behavior, sequential circuits are the more complex and powerful digital circuit class [KB05].

The behavior of sequential circuits can be modeled by finite state machines where the states represent an unique configuration of the internal state variables of the circuit and the transitions are labeled with the input combinations that cause a transition from a state to its successor state. Boolean functions representing sequential circuits can be easily transferred into finite state machines and vice versa via state transition tables.

The example circuit shown in Figure 2.2(a) is a two-bit down counter. Triggered by the clock signal *clk*, the outputs Q_1 and Q_0 of the two flip-flops cyclically count the binary values $11, 10, 01, 00$ when the input *enable* is set to 1. If *enable* is set to 0, the circuit remains in its current state. The state transition table for this behavior is shown in Table 2.1 with Q_1 and Q_0 being the outputs of the flip-flops and the next states of the counter denoted as Q_1^+ and Q_0^+.

Table 2.1: *State transition table for the two-bit down counter.*

enable	Q_1	Q_0	Q_1^+	Q_0^+
0	1	1	1	1
0	1	0	1	0
0	0	1	0	1
0	0	0	0	0
1	1	1	1	0
1	1	0	0	1
1	0	1	0	0
1	0	0	1	1

From the state transition table, the Boolean functions for Q_1^+ and Q_0^+ can be directly read by finding the Boolean function of Q_1 and Q_0 with the result Q_1^+ and Q_0^+, respectively:

$$Q_1^+ = enable \wedge (Q_1 \oplus \overline{Q_0}) \vee \overline{enable} \wedge Q_1 \tag{2.8}$$

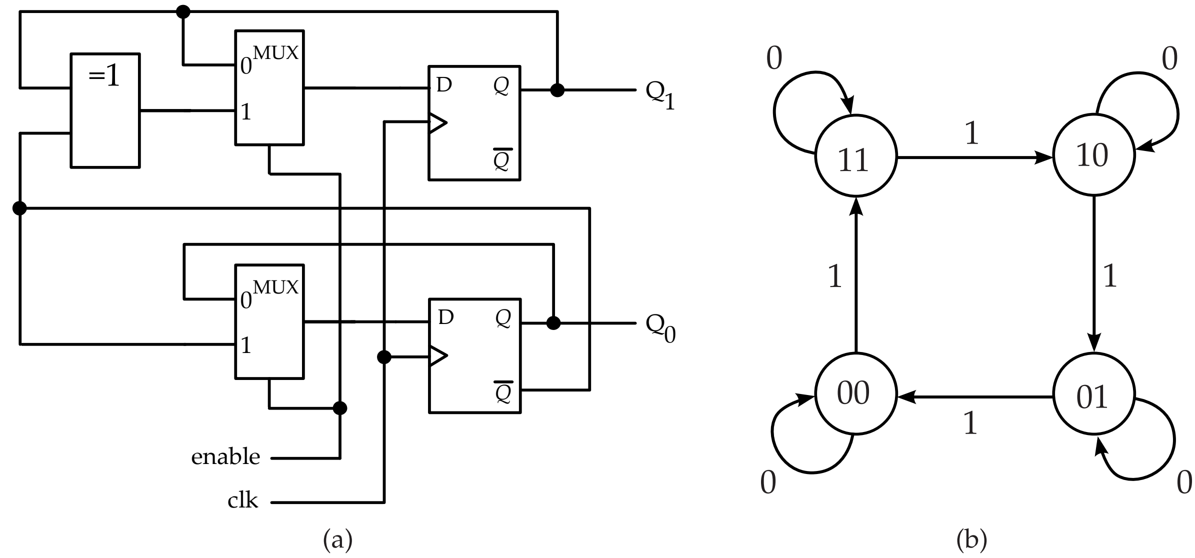
$$Q_0^+ = enable \wedge \overline{Q_0} \vee \overline{enable} \wedge Q_0 \tag{2.9}$$

The corresponding state transition graph is illustrated in Figure 2.2(b), with the edge labels representing the value of the input *enable*. The states are labeled with the values for Q_1 and Q_0 .

This state transition graph can, with some additions, directly serve as a circuit model which can be processed by digital formal verification tools using temporal logic property specifications [Pnu77, CE82] that will be described in Section 3.3. In the domain of discrete state system modeling for formal verification approaches, the Kripke structure [CGP99] is a common model combining a transition graph structure and a labeling of the states with atomic propositions for identifying sets of states where a certain proposition is true. The following definition of the Kripke structure is the theoretical model to which temporal logic property verification is applied.

Definition 2.2.1 (Kripke Structure)
For a set of atomic propositions AP, the Kripke structure M over AP is a four tuple $M = (\Sigma, \Sigma_0, R, L)$ where

- *Σ is a finite set of states of the system.*
- *$\Sigma_0 \subseteq \Sigma$ is the set of initial states.*
- *$R \subseteq \Sigma \times \Sigma$ is a total transition relation, hence for every state $\sigma \in \Sigma$ there exists a state σ' such that $(\sigma, \sigma') \in R$.*



(a) (b)

Figure 2.2: *Circuit schematic for the two-bit down counter (a) and the corresponding state transition graph (b).*

- $L : \Sigma \to 2^{\#AP}$ *is a labeling function that labels each state with the set of atomic propositions that are true in that state.*

Within the structure M, a path π *beginning at state* σ *is a sequence of states* $\pi = \sigma_0, \sigma_1, \sigma_2, ..., \sigma_n$ *with* $\sigma_0 = \sigma$ *and* $(\sigma_i, \sigma_{i+1}) \in R$ *for* $0 \leq i < n$.

However, the explicit representation of the states of a digital hardware system quickly exceeds what is efficiently manageable by computer memories. Consider a sequential system containing 64 flip-flops. The state machine describing this system can be estimated to contain 2^{64} states which corresponds to several exabytes of data even if it was possible to require only one byte of information per state for its representation. This problem was solved by an implicit state representation. Using a symbolic representation based on binary decision diagrams (BDDs), the state space can be described by a symbolic transition relation on which the verification algorithms can be evaluated [McM92]. This leads to a logarithmic decrease in state space representation complexity for digital systems.

2.3 Analog Systems

Analog systems are characterized by their continuous value and time domain. In the scope of this thesis, analog electronic circuits are nonlinear dynamic analog systems that are defined by the connection of the physical device models of the circuit elements via Kirchhoff's circuit laws [Kir47]. The methodology of computer-supported analog circuit analysis is based on the idea that there is an appropriate mathematical model of the physical system which can be analyzed using mathematical methods. The results of the analysis then correspond to the behavior of the real system.

The physical device models are mathematically described by differential equations. These differential equations of the circuit elements and their connection via Kirchhoff's laws form a differential algebraic equation (DAE) system describing the analog electronic circuit. To obtain a differential equation system from a circuit topology modeled by a network of circuit elements, network analysis techniques such as the modified nodal analysis are applied (see Section 2.3.2). The type of differential equation system set up for analysis of analog electronic circuit is defined by the equations of the physical device models of the circuit. Depending on the accuracy of the model and therewith affecting the accuracy of the analysis, several levels of model complexity can be available.

2.3.1 Device Models

Modeling the physical devices of an electronic circuit in form of mathematical equations is a basic necessity for analyzing analog circuits using mathematical methods. Therefore, depending on the purpose, compact models for circuit design or physical device models for device design can be distinguished. For electronic circuit simulation, only compact models are considered due to the fact that a large number of devices within a circuit has to be simulated simultaneously within a tolerable time frame. In contrast, for designing devices for fabrication processes, substantially more complex device models are required for simulation of subtle physical effects of one single device.

The main devices for integrated circuit design are resistors, capacitors, inductors, independent and controlled voltage and current sources, diodes and transistors. The most complex device behavior in integrated circuits is exhibited by transistors for which several compact model families exist. Due to its well-suited characteristics for integrated circuits, the metal-oxide-semiconductor field-effect transistor (MOSFET) is of great importance. Depending on the type of analysis to conduct and the trade-off between accuracy and simulation complexity, different types of compact models exist in form of physical models, empirical models and table models [Tsi03]. Table models are generated using measurements of real transistors by capturing the behavior at discrete parameter steps. Therewith, fast models are generated but for each combination of the width and length of the transistors, new tables have to be measured. Empirical models describe the captured characteristics by mathematical functions using curve fitting approaches.

The most flexible approach is given by physical models that can be adjusted in their level of complexity. This is achieved by setting up equations for the basic physical behavior and adding more and more physical parameters in each level. For a n-channel

MOSFET transistor, the drain-source current I_{DS} for the DC case has three regions of operation, dependent on the gate-source voltage V_{GS}, the threshold voltage V_{TH} and the drain-source voltage V_{DS}: cutoff ($V_{GS} \leq V_{TH}$), saturation ($0 < V_{GS} - V_{TH} \leq V_{DS}$) and linear ($0 < V_{DS} < V_{GS} - V_{TH}$). These three regions of operation can be quadratically approximated by the following equations [Vla94]:

$$I_{DS} = \begin{cases} 0 & \text{for } V_{GS} \leq V_{TH} \\ \frac{KP}{2}\frac{W}{L_{eff}}(V_{GS} - V_{TH})^2(1 + LAMBDA \cdot V_{DS}) & \text{for } 0 < V_{GS} - V_{TH} \leq V_{DS} \\ \frac{KP}{2}\frac{W}{L_{eff}}V_{DS}(2(V_{GS} - V_{TH}) - V_{DS})(1 + LAMBDA \cdot V_{DS}) & \\ & \text{for } 0 < V_{DS} < V_{GS} - V_{TH} \end{cases} \tag{2.10}$$

The parameters W and L_{eff} are the width and effective length of the transistor and KP and $LAMBDA$ are the MOSFET parameters of transconductance factor and output conductance factor in saturation.

This is a basic level 1 model for the static operation of the transistor which can be refined to a higher level model by adding more physical effects [CJL$^+$97]. For modeling the dynamic behavior, additionally the charge effects of the gate capacitance have to be considered, resulting in more complex equations [CHHK98] that improve the accuracy of the solutions of AC and TR analysis.

2.3.2 Network Analysis using the Modified Nodal Approach

In order to obtain a mathematical model for nonlinear analog circuits, modified nodal analysis (MNA) [HRB75] is used by most circuit analysis tools to set up the circuit equations as a DAE system. The MNA is based on three basic fundamentals:

- Kirchhoff's current law (KCL), stating that the sum of the currents flowing into a circuit node must equal the currents flowing out of this circuit node. Hence, their sum must be zero at any time.
- Kirchhoff's voltage law (KVL), stating that the sum of voltages around any closed loop of the circuit must be zero at any time.
- Branch constitutive equations (BCE), defining the mathematical model of the behavior of the physical circuit elements. For application of the MNA, the current of each BCE has to be described by a function of the connected node voltages and the device model of the respective circuit element.

Consider the circuit illustrated in Figure 2.3. It is a simple nonlinear analog circuit example with input voltage V_{in}, resistor R_1, diode D_1 and capacitor C_1. The KCL and the BCEs for the circuit elements are sufficient for setting up the network equations for nodes n_1 and n_2. The node voltages of the circuit represent the vector $\mathbf{x} = (V_{n_1}, V_{n_2})^T$

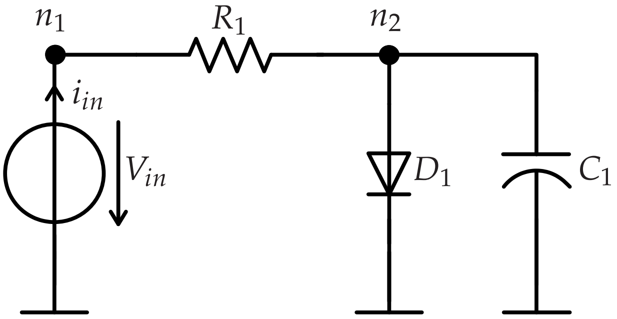


Figure 2.3: *Simple nonlinear analog circuit example with input voltage V_{in}, resistor R_1, diode D_1 and capacitor C_1.*

of unknowns of the equation system and the vector $\mathbf{u} = (V_{in})$ contains the inputs. The goal is to obtain a general formulation as a first-order nonlinear DAE system in form of:

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{u}) = \mathbf{0} \tag{2.11}$$

For node n_1, according to KCL, the sum of the input current i_{in} and the current through resistor R_1 has to be zero:

$$f_{n_1} : i_{in} - \frac{V_{n_1} - V_{n_2}}{R_1} = 0 \tag{2.12}$$

For node n_2, the ingoing current is the current through resistor R_1 and outgoing currents run through the parallel diode D_1 and the capacitor C_1. By filling in the device equations, the following node equation is set up:

$$f_{n_2} : \frac{V_{n_1} - V_{n_2}}{R_1} - I_s \left(e^{\frac{V_{n_2}}{u_T}} - 1 \right) - C_1 \cdot \frac{d}{dt} V_{n_2} = 0 \tag{2.13}$$

Finally, the voltage source sets the voltage V_{n_1} of node n_1 to V_{in}, resulting in:

$$V_{n_1} = V_{in} \tag{2.14}$$

Application of the MNA results in implicit equations for each circuit node with the system variables usually being the node voltages, some device currents and additional variables resulting from device equations or behavioral description of parts of the analog circuit. Another characteristic of the MNA is the high occurrence of algebraic equations and the occurrence of only some differential equations.

2.3.3 Numerical Simulation

By having set up the DAE system for the circuit, different numerical simulation techniques can be applied in order to analyze its behavior. The basic type of analysis is the

DC-analysis, where the operating point for a given constant input vector is calculated. Due to changes over time not being considered for the solution of the system in its steady state, the vector $\dot{\mathbf{x}}$ is zero. Hence, the equation system $\mathbf{f}(\mathbf{x}, \mathbf{u}) = 0$ for given $\mathbf{u}$ has to be solved. Due to the nonlinearity of the equation system in the general case, this is a mathematically challenging task.

The common algorithm used for numerically solving the nonlinear equation system is the Newton-Raphson (NR) iteration [Rap90, Ypm95]. Starting with a guess of an initial solution $\mathbf{x_0}$, this initial solution is iteratively refined until it falls below a specified error bound. However, there has to be a limit on the number of iterations as there is no guarantee that the NR-algorithm will succeed in finding a solution. On the other hand, in the neighborhood of a solution, the convergence is quadratic. For the NR-algorithm, the Jacobian matrix $\mathbf{J}$ containing the derivatives of the node equations for the system variables is needed:

$$\mathbf{J} = \begin{bmatrix} \frac{df_{n_1}}{dx_1} & \cdots & \frac{df_{n_1}}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{df_{n_n}}{dx_1} & \cdots & \frac{df_{n_n}}{dx_n} \end{bmatrix} \tag{2.15}$$

The NR-algorithm is based on the idea that the evaluation of $\mathbf{f}$ for an approximated $\mathbf{x}^{(a)}$ is related to the correct $\mathbf{x}$ by an amount $\Delta\mathbf{x}$ given by:

$$\Delta\mathbf{x} = -\mathbf{J}^{-1}(\mathbf{x}^{(a)})\mathbf{f}(\mathbf{x}^{(a)}) \tag{2.16}$$

Equation 2.16 can be solved using linear equation system approaches such as LU factorization [GVL96], which is a modification of the Gaussian elimination method in order to obtain a triangular system. Finally, the goal of the NR-algorithm is improving the initial guess for $\mathbf{x}$ in every iteration by $\mathbf{x_i} = \mathbf{x_{i-1}} + \Delta\mathbf{x}$ until the norm of the evaluated $\mathbf{f}(\mathbf{x})$ falls below the error bound and hence a sufficient solution is found.

For TR-analysis, a time domain solution for the circuit's transient response to arbitrary piecewise linear input stimuli has to be calculated [Nag75, Vla94]. This is achieved by dividing the simulation over time into a sequence of quasi-static solutions at time points $t_n, t_{n+1}, ...$ Consider a solution $\mathbf{x}(t_n)$ for the initial time point t_n obtained by the DC solution. The idea is to express the solution at t_{n+1} by a Taylor series approximation around the previous time point t_n and time step h either by the explicit forward Euler (FE) integration formula

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + h\dot{\mathbf{x}}(t_n) \tag{2.17}$$

or by the implicit backward Euler (BE) integration formula

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + h\dot{\mathbf{x}}(t_{n+1}) \tag{2.18}$$

Depending on the size of the chosen time step, the accuracy of the numerical integration is affected by the approximation of the first term of the Taylor series. Hence, the local truncation error (LTE) can be approximated by evaluating the second term of the Taylor series. Therewith, the LTE for the FE integration is given as:

$$\text{LTE} = \left| \frac{h^2}{2} \ddot{\mathbf{x}}(t_n) \right| \tag{2.19}$$

A combination of the FE and BE formula in form of

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \frac{1}{2} h(\dot{\mathbf{x}}(t_n) + \dot{\mathbf{x}}(t_{n+1})) \tag{2.20}$$

results in a better approximation, which can be proven by calculating the LTE. By evaluating the LTE, an adaptive time step control algorithm can be applied by calculating h according to a specified upper error bound [Vla94].

2.3.4 DAE Index

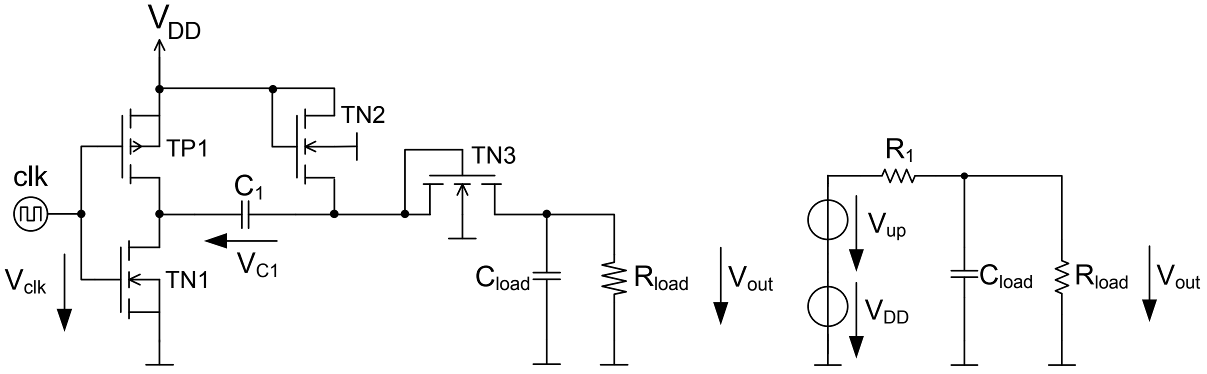
The differentiation index of a differential equation system determines its solvability. While ordinary differential equations (ODEs) have no algebraic variables and are of index 0, DAE systems contain algebraic variables and their index is of at least 1. Index-1-DAEs can be directly targeted by the numerical solution approach described in the previous subsection, while systems with an index greater than 1 have to be preprocessed by advanced techniques that are still subject to research [Tis96, ES00]. Due to only special cases of analog circuits being of index greater 1, the presented simulation approach is used in most circuit analysis tools. Nevertheless, approaches for higher index systems can be applied in the numerical simulation algorithms used in the remainder of this thesis.

2.3.5 Analog Behavioral Modeling and Hardware Description Languages

The description of analog circuits by a netlist containing the physical circuit components is directly transferable to a real physical design, as well as into a mathematical model for design analysis. Due to the abstraction hierarchy of the circuit design flow, dealing with transistor netlists is not appropriate at higher abstraction levels. Hence, building hierarchical entities of such low-level circuits allows a structuring of the design in higher levels. However, due to the large number of transistors, the representation on transistor level in conjunction with the complex mathematical analysis efforts described in the previous subsections render a transistor-level system analysis impossible for practical design flows.

Therefore, an abstraction of the structurally modeled low-level circuits in form of behavioral models is needed. Such behavioral models enable a higher-level analysis, capturing the important characteristics of the system building blocks with less computation time. In mathematical terms, behavioral modeling abstracts from the detailed complex equations of the BCEs and replaces them by simplified equations. These simplified equations capture the characteristics necessary for initial evaluation of the design on high abstraction levels. Of course, the abstractions cause the system based on behavioral models not to be transferable to a physical circuit level directly. However, the entities modeled on BCE level and those using behavioral descriptions can be interchanged.

Replacing structural modeling using BCEs with behavioral modeling, two common approaches are available. On the one hand, behavioral models can be created by describing more complex circuits with simplified structures that are still implemented on transistor level using BCEs. As an example, a charge pump circuit as illustrated in Figure 2.4(a) is considered. The transient startup behavior of the transistor netlist can be abstracted by a circuit as illustrated in Figure 2.4(b). While the transient response of the abstracted model retains the main behavior, the ripple originating from the clock switching is not present as illustrated in Figure 2.5. This approach, sometimes referred to as macro modeling, is often incorporated in higher level modeling in order to use the transistor netlist environment without invoking another circuit description approach.



(a) (b)

Figure 2.4: *Schematic of the CMOS implementation of the charge pump circuit (a) and simplified macro model (b).*

On the other hand, specific analog hardware description languages are available in order to efficiently describe analog circuit behavior using a programming-language like concept. Such hardware description languages have very sophisticated description methodologies. Depending on the complexity of the language syntax, simulation of the descriptions is possible by setting up a differential equation system from the code which then again can be numerically processed directly by the simulation algorithm detailed in Section 2.3.3. Popular analog hardware description languages are VHDL-AMS [APT03] and Verilog-AMS [KZ04], which both are extensions to the dig-

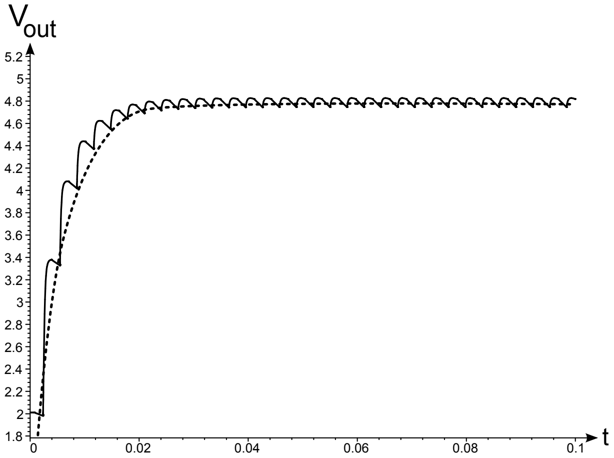


Figure 2.5: *Transient startup response of the transistor netlist charge pump (solid line) and the simplified model (dashed line).*

ital versions of the languages for description of analog and mixed analog/digital system behavior. The VHDL-AMS behavioral model for the charge pump is described in Listing 2.1, using an implicit differential equation for defining the startup-behavior of the output voltage.

Listing 2.1: *VHDL-AMS implementation modeling the charge pump behavior.*

```
LIBRARY DISCIPLINES;
USE DISCIPLINES.ELECTROMAGNETIC_SYSTEM.ALL;

ENTITY chargepumpmodel IS
  GENERIC( r1    :real := 1.0e5;
           rload :real := 1.0e9;
           c     :real := 5.0e-8;
           vdd   :real := 3.0);
  PORT(TERMINAL outp, gnd : ELECTRICAL );
END chargepumpmodel;

ARCHITECTURE behavior OF chargepumpmodel IS
  QUANTITY vout ACROSS iout THROUGH outp TO gnd;
BEGIN
  (1.6*vdd-vout)/r1 - c*vout'dot - vout/rload == 0.0;
END behavior;
```

2.3.6 State Space Representation

Based on the idea of a system's state space in Definition 2.1.5, the linear independent state space variables $\mathbf{z} = (z_1, ..., z_{n_z})^T$ of the implicit DAE system with $\mathbf{z} \subseteq_\zeta \mathbf{x}$ span a subspace of $\mathbb{R}^n$, representing the state space of an analog system. The number of independent state variables is not always clear, e.g. due to capacitor loops [ES00]. Additionally, extracted netlists have lots of resistor-capacitor paths leading to many state variables which may not all be of interest for the main input-output behavior. The state space variables of the analog circuit are given by the representation of the linear independent energy storing elements of the circuit such as capacitors and inductors in the differential equation system set up for the circuit by network analysis. Depending on the level of modeling abstraction, additionally, parasitic capacitances can be considered.

Candidates $\mathbf{z}^{(all)} \subseteq_\zeta \mathbf{x}$ for state space variables can be identified in the DAE system by their occurrence as first-order time derivatives. This is due to the BCEs of the circuit elements containing first-order time derivatives such as the current I_{Cap} through a capacitor Cap with capacity C is given by $I_{Cap} = C \cdot \frac{d}{dt} V_{Cap}$. Similarly, inductors introduce inductor currents as a state space variable. Capacitor loops and their dual equivalent of inductor nodes lead to linear dependencies, reducing the number of linear independent state space variables. The linear independent state space variables $\mathbf{z}$ are a subset of $\mathbf{z}^{(all)}$, hence $\mathbf{z} \subseteq_\zeta \mathbf{z}^{(all)} \subseteq_\zeta \mathbf{x}$.

If there are capacitors within the circuit that are not connected to ground nodes, the MNA generates two derived node voltages, one for each node the capacitor is connected to. In order to obtain only one state variable for the capacitor, the nodal analysis can be changed to a charge-oriented equation formulation [ST00].

The extended state space $\mathbf{z}^{(e)}$ of an analog system is spanned by the linear independent state space variables $\mathbf{z}$ and the input variables $\mathbf{u}$ with dimension $n_d = n_z + n_u$:

$$\mathbf{z}^{(e)} = \begin{bmatrix} \mathbf{z} \\ \mathbf{u} \end{bmatrix} \tag{2.21}$$

2.4 Discrete Modeling of Analog Systems

The value and time continuous characteristics of analog circuits require a special representation for application of formal methods. While in the digital circuit domain automata models are a common approach for system representation, analog circuits are not easily transferable to such a discrete model. Digital systems have an enumerable finite number of states which can be directly transferred into a finite state machine automata representation. Formal verification algorithms can check this set of states completely in order to identify whether there exist states that violate the specification.

In contrast, the continuous state space of analog systems is not enumerable. Hence, for completely checking their behavior, either an analytical approach reasoning on the continuous model or a discretization to a state model is required at the cost of introducing a discretization error. Due to the difficult solution of nonlinear DAE systems, analytical approaches are not feasible. Therefore, a discrete modeling based on numerical sampling-based algorithms is necessary for state space-based formal verification of analog circuits. In the following, the requirements of a discrete modeling will be discussed and after analyzing the only state-of-the-art approach operating on DAE systems [Har03], a new discrete modeling algorithm will be introduced. Other approaches from the domain of hybrid system analysis, operating on ODEs, that either do not offer a complete representation of the state space or cannot automatically generate models from circuit descriptions will be discussed in Section 4.5.1 and Section 4.5.2.

2.4.1 Discrete Analog Transition Structure

In the domain of discrete state system modeling for formal verification approaches, the Kripke structure as described in Definition 2.2.1 is a common model combining a timed automaton and a labeling of the states with atomic propositions for identifying sets of states where a certain proposition is true. In order to generate a discrete model of an analog circuit for verification purposes, the Kripke structure can be extended to a discrete analog transition structure (DATS) which incorporates the following additional information needed for describing an analog system:

- The states of the DATS represent value combinations of the extended state space variables of an analog system. Therefore, an extended labeling of the states has to assign this extended state space variable value vector to each state. Additionally, the values of the algebraic variables of the DAE system in this state are stored.

- As the structure of the DATS is determined by discretization algorithms described in the following subsections, the transition times between states cannot be considered equal like in transition systems for synchronous clocked digital systems. Hence, the transitions of the DATS have to be labeled with real valued transition times. As the discretization algorithms will only describe the start and end points of the transitions with the intermediate behavior considered as a linear trajectory, the transition time is considered as a linear change from the variable vector of the initial state to its successor described by the transition relation. Therewith, a transition sequence within the DATS corresponds to the idea of a piecewise linear trajectory obtained from transient analysis of analog circuits where time steps represent the sampled state variable values which are connected by piecewise linear transitions.

- The atomic propositions for states identify sets of states where these propositions are true. In addition, a selection on the transitions will be necessary in order to identify transitions not introduced by dynamic transitions in the state space but from an input variable model. Therefore, a labeling of the transitions by propositions is necessary for the DATS.
- The analog system model does not need to identify a set of initial states. The set of initial states for an operation on the DATS will be determined by an atomic proposition.

With these considerations, the DATS can be defined as follows.

Definition 2.4.1 (Discrete Analog Transition Structure (DATS))
For a set of atomic state propositions AP and a set of atomic transition propositions TP, the DATS M_{DATS} over AP, TP is a seven-tuple $M_{DATS} = (\Sigma, R, L_A, L_V, L_X, T, L_T)$ where

- *Σ is a finite set of states of the system.*
- *$R \subseteq \Sigma \times \Sigma$ is a total transition relation, hence for every state $\sigma \in \Sigma$ there exists a state σ' such that $(\sigma, \sigma') \in R$.*
- *$L_A : \Sigma \rightarrow 2^{\#AP}$ is a labeling function that labels each state with the set of atomic propositions that are true in that state.*
- *$L_V : \Sigma \rightarrow \mathbb{R}^{n_d}$ is a labeling function that labels each state with the vector of n_d variables containing the values in this state of the extended state space variables $\mathbf{z}^{(e)}$ of the DAE system.*
- *$L_X : \Sigma \rightarrow \mathbb{R}^{n_x}$ is a labeling function that labels each state with the vector of n_x variables containing the values in this state of the inner variables $\mathbf{x}$ of the DAE system.*
- *$T : R \rightarrow \mathbb{R}_0^+$ is a labeling function that labels each transition from σ to σ' with a real valued positive or zero transition time that represents the time required for the trajectory in the extended state space between these states.*
- *$L_T : R \rightarrow 2^{\#TP}$ is a labeling function that labels each transition with the set of atomic transition propositions that are true for that transition. This labeling will be used in Section 2.4.4.4 for distinguishing between dynamic transitions and those transitions that are introduced into the DATS by an input model.*

Within the structure M_{DATS}, a path π beginning at state σ is a sequence of states $\pi = \sigma_0, \sigma_1, \sigma_2, ..., \sigma_n$ with $\sigma_0 = \sigma$ and $(\sigma_i, \sigma_{i+1}) \in R$ for $0 \leq i < n$.

Figure 2.6 illustrates a schematic graph structure representing a DATS with nine vertices modeling an imaginary analog circuit. The DC-operating-points of the modeled circuit, that can be identified by DC-analysis as introduced in Section 2.3.3, are represented by vertices 1, 5 and 9. Thus, they have a loop transition to themselves stating that the circuit stays in this steady state infinitely until a change of input occurs. These transitions have zero transition time. A transition induced by an input change is modeled by a bidirectional edge, implying that this transition can only be taken if there is an input variable change in the corresponding extended state space dimension. Any non-steady state of the system has outgoing directed edges representing the dynamic behavior of the circuit.

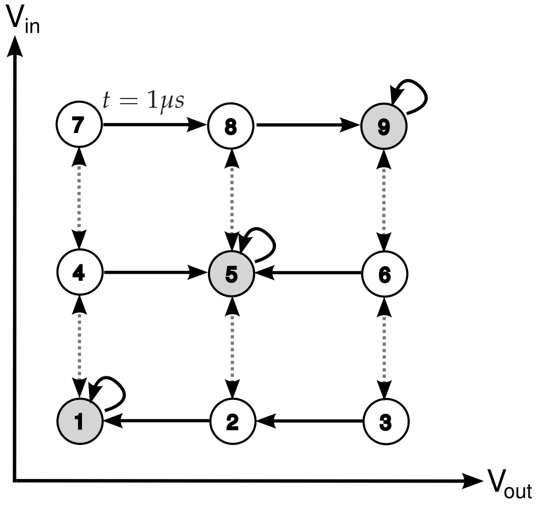


Figure 2.6: *Schematic illustration of a graph structure representing a DATS.*

2.4.2 The Discretization Problem for Analog Circuits

Discrete model generation for analog circuits is the key to applying graph-oriented verification algorithms that will be introduced in Section 3.4 and Chapter 5.

Definition 2.4.2 (Discrete State Space Modeling Task)
The task of discrete state space modeling is to transfer a continuous analog system represented as a DAE system into a DATS:

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{u}) = 0 \overset{\text{discrete modeling}}{\longrightarrow} M_{DATS} \tag{2.22}$$

Therefore, the continuous vector field of the time derivatives of the state space variables that are representing the dynamics of the analog circuit has to be partitioned. As will be detailed in the following, each partition is represented by a state of the DATS model with a transition relation connecting the states corresponding to the dynamic behavior of the circuit. However, the quality of the discretization determines how significant the verification results are. Therefore, there is an optimization problem connected to the discrete modeling task which has to capture the continuous behavior of the DAE system with a minimal discretization error.

Consider an infinite point set $\mathcal{Z}$ of points $\mathbf{p}$ in $\mathbb{R}^{n_d}$ of the state space of the circuit that is constrained to user-defined interval boundaries $r = [\underline{r}, \overline{r}]$ for every of the n_d extended state space dimensions:

$$\mathcal{Z} = \{\mathbf{p} \,|\, \underline{r}_i \leq p_i \leq \overline{r}_i\} \text{ for all } 1 \leq i \leq n_d \tag{2.23}$$

On $\mathcal{Z}$, the continuous vector field $\mathcal{V} : \mathbb{R}^{n_d} \rightarrow \mathbb{R}^{n_d}$ is generated by the time derivatives of the state space variables in the vector of extended state space variables

$$\dot{\mathbf{z}}^{(e)} = \begin{bmatrix} \dot{\mathbf{z}} \\ \mathbf{0} \end{bmatrix} \tag{2.24}$$

of the DAE system describing the circuit:

$$\mathcal{V}(\mathbf{z}^{(e)}) = \{\dot{\mathbf{z}}^{(e)} \,|\, \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{u}) = 0\} \tag{2.25}$$

The vectors $\mathbf{v}_i$ are defining a linearized trajectory in $\mathcal{V}$ from points $\mathbf{p}_i$ to the points $\mathbf{p'}_i$ with $\mathbf{v}_i = \mathbf{p'}_i - \mathbf{p}_i$, calculated by a time step-controlled transient simulation, as described in Section 2.3.3, starting in $\mathbf{p}_i$ with integration time Δt.

The goal is to generate a partitioning of $\mathcal{Z}$ to k non-overlapping partitions $\mathcal{R}_j$ with

$$\bigcup_{1 \leq j \leq k} \mathcal{R}_j = \mathcal{Z} \tag{2.26}$$

such that the inhomogeneity of the vector field flow within each $\mathcal{R}_j$ is minimal. Each $\mathcal{R}_j$ will represent a state σ_j of the DATS. The concept of inhomogeneity is defined by two criteria which are for a given integration time

1. the difference in direction
2. and the length difference

of the infinite set of trajectory vectors in $\mathcal{R}_j$.

Let θ_{rs} with

$$\theta_{rs} = \arccos\left(\frac{\mathbf{v}_r \cdot \mathbf{v}_s}{\|\mathbf{v}_r\|\|\mathbf{v}_s\|}\right) \tag{2.27}$$

be the angle between any of two sampled transition vectors $\mathbf{v}_r$ and $\mathbf{v}_s$ starting in $\mathcal{R}_j$.

Definition 2.4.3 (Direction Error)
The maximum direction error $\epsilon_\theta^{(\mathcal{R}_j)}$ *represented by the maximum angle between some* $\mathbf{v}_r$ *and* $\mathbf{v}_s$ *is given by:*

$$\epsilon_\theta^{(\mathcal{R}_j)} = \max(\theta_{rs} : \mathbf{p}_r, \mathbf{p}_s \in \mathcal{R}_j) \tag{2.28}$$

The overall mean direction error over $\mathcal{Z}$ is then defined by:

$$\epsilon_\theta = \frac{1}{k} \sum_{1 \leq j \leq k} \epsilon_\theta^{(\mathcal{R}_j)} \tag{2.29}$$

Let Δ_{rs} with

$$\Delta_{rs} = \max\left(\frac{\|\mathbf{v}_r\|}{\|\mathbf{v}_s\|}, \frac{\|\mathbf{v}_s\|}{\|\mathbf{v}_r\|} \right) \tag{2.30}$$

be the length difference ratio between any of two sampled transition vectors $\mathbf{v}_r$ and $\mathbf{v}_s$ starting in $\mathcal{R}_j$.

Definition 2.4.4 (Length Error)
The maximum length error $\epsilon_\Delta^{(\mathcal{R}_j)}$ *between some* $\mathbf{v}_r$ *and* $\mathbf{v}_s$ *is given by:*

$$\epsilon_\Delta^{(\mathcal{R}_j)} = \max(\Delta_{rs} : \mathbf{p}_r, \mathbf{p}_s \in \mathcal{R}_j) \tag{2.31}$$

The overall mean length error over $\mathcal{Z}$ is then defined by:

$$\epsilon_\Delta = \frac{1}{k} \sum_{1 \leq j \leq k} \epsilon_\Delta^{(\mathcal{R}_j)} \tag{2.32}$$

While minimizing the overall mean length and direction error are optimization criteria for obtaining an accurate discretization, for describing the optimization problem of state space discretization, a third optimization criterion, which is the number of partitions, has to be considered. Increasing the number of partitions and therewith decreasing the size of the partitions has negative effects on the efficiency of the verification algorithms. Moreover, in a n-dimensional state space, decreasing the size of every partition to half of its initial size in every dimension, the number of partitions increases by the factor 2^n. Therefore, keeping the number of partitions as small as possible is critical for developing feasible discretization algorithms.

Another important part is the determinism of the successor relation between adjacent partitions. If the partitions are perfectly enclosing homogeneous state space dynamics, every trajectory starting in a partition ends in a single adjacent partition. Hence, the out-degree $\deg(\mathcal{R}_j)$ of dynamic transitions ending in other partitions shall be minimal for each partition, ideally being 1. Additionally, by transferring the partitioning to a DATS, the successor relation between states representing the partitions determines the paths in the DATS. As the state space variable vectors of the states in the DATS are determined by the centers of the partitions, a sequence of transitions in the DATS corresponds to a trajectory in the state space. If the successor relation of the DATS is determined inaccurately, the behavior of the model does not correspond to the real state space trajectories. Therewith, a successor relation error has to be defined.

Consider two adjacent partitions $\mathcal{R}_i$ and $\mathcal{R}_j$ represented by states σ_i and σ_j, connected by a transition $(\sigma_i, \sigma_j) \in R$ with center points $L_V(\sigma_i) = \mathbf{p}_i^{(c)}$ and $L_V(\sigma_j) = \mathbf{p}_j^{(c)}$ and the vectors

$$\mathbf{v}_{ij}^{(c)} = \mathbf{p}_j^{(c)} - \mathbf{p}_i^{(c)} \tag{2.33}$$

and

$$\mathbf{v}_i^{(tr)} = \mathbf{p}'^{(c)}_i - \mathbf{p}_i^{(c)} \tag{2.34}$$

with $\mathbf{v}_i^{(tr)}$ determined by a transient step of length $\|\mathbf{v}_{ij}^{(c)}\|$ starting in $\mathbf{p}_i^{(c)}$.

Definition 2.4.5 (Successor Relation Error)
The successor relation error $\epsilon_{suc}^{(ij)}$ between two connected adjacent partitions $\mathcal{R}_i$ and $\mathcal{R}_j$ is defined by:

$$\epsilon_{suc}^{(ij)} = \arccos\left(\frac{\mathbf{v}_{ij}^{(c)} \cdot \mathbf{v}_i^{(tr)}}{\|\mathbf{v}_{ij}^{(c)}\| \|\mathbf{v}_i^{(tr)}\|}\right) \tag{2.35}$$

The overall mean successor relation error for a given partitioning is then defined by:

$$\epsilon_{suc} = \frac{1}{k} \sum_{1 \le i \le k} \max \{\epsilon_{suc}^{(ij)} | (\sigma_i, \sigma_j) \in R\} \text{ with } 1 \le j \le k \tag{2.36}$$

Definition 2.4.6 (Optimization Problem for the Discrete Modeling Task)
Based on the definition of

- *the direction error $\epsilon_\theta^{(\mathcal{R}_j)}$,*
- *the length error $\epsilon_\Delta^{(\mathcal{R}_j)}$,*
- *the number of partitions k,*
- *the determinism $\deg(\mathcal{R}_j)$ for all $1 \le j \le k$ of the successor relations,*
- *and the overall mean successor relation error ϵ_{suc},*

the multi-objective optimization problem connected to partitioning $\mathcal{Z}$ into $\mathcal{R}_j$ with $1 \le j \le k$ can now be stated with user defined maximum error bounds r_θ and r_Δ and a minimum number of partitions r_k:

$$\{\mathcal{R}_1, ..., \mathcal{R}_j, ..., \mathcal{R}_k\} = \arg\min \left\{ \begin{array}{l} \frac{1}{k} \sum\limits_{1 \le j \le k} \deg(\mathcal{R}_j) \\ \epsilon_{suc} \\ k \end{array} \right\} \text{ s.t. } \left\{ \begin{array}{l} \forall 1 \le j \le k : \epsilon_\theta^{(\mathcal{R}_j)} < r_\theta \\ \forall 1 \le j \le k : \epsilon_\Delta^{(\mathcal{R}_j)} < r_\Delta \\ k > r_k \end{array} \right\} \tag{2.37}$$

This optimization problem illustrates which criteria have to be considered for algorithmic approaches. Therewith, an algorithm that defines how the partitioning of the state space vector field has to be constructed is needed to meet the challenge.

2.4.3 Hyperbox Discretization

The only state-of-the-art approach that can transfer a circuit represented by a nonlinear DAE system into a discrete graph structure for application of verification algorithms is presented in [Har03] and recapitulated in the following.

A paraxial binary slicing algorithm is used to partition the state space in form of an infinite point set $\mathcal{Z}$ of the analog system. $\mathcal{Z}$ is constrained to user-defined interval boundaries r for every of the n_d extended state space dimensions:

$$\mathcal{Z} = \{\mathbf{p} \,|\, \underline{r}_i \leq p_i < \overline{r}_i\} \text{ for all } 1 \leq i \leq n_d \tag{2.38}$$

The slicing algorithm determines partitions $\mathcal{R}_j$ with paraxial boundaries s representing hyperboxes for each state space dimension:

$$\mathcal{R}_j = \{\mathbf{p} \,|\, \underline{s}_i^{(j)} \leq p_i < \overline{s}_i^{(j)}\} \text{ for all } 1 \leq i \leq n_d \tag{2.39}$$

The algorithmic approach to obtain this partitioning is sampling the state space with randomly distributed test points for which a transition step is calculated. By comparing the transition vectors of the transient steps to predefined error margins for length and direction, it is decided whether the vector field flow is homogeneous enough. If the error margins are exceeded, the state space partition is split into two partitions using a binary paraxial partitioning. For each of the two resulting partitions, the length and direction error is again checked and, if above the error margin, split again. This process is continued recursively until a predefined maximum recursion depth is reached. Each resulting hyperbox is considered as a state of the discrete model representing the analog system, and a tree structure with the hyperbox containing the complete state space as root node and the binary partitions hierarchically forming the children is created. The states of the analog system model are the leaf nodes of the tree structure.

In order to obtain a discrete transition system, the transition relation between the created hyperboxes from the partitioning algorithm has to be determined. The exact calculation of the geometric structure $\mathcal{R}_{exact}$ representing the successor of a hyperbox $\mathcal{R}_{test}$ would be obtained by a point-to-point mapping from points $\mathbf{p}$ to $\mathbf{p}'$:

$$\mathcal{R}_{exact} = \{\mathbf{p}' \,|\, \mathbf{p} \in \mathcal{R}_{test}\} \tag{2.40}$$

This exact mapping would result in the impossibility to create the simple state space partitioning geometry of the hyperbox discretization. Hence, for each hyperbox, the successors shall be those adjacent hyperboxes that enclose the exact geometric structure of the successor relation. This is achieved again by using transient simulation steps for a discrete number $n_{testpoints}$ of test points $\mathbf{p}_i$ within a box. All those neighboring boxes $\mathcal{R}_j$ which can be reached by transient simulation steps to points $\mathbf{p}_i'$ that start in $\mathbf{p}_i$ in the box $\mathcal{R}_{test}$ for which the successors shall be determined, are a successor of this investigated box:

$$\mathcal{R}_{succ} = \bigcup_j \{\mathcal{R}_j \mid \mathbf{p}_i' \in \mathcal{R}_j \wedge \mathbf{p}_i \in \mathcal{R}_{test}\} \text{ with } 1 \leq i \leq n_{testpoints} \tag{2.41}$$

Therewith, in general, an over-approximation of the analog transition relation is generated due to the hyperboxes forming $\mathcal{R}_{succ}$ enclosing the more complex geometric structure $\mathcal{R}_{exact}$ being the convex hull of all computed $\mathbf{p}_i'$ for $n_{testpoints} \rightarrow \infty$. Due to the number of test points being limited in order to reduce the computational effort, an under-approximation of the transition relation is possible as well.

While this approach of state space discretization for analog circuits has proven to be robust and algorithmically very manageable, there are some downsides that motivate the search for an improved, yet algorithmically not too complex approach. The main problem of the approach is due to the paraxial slicing of the state space. Therewith, the discretization is not rotation invariant to the vector field which results in an insufficient capture of the vector field flow dynamics within the state space and in large out-degrees $\deg(\mathcal{R}_j)$.

Considering for example the flow in Figure 2.7(a) which is parallel to the axis x_1. A trajectory from point $\mathbf{p}_1$ to $\mathbf{p}_2$ in the continuous flow maps to the transition illustrated in Figure 2.7(b) from box d_1 to d_2 with a deterministic successor representing a good discrete approximation of the vector field flow. By rotating the vector field by 45 degrees in Figure 2.7(c), the trajectory from $\mathbf{p}_3$ to $\mathbf{p}_4$ maps to the massively over-approximated nondeterministic transition paths illustrated in Figure 2.7(d) from box d_3 to d_4. Due to the paraxial slicing, the calculation of the subsequent boxes reports three successors enclosing the exact successor geometry of each box. This leads to a large number of possible paths and hence a substantial over-approximation of the reachable area which is not accurately representing the dynamic system behavior for practical verification purposes. While this over-approximation is considerable for pessimistic reachability computation for safety verification in order to prove that a bad state will never be reached, false negatives have to be expected.

2.4.4 Trajectory-Directed Discretization

In order to overcome the limitations of the hyperbox discretization approach described in the previous section, the desired behavior of an improved discretization algorithm has to be characterized.

The discretization shall be rotation invariant and therefore the state space intersections for partitioning cannot be paraxial. As a massive over-approximation of the

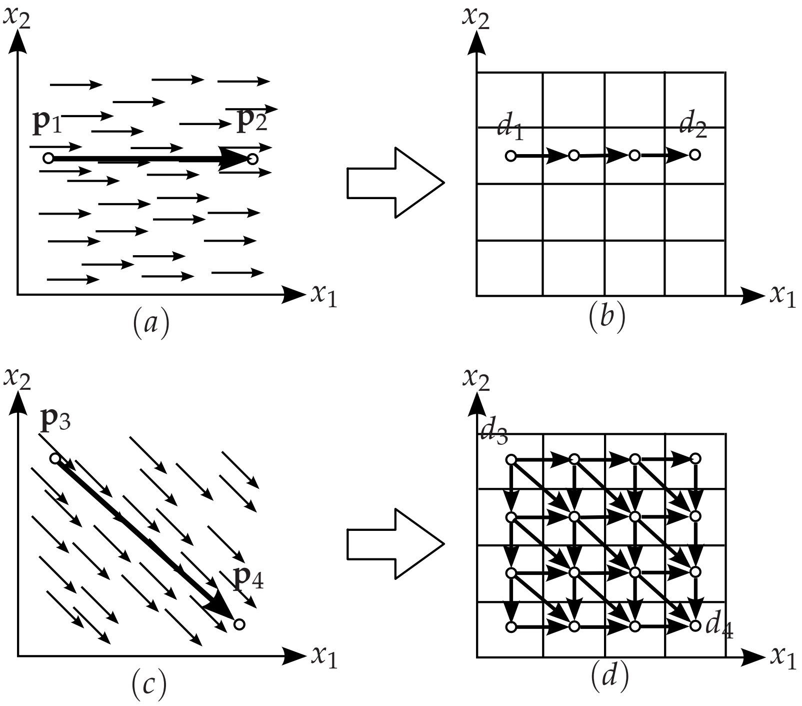


Figure 2.7: *Illustration of a vector field with paraxial flow (a) and the corresponding discretization (b) including the transition path. In comparison, the vector field flow rotated by 45 degrees (c) results in a massive over-approximation of the transition path in the hyperbox discretization (d).*

successor relation of the state space partitions significantly weakens the expressiveness of the verification algorithms, the geometric structure of the state space partitions shall follow the flow of the state space dynamics. Hence, the intersections dividing the state space shall be either parallel or orthogonal to the state space trajectories enclosed by the partitions. Therewith, the nondeterminism of the successor relation of the state space partitions shall be minimized due to the uniqueness of the successor relation being determined by the real trajectories between the preceding state space partition and its successor. By using time step control algorithms for determining the acceptable trajectory length which can be approximated by a straight line between two points in state space, the homogeneity of the enclosed state space dynamics in the partitions shall be guaranteed. Figure 2.8 illustrates a non-paraxial trajectory-directed state space partitioning applied to the example flow from Figure 2.7(c).

2.4.4.1 Calculating the State Space Partitioning

The central idea for a new discretization algorithm is that the intersections of the state space are no longer determined by paraxial slicing but by the trajectories of the state

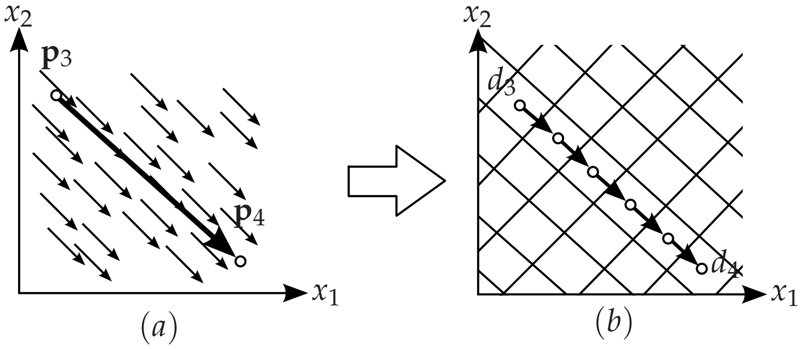


Figure 2.8: *Illustration of a non-paraxial vector field flow with a trajectory-directed state space partitioning resulting in a transition path from d_3 to d_4 (b) matching the initial trajectory between $\mathbf{p}_3$ and $\mathbf{p}_4$ (a).*

space dynamics. Hence, starting from the infinite point set $\mathcal{Z}$ constrained to user-defined interval boundaries r for each of the n_d extended state space dimensions as defined in Equation 2.23 representing the state space of the analog system, a slicing structure into k non-overlapping state space regions $\mathcal{R}_j$ of the state space shall be constructed:

$$\bigcup_{1\leq j\leq k} \mathcal{R}_j = \mathcal{Z} \tag{2.42}$$

A linear transformation of $\mathcal{Z}$ is necessary in order to handle size differences of the ranges of the state space variables. All ranges shall be translated and normalized to the interval $[0,1]$. Therefore, a translation vector $\mathbf{v}^{(t)}$ has to be calculated to move the lower bound of the n_d extended state space variable ranges $\underline{r}_i$ to 0 and to scale them to the interval $[0,1]$ by factors λ_i:

$$\lambda_i = (\overline{r}_i - \underline{r}_i)^{-1} \tag{2.43}$$

$$\mathbf{v}^{(t)} = \begin{bmatrix} -\underline{r}_1 \cdot \lambda_1 \\ \vdots \\ -\underline{r}_{n_d} \cdot \lambda_{n_d} \end{bmatrix} \tag{2.44}$$

The transformation matrix $\mathbf{T}$ for the scaling vector λ and the translation vector $\mathbf{v}^{(t)}$ in homogeneous coordinates [Mir95] is then given by:

$$\mathbf{T} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 & v_1^{(t)} \\ 0 & \lambda_2 & \ddots & \vdots & v_2^{(t)} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ \vdots & & \ddots & \lambda_{n_d} & v_{n_d}^{(t)} \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \tag{2.45}$$

The linear transformation for a point $\mathbf{p}^{(orig)}$ to $\mathbf{p}$ with

$$\begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \mathbf{T} \cdot \begin{bmatrix} \mathbf{p}^{(orig)} \\ 1 \end{bmatrix} \tag{2.46}$$

is applied to all following coordinate calculations and reversed for determining the final structure in the untransformed space. The 1-entry in position $n_d + 1$ of the vectors for using homogeneous coordinates is added or removed correspondingly.

After the preceding preparations, the actual partitioning can be described. The goal of the partitioning algorithm is to determine the vertices and edges of the geometric objects partitioning $\mathcal{Z}$ into the k $\mathcal{R}_j$. Therefore, the algorithm starts from a random starting point that is no DC-operating-point of the system by appending it to an initially empty waiting list WL. DC-operating-points are detected by a threshold level r_{DC} where the ratio between the norm of the transient step vector and the used integration time falls below this value.

For every point $\mathbf{p}$ in the waiting list, a step-length controlled transition vector $\mathbf{v}$ to the point $\mathbf{p}'$ with $\mathbf{v} = \mathbf{p}' - \mathbf{p}$ is calculated using a transient simulation back-end. In order to identify new starting points for transition vectors, across each vector $\mathbf{v}$ an orthogonal basis vector set $\mathbf{B}$ is constructed:

$$\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_{n_d} \,:\, \mathbf{b}_i \cdot \mathbf{b}_j = 0\} \text{ for all } 1 \leq i, j \leq n_d;\ i \neq j;\ \mathbf{v} \in \mathbf{B} \tag{2.47}$$

$\mathbf{B}$ is constructed using the Gram-Schmidt orthogonalization algorithm [GVL96]. The input to the Gram-Schmidt algorithm

$$\mathbf{B} = \mathbf{GramSchmidt}(\mathbf{M}) \tag{2.48}$$

is the matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{v} & \mathbf{i}_1 & \cdots & \mathbf{i}_{j-1} & \mathbf{i}_{j+1} & \cdots & \mathbf{i}_{n_d} \end{bmatrix} \tag{2.49}$$

consisting of the vector $\mathbf{v}$ and $n_d - 1$ of the n_d column vectors of the unity matrix $\mathbf{I}_{n_d}$ such that the eliminated vector $\mathbf{i}_j$ has its 1-entry in the same dimension j where $\mathbf{v}$ has its maximum absolute magnitude $|v_j|$:

$$j = \arg \max_{1 \leq j \leq n_d} |v_j| \tag{2.50}$$

The vectors returned by the algorithm are normalized to length 1. The resulting orthogonal basis set is now scaled to the initial length of $\mathbf{v}$. Additionally, in order to control the discretization error, for each element $\mathbf{b}_i$ of $\mathbf{B}$ a scaling factor $\beta_i^{(a)}$ is calculated. This $\beta_i^{(a)}$ shall assure that the direction and length differences between two transition vectors $\mathbf{v}_r$ and $\mathbf{v}_s$, with $\mathbf{v}_r$ being calculated starting from $\mathbf{p} + \beta_i^{(a)} \cdot \mathbf{b}_i$ and $\mathbf{v}_s$

being the initial transient vector starting from $\mathbf{p}$, are below predefined tolerance levels r_θ and r_Δ:

$$\theta_{rs} < r_\theta \wedge \Delta_{rs} < r_\Delta \tag{2.51}$$

Starting with $\beta_i^{(a)} = 1$, the scaling algorithm compares the two transition vectors $\mathbf{v}_r$ and $\mathbf{v}_s$ and iteratively halves the length of $\beta_i^{(a)}$ until the error criteria are satisfied. Therewith, just as the time step control using the LTE for controlling the error of transient simulation that was discussed in Section 2.3.3, a state space step control is applied for controlling $\epsilon_\theta^{(\mathcal{R}_j)}$ and $\epsilon_\Delta^{(\mathcal{R}_j)}$ within each state space partition $\mathcal{R}_j$, determined by the correspondingly scaled $\mathbf{B}$ and the adjacent orthogonal sets.

Satisfying r_θ and r_Δ in the state space area around singularities such as attractors represented by DC-operating-points would cause the length of the vectors $\beta_i^{(a)} \cdot \mathbf{b}_i$ to be decreased infinitely. Therefore, the previously mentioned threshold r_{DC} controls the minimum length of the vectors for obtaining a finite set of partitions.

Each of the scaled orthogonal basis set vectors added to $\mathbf{p}$ describes starting points $\mathbf{q}^{(a)}$ for a new transient step calculation for which in turn the orthogonal basis set is calculated. By an additional point reflection of the vector set $\mathbf{B}$ across $\mathbf{p}$ by vector subtraction, resulting in the point set $\mathbf{q}^{(b)}$, the expansion into all trajectory-orthogonal directions of the state space is obtained with correspondingly calculated error control scaling factors $\beta_i^{(b)}$:

$$\mathbf{q}_i^{(a)} = \mathbf{p} + \beta_i^{(a)} \cdot \mathbf{b}_i \text{ for all } 1 \le i \le n_d \tag{2.52}$$

$$\mathbf{q}_i^{(b)} = \mathbf{p} - \beta_i^{(b)} \cdot \mathbf{b}_i \text{ for all } 1 \le i \le n_d \tag{2.53}$$

For every new starting point put into the WL, the inclusion in the defined discretization ranges of the extended state space has to be assured:

$$WL = WL \cup \{\mathbf{q}_i \in (\mathbf{q}^{(a)} \cup \mathbf{q}^{(b)}) | \mathbf{q}_i \in [0,1]^{n_d}\} \tag{2.54}$$

Figure 2.9 outlines the described process of determining the set of trajectory-orthogonal points $\mathbf{q}^{(a)}$ and $\mathbf{q}^{(b)}$ in a two-dimensional space.

The point $\mathbf{q}_1^{(b)}$ generated by the point reflection of the initial transition vector $\mathbf{v}$ across $\mathbf{p}$ is critical. Between $\mathbf{q}_1^{(b)}$ and $\mathbf{p}$ shall be a transition in the direction of the trajectory flow, which is not given by the reflection of $\mathbf{v}$. Hence, the transition vector obtained from a transient step starting in $\mathbf{q}_1^{(b)}$ must not necessarily map to $\mathbf{p}$. Therefore, a correction has to be calculated such that a transient step starting in $\mathbf{q}_1^{(b)}$ goes through $\mathbf{p}$. This can be iteratively resolved by determining the deviation vector

$$\Delta\mathbf{p}_i = \mathbf{p} - \mathbf{q}'_{1_i} \tag{2.55}$$

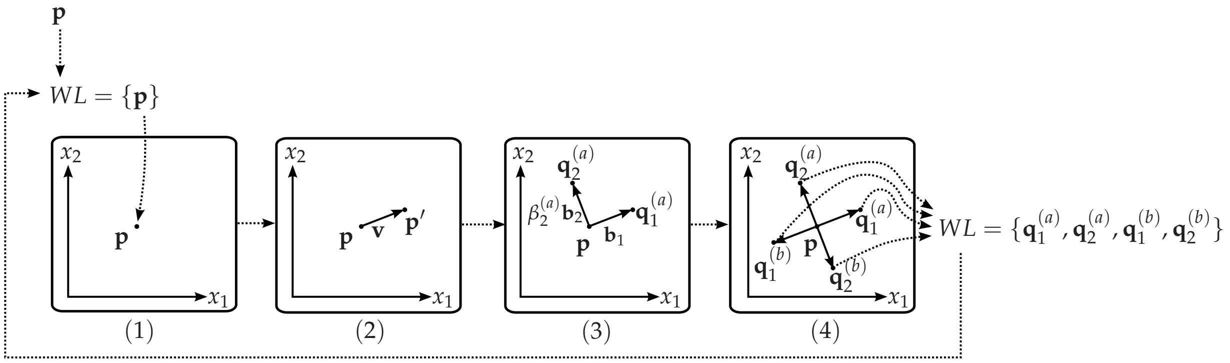


Figure 2.9: *Schematic visualization of the process of determining the trajectory-orthogonal point sets* $\mathbf{q}^{(a)}$ *and* $\mathbf{q}^{(b)}$ *in a two-dimensional space.*

between $\mathbf{p}$ and the point $\mathbf{q}'_{1_i}$ with $\mathbf{q}'_{1_i}$ determined by a transient step starting in $\mathbf{q}^{(b)}_{1_i}$. The corrected $\mathbf{q}^{(b)}_{1_{i+1}}$ is then given by:

$$\mathbf{q}^{(b)}_{1_{i+1}} = \mathbf{q}^{(b)}_{1_i} + \Delta\mathbf{p}_i \tag{2.56}$$

This correction algorithm that corresponds to the ideas in [DL09] is repeated either up to a predefined number i_{max} of times or until $\Delta\mathbf{p}_i$ is below a user-defined error bound. If the algorithm terminates without $\Delta\mathbf{p}$ being acceptable, the length of $\mathbf{v}$ being projected to generate the initial guess for $\mathbf{q}^{(b)}_1$ as well as of all transient step calculations are halved and the process is repeated until the error bound is reached.

Another issue is posed by the set of new starting points to put into the waiting list possibly containing points that are very close to points that have already been processed. Hence, a proximity criterion has to control the structure of the new starting points in order to avoid overlapping with existing points, giving priority to those points generated by transition vectors over those generated by the orthogonal vectors. If any of the new starting points $\mathbf{q}_i$ from $\mathbf{q}^{(a)}$ or $\mathbf{q}^{(b)}$ is closer to an already calculated existing point than 0.75 times the distance between $\mathbf{p}$ and $\mathbf{q}_i$, $\mathbf{q}_i$ is considered as redundant. In this case, $\mathbf{q}_i$ is replaced by the existing point or vice versa, keeping points originating from transient steps. The accepted points from $\mathbf{q}^{(a)}$ and $\mathbf{q}^{(b)}$ are appended to the waiting list WL. Every point in WL that has been processed is removed from WL and put into the list PL of accepted partitioning points.

The orthogonal sets to which an accepted point is connected to are stored, in order to later reconstruct the topology of geometric objects from these points. These connections are either represented by transition step vectors or by those from the orthogonal basis set. The coordinates of the accepted points in the untransformed state space can be calculated by inversing the transformation from Equation 2.45. Figure 2.10 illustrates a subset of the intended trajectory-directed partitioning for a three-dimensional state space with starting points, transition vectors and their orthogonal basis vectors forming the partitioning of the state space. The trajectory-directed partitioning algorithm is summarized in Algorithm 1.

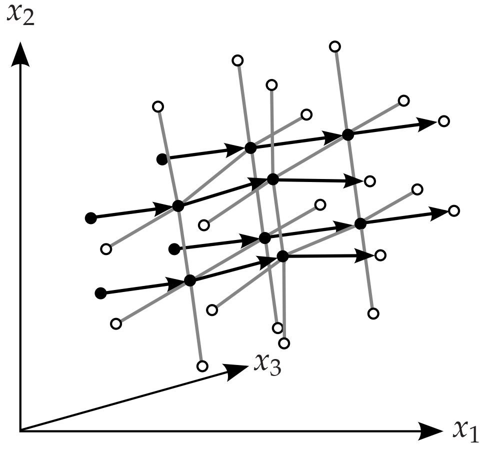


Figure 2.10: *Illustration of the orthogonal sets constructed around the transition vectors in a three-dimensional state space.*

2.4.4.2 Geometric Structure of the State Space Partitions

From the vertices and edges of the state space partitions calculated, geometric objects can be formed in order to define enclosures of distinct subsets $\mathcal{R}_j$ of $\mathcal{Z}$ where for every $\mathbf{p} \in \mathcal{Z}$ can be decided in which $\mathcal{R}_j$ it is enclosed.

The geometric object, spanned by the orthogonal set across the initial transition vector and constrained by those of neighboring transition vectors enclosing a region of the state space dynamic flow, exhibits a very general structure. It does not offer a regularity such as rectangular edges in order to be fitted into geometrical object classification. In the following, it will be referred to as hypercell in the n-dimensional case using the terminology of hyperdimensional partitioning. However, it can be described by the topology of its vertices and edges such that the undirected graph formed by the vertices and edges of the hypercell is isomorphic to the graph constructed for a hypercube in the same manner. In two dimensions, the object is a general quadrilateral. In three dimensions, the object is described of six of such quadrilateral facets, each of them spanned by four non-coplanar points connected by straight lines which can be imagined as a distorted cube. The hyperdimensional equivalent to the sides or facets of the hypercell in the three-dimensional case will be referred to as hyperfacets which, due to their non-planar form, are not (n-1)-hyperplanes.

The facets can be described by the mathematical concept of ruled surfaces which require for every point on the surface that there exists a straight line through this point

Algorithm 1: Trajectory-Directed Partitioning Algorithm.

```
Input: Waiting list $WL = \{\mathbf{p}_1\}$
Output: List of accepted partitioning points $PL$ with geometric topology
apply transformation $\mathbf{T}$ to all following steps
foreach $\mathbf{p}_j \in WL$ do
    $WL = WL \setminus \mathbf{p}_j$
    calculate transient step vector $\mathbf{v}_j = \mathbf{p}'_j - \mathbf{p}_j$
    generate orthogonal set $\mathbf{B}$ from $\mathbf{v}_j$
    foreach $\mathbf{b}_i \in \mathbf{B}$ do
        calculate error control scaling factors $\beta_i^{(a)}$ and $\beta_i^{(b)}$
        calculate points $\mathbf{q}_i^{(a)}$ and $\mathbf{q}_i^{(b)}$
        if $i$==$1$ then
            calculate corrected $\mathbf{q}_1^{(b)}$
        end
    end
    foreach $\mathbf{q}_i \in \{\mathbf{q}^{(a)} \cup \mathbf{q}^{(b)}\}$ do
        if $\mathbf{q}_i \in [0,1]^{n_d}$ then
            if $\neg\exists\, \mathbf{p}^{(ex)} \in (PL \cup WL) : \|\mathbf{p}^{(ex)} - \mathbf{q}_i\| < 0.75 \cdot \|\mathbf{p}_j - \mathbf{q}_i\|$ then
                $WL = WL \cup \mathbf{q}_i$
            end
        end
    end
    $PL = PL \cup \mathbf{p}_j$
end
reverse transformation $\mathbf{T}$
```

with every point on this line again being a point on the ruled surface. This matches the idea that the surface is a linearization between its spanning edges, which is adequately approximating the behavior of imaginary flow trajectories in the state space. Figure 2.11 illustrates a hypercell in three dimensions with faces defined by ruled surfaces.

While offering high approximation accuracy, in higher dimensions, the concept of non-planar hypersurfaces defining the faces of geometric objects makes it very difficult to calculate for a given point in state space to which hypercell it is assigned. Nevertheless, the non-overlapping complete non-uniform irregular partitioning of the state space into hypercells is the theoretical model for a high accuracy discretization with the introduced linearization error between vectors controlled by step-size control.

The goal is to reduce the complexity of the hypercell description in order to obtain a feasible description. The first observation is that, if the surface facets were hyperpla-

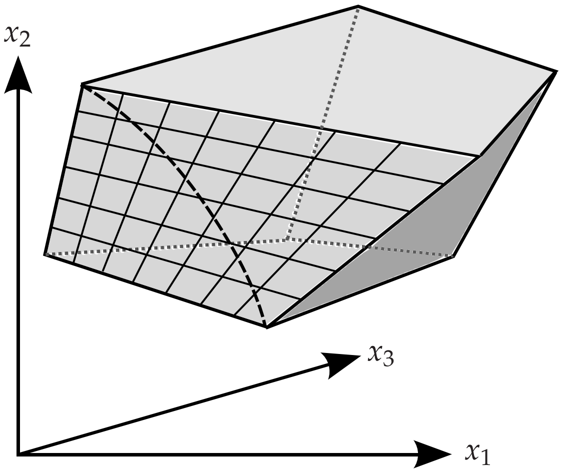


Figure 2.11: *Illustration of a hypercell object in a three-dimensional space.*

nar, the hypercell could be represented by n-polytopes due to the faces of n-polytopes by definition being (n-1)-polytopes for which the decision of point-enclosure is easy. However, transferring the hypercell to n-polytopes would destroy the accuracy of the enclosures as this is a major simplification.

The inclusion of a point $\mathbf{p}$ within a hypercell can be decided by checking the position of $\mathbf{p}$ relative to each of the $2n$ hyperfacets of the hypercell. If for every hyperfacet of the hypercell $\mathbf{p}$ lies, relative to the hypercell, on the inner side of the hyperfacet, $\mathbf{p}$ lies within the hypercell. Due to the edge-topology of the hypercell being isomorphic to the one of a hypercube, the 2^{n-1} vertices spanning each facet of the hypercell can be identified by graph traversal. A function $f(\mathbf{p})$ describing the position of $\mathbf{p}$ to the hyperfacet is needed such that:

$$f(\mathbf{p}) = \begin{cases} > 0 & \text{if } \mathbf{p} \text{ lies on the inner side of the hyperfacet} \\ 0 & \text{if } \mathbf{p} \text{ lies on the hyperfacet} \\ < 0 & \text{if } \mathbf{p} \text{ lies on the outer side of the hyperfacet} \end{cases} \tag{2.57}$$

Such a function is given by the distance of the normal vector on the hypersurface pointing to $\mathbf{p}$. Due to the hypersurface equation not being determinable analytically from the set of its vertices, a nonlinear hypersurface function shall be developed, describing the hypersurface by a weighted combination of (n-1)-hyperplanes, each spanned by the $n-1$ edges connected to each vertex of the hyperplane. As illustrated in Figure 2.12 for a facet of a three-dimensional cell, for each of the four vertices $\mathbf{a}_1$ to $\mathbf{a}_4$, a plane is spanned by its adjacent edges. For vertex $\mathbf{a}_1$ the corresponding plane is spanned by the edges $\mathbf{e}_{14}^{(a)} = \mathbf{a}_4 - \mathbf{a}_1$ and $\mathbf{e}_{12}^{(a)} = \mathbf{a}_2 - \mathbf{a}_1$.

If a point $\mathbf{p}$ is on this plane, it can be directly described by a linear combination of both edge vectors representing a parametric formulation of the plane. In order to

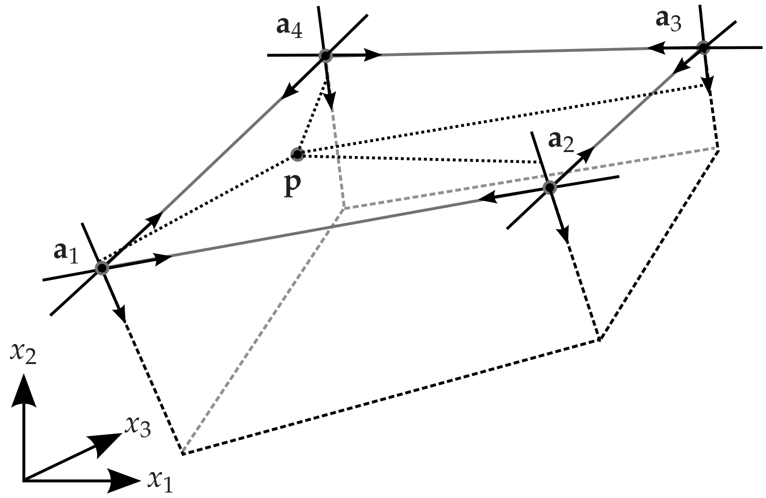


Figure 2.12: *Basis vector sets for each vertex of a hypercell facet for determination of the position of point* $\mathbf{p}$ *with respect to the facet.*

identify the position of an arbitrary point relative to the plane, a normal vector $\mathbf{n}_1$ on the plane is added to the linear combination. With the coefficient of $\mathbf{n}_1$ determining the distance to the plane corresponding to Equation 2.57, the position of $\mathbf{p}$ relative to the plane can be calculated.

For the n-dimensional case, a hypercell has $2n$ hyperfacets and each of the hyperfacets is described by $m = 2^{n-1}$ vertices $\mathbf{a}_1$ to $\mathbf{a}_m$. Each vertex $\mathbf{a}_i$ is connected to n edges $\mathbf{e}_{ij}$ of the hypercell of which $n-1$ span a hyperplane. The position of a point $\mathbf{p}$ relative to such a hyperplane can be described by the linear combination of the edges $\mathbf{e}_{ij}$ spanning the hyperplane and the normal vector $\mathbf{n}_i^{(a)}$ added to $\mathbf{a}_i$:

$$\mathbf{p} = \alpha_{i1}\mathbf{e}_{i1} + \alpha_{i2}\mathbf{e}_{i2} + \cdots + \alpha_{in-1}\mathbf{e}_{in-1} + \alpha_{in}\mathbf{n}_i^{(a)} + \mathbf{a}_i \tag{2.58}$$

For a subset of the hyperplanes, the normal vectors already have been computed as a member of the orthogonal set around the vertex connected to the hyperplane. However, due to the proximity criterion allowing vectors that are not in the direction of the trajectories to be non-orthogonal to the vectors spanning the hyperplane, a calculation of the normal vector can be mandatory for those hyperplanes. Unfortunately, the calculation of the normal vector $\mathbf{n}_i^{(a)}$ is nontrivial in the higher dimensional case. While in $\mathbb{R}^3$ the cross product of the two spanning vectors yields the normal vector on the plane, the general case in $\mathbb{R}^n$ is not directly accessible. The cross product $\mathbf{v}_1 \times \mathbf{v}_2 \times ... \times \mathbf{v}_{n-1}$ of any ordered (n-1)-tuple of vectors can be computed by forming a matrix whose subsequent rows are the vectors $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_{n-1}$. The k-th component of $\mathbf{v}_1 \times \mathbf{v}_2 \times ... \times \mathbf{v}_{n-1}$ is $(-1)^k$ times the determinant of the submatrix obtained by deleting the k-th column [Mas83]. As an alternative to the generalized cross product, the normal vector can again be computed by the Gram-Schmidt orthogonalization where the $n-1$ vectors spanning the edges of the hyperplane and the pseudo-orthogonal vector $\mathbf{v}_n$ from the initial point determination are used as input, resulting in an orthogonalization of $\mathbf{v}_n$ to the vectors spanning the (n-1)-hyperplane.

The parametric equation for the hyperplane is given by setting $\alpha_{in} = 0$ in Equation 2.58. In order to determine the coefficients α_{ij} for a given $\mathbf{p}$, an equation system consisting of m equations for every vertex of the hyperplane can be set up:

$$\begin{array}{ccccccccccccccc}
\alpha_{11}\mathbf{e}_{11} & + & \alpha_{12}\mathbf{e}_{12} & + & \cdots & + & \alpha_{1n-1}\mathbf{e}_{1n-1} & + & \alpha_{1n}\mathbf{n}_1^{(a)} & + & \mathbf{a}_1 & - & \mathbf{p} & = & 0 \\
\alpha_{21}\mathbf{e}_{21} & + & \alpha_{22}\mathbf{e}_{22} & + & \cdots & + & \alpha_{2n-1}\mathbf{e}_{2n-1} & + & \alpha_{2n}\mathbf{n}_2^{(a)} & + & \mathbf{a}_2 & - & \mathbf{p} & = & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
\alpha_{m1}\mathbf{e}_{m1} & + & \alpha_{m2}\mathbf{e}_{m2} & + & \cdots & + & \alpha_{mn-1}\mathbf{e}_{mn-1} & + & \alpha_{mn}\mathbf{n}_m^{(a)} & + & \mathbf{a}_m & - & \mathbf{p} & = & 0
\end{array} \tag{2.59}$$

Two out of four planes spanned by the edge vectors of a facet in a three-dimensional space are illustrated in Figure 2.13.

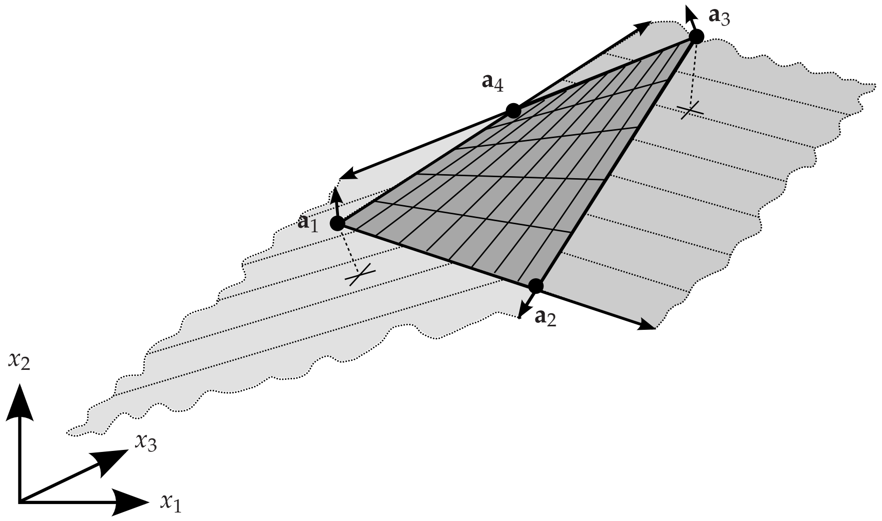


Figure 2.13: *Curved surface in a three-dimensional space representing a facet of a cell. Two of the four planes spanned by the edge vectors of the vertices* $\mathbf{a}_1$ *and* $\mathbf{a}_3$ *are illustrated. The weighting function uses one plane for each vertex of the facet in order to obtain a nonlinear surface function describing the facet.*

Finally, after calculating the coefficients α_{in} of the m normal vectors $\mathbf{n}_i^{(a)}$, the curved surface function of the hypersurface can be approximated by a combination of the planes spanned by the edge vectors of each vertex using a weighting of the coefficients

α_{in} corresponding to the distance of points $\mathbf{p}$ to each $\mathbf{a}_i$. This yields the function $f(\mathbf{p})$ determining the position of $\mathbf{p}$ relative to the hypersurface:

$$f(\mathbf{p}) = \frac{\alpha_{1n} c^{-\|\mathbf{p}-\mathbf{a}_1\|} + \alpha_{2n} c^{-\|\mathbf{p}-\mathbf{a}_2\|} + \cdots + \alpha_{mn} c^{-\|\mathbf{p}-\mathbf{a}_m\|}}{c^{-\|\mathbf{p}-\mathbf{a}_1\|} + c^{-\|\mathbf{p}-\mathbf{a}_2\|} + \cdots + c^{-\|\mathbf{p}-\mathbf{a}_m\|}} \tag{2.60}$$

The basis c shall be chosen such that every other $c^{-\|\mathbf{p}-\mathbf{a}_i\|}$ evaluates to 0 if there is an a_j with $\|\mathbf{p} - \mathbf{a}_j\|$ below a threshold value. This ensures that the weight of other hyperplanes goes towards zero when a point is near a vertex $\mathbf{a}_j$.

The hypersurface function describing the points on the facet is obtained by setting $f(\mathbf{p}) = 0$. The denominator scales the sum of the weights to 1. By using the exponential function such that distance 0 results in a weight of 1 and by the normalization of the distances, high approximation accuracy is guaranteed.

2.4.4.3 Transition Relation of the Hypercells

The paraxial hyperbox partitioning approach needed to calculate the trajectories of sample points in a hyperbox in order to determine which hyperboxes are reached by these trajectories for determining the nondeterministic over-approximated successor relation. In contrast, the trajectory-directed approach guarantees by design that the successor of a hypercell is the adjacent hypercell in the direction of the transition vectors spanning the hypercell. The flow of the state space dynamics is considered to be homogeneous within a partition enclosed by the transition vectors of the edges of the partition. Due to the edges in flow direction being parallel to the flow, every point within a hypercell maps to the same adjacent subsequent hypercell. This successor relation can be directly obtained from the iterative partitioning algorithm that explores the state space using the trajectory-orthogonal expansion.

The only exception arises when the partitioning algorithm is splitting the subsequent hypercell by insertion of additional points. This is due to an expansion of the vector field flow or merging the flow of preceding hypercells into a single successor caused by a contraction of the flow. In such cases, the proximity criterion maps one successor to more than one predecessor or vice versa. Hence, more than one hyperfacet can be adjacent to a larger facet of a hypercell if the flow contracts. If the flow expands, one hyperfacet of a hypercell can be adjacent to more than one hyperfacet of subsequent hypercells. In such a case, the successor relation has to be adapted to map to the corresponding successors as illustrated in Figure 2.14. In order to calculate the surface of the larger hyperfacet, the hypersurface formula, as introduced in Equation 2.59, can then be computed by taking into account all vertices of the adjacent smaller hyperfacets.

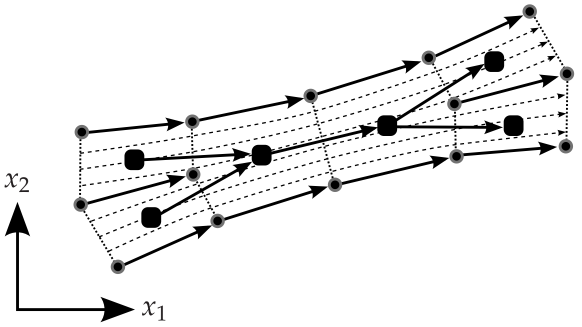


Figure 2.14: *Schematic illustration of contracting and expanding vector field flow, with the corresponding partitioning and the successor relation for the hypercells.*

2.4.4.4 Mapping the Trajectory-Directed Partitioning to a DATS

With the vertices and edges of the state space partitioning determined by using the trajectory-directed approach, a mapping onto a DATS has to be generated in order to apply verification algorithms.

While the state space partitions, represented by hypercells, and the inherent successor relation have been introduced in the previous subsections, it has to be defined how a hypercell is represented by a state of the DATS. Moreover, defining the successor relation of the DATS connecting the states, as well as the rest of the parameters of the DATS, is required to complete the discrete model.

Determining point inclusion in the hypercells is important for the completeness of the theoretical model of the partitioning. It could for example be applied to develop verification algorithms not operating on a discrete graph structure but directly on symbolically determined point-to-point mappings within the finite set of hypercells. In such a concept, the homogeneous flow in the hypercells, determined by the transition vectors spanning the hypercells, could define the transition vector length and direction of a point by taking into account the weighted position of the point to the hypercell transition vectors.

However, for application of graph-based verification algorithms, each state σ_i of the DATS is corresponding to one hypercell $\mathcal{R}_i$ with $1 \leq i \leq k$. Hence, the cardinality of Σ is k:

$$\Sigma = \{\sigma_1, ..., \sigma_k\} \tag{2.61}$$

The DATS is then constructed by the following mappings. The parameter vectors of the states is given by the labeling

$$L_V(\sigma_i) = \mathbf{p}_i^{(c)} \tag{2.62}$$

with $\mathbf{p}_i^{(c)}$ being the center point of $\mathcal{R}_i$, representing the extended state space variables in this point. The center is calculated from the $m = 2^{n_d}$ vertices $\mathbf{a_j}$ that constrain a hypercell:

$$\mathbf{p}_i^{(c)} = \frac{1}{m} \sum_{1 \leq j \leq m} \mathbf{a}_j \tag{2.63}$$

A transition relation exists between those states where the state space trajectories starting in a hypercell $\mathcal{R}_i$ reach the adjacent hypercell $\mathcal{R}_j$. The adjacency is determined by the intersection of the sets of vertices $\mathbf{a}^{(i)}$ spanning $\mathcal{R}_i$ and $\mathbf{a}^{(j)}$ spanning $\mathcal{R}_j$ not being empty:

$$R = \{\bigcup(\sigma_i, \sigma_j) | \, \forall \, \mathbf{p} \in \mathcal{R}_i \, \exists \, \Delta t : \mathbf{p} + \mathbf{v} \cdot \Delta t \in \mathcal{R}_j\} \text{ with } \mathbf{a}^{(i)} \cap \mathbf{a}^{(j)} \neq \varnothing \tag{2.64}$$

The transition times between σ_i and σ_j are determined by the trajectory time Δt computed by a transient step from $\mathbf{p}_i^{(c)}$ to $\mathbf{p}_j^{(c)}$:

$$T(R(\sigma_i, \sigma_j)) = \Delta t(\mathbf{p}_i^{(c)}, \mathbf{p}_j^{(c)}) \tag{2.65}$$

Each transition with a transition time greater zero has an atomic transition proposition of 0, marking it as dynamic transition created by a state space trajectory:

$$L_T(R(\sigma_i, \sigma_j)) = 0 \Leftrightarrow T(R(\sigma_i, \sigma_j)) > 0 \tag{2.66}$$

A trajectory starting in a DC-operating-point ends within this point, hence there is a transition of a state representing a DC-operating-point to itself:

$$(\sigma_i, \sigma_i) \in R \Leftrightarrow \{\forall \, \mathbf{p} \in \mathcal{R}_i \, : \, \mathbf{p} + \mathbf{v} \cdot \Delta t \in \mathcal{R}_i\} \text{ for all } \Delta t \geq 0 \tag{2.67}$$

By definition, the transition time of such transitions shall be zero, as the circuit stays in this loop state until an external excitation makes the circuit leave this state:

$$T(R(\sigma_i, \sigma_i)) = 0 \text{ and } L_T(R(\sigma_i, \sigma_i)) = 0 \tag{2.68}$$

2.4.4.5 Duality of the Trajectory-Directed Partitioning

With the mapping of the trajectory-directed state space partitioning onto a DATS, application of verification algorithms to the DATS modeling an analog circuit would be possible. However, one observation can motivate an approach to decrease the modeling effort of determining the hypercells significantly without decreasing the discretization accuracy for the DATS. This observation is the duality of the trajectory-directed edges connecting the vertices of hypercells and the transition vectors between the centers of the hypercells. Figure 2.15 illustrates this duality where the transient steps between the vertices determining the edges of the hypercells exhibit the same transition behavior as the transitions connecting the centers of the hypercells, themselves approximating the trajectories between adjacent centers of the hypercells. Moreover, while the transition relation between two adjacent centers is an approximation calculated by the connection of the centers of both hypercells, the transition steps creating the edges of the hypercells are by design computed with the accuracy of transient analysis.

Therefore, for an implementation of the trajectory-directed partitioning, the vertices and edges computed by the partitioning algorithm shall be directly defining the states of the DATS and the corresponding transition relations.

When transferring the trajectory-directed partitioning to a DATS, only the parameter vector for each state has to be changed compared to the mapping defined in Section 2.4.4.4:

$$L_V(\sigma_i) = \mathbf{p}_i \tag{2.69}$$

with $\mathbf{p}_i$ representing the original sampled points in state space of the trajectory-directed discretization algorithm. The transition relations are directly determined by the transient step vectors $\mathbf{v}_{ij}$ computed during the initial state space sampling, resulting in a deterministic transition relation where each state has exactly one successor state:

$$R = \{\bigcup(\sigma_i, \sigma_j) | \exists\, \mathbf{v}_{ij} : \mathbf{p}_i + \mathbf{v}_{ij} = \mathbf{p}_j\} \tag{2.70}$$

All other mappings defined in Section 2.4.4.4 apply correspondingly by replacing the centers of the hypercells $\mathbf{p}_i^{(c)}$ with the initially sampled $\mathbf{p}_i$.

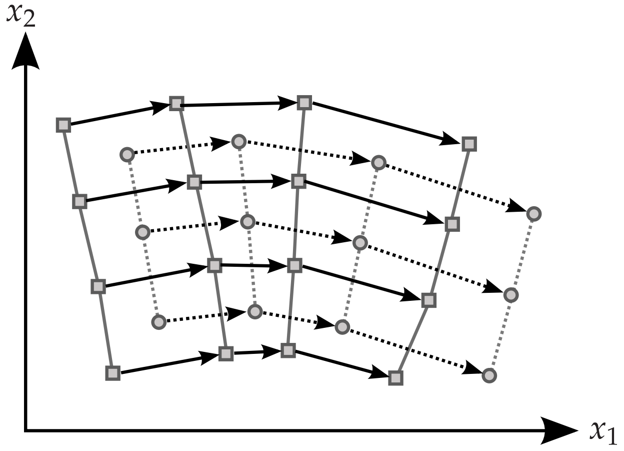


Figure 2.15: *Illustration of a two-dimensional vector field with calculated transition endpoints as boxes and the quadrilateral enclosed regions of the state space. The circles represent the centers of the enclosed state space regions, connected corresponding to the initially calculated transition relation. The regions enclosed by the quadrilaterals of four region centers are dual to the regions bounded by the transition vectors.*

2.4.4.6 Handling Input Variables

The DATS obtained from the trajectory-directed discretization of the state space presented in Section 2.4.4.4 with the simplification from Section 2.4.4.5 models the dynamic behavior of an analog circuit. For systems without inputs or with constant input values, this discrete model completely represents the system in order to be verified using verification algorithms on the graph structure. However, for systems with input variables determined by external input stimuli, this model has to be extended.

Therefore, an input variable model for the trajectory-directed discrete modeling has to be developed. This model shall correspond to the idea of the transient simulation algorithm for analog circuits, where in every time step the solution is computed considering constant input values and changes of the input variables are only considered between the time steps. Hence, those dimensions of the extended state space represented by input variables are explored by the trajectory-orthogonal sampling just as the other state space variables determined by energy-storing elements of the circuit.

In order to allow arbitrary input changes for transitions between states in the discrete model, for each state and for each input dimension, an input transition for a state to its successor and predecessor parallel to the axis of the corresponding input dimension has to be created in the DATS.

Fortunately, using the dual representation where the sampled points in the state space correspond to the centers of the states $L_V(\sigma_i) = \mathbf{p}_i$, the trajectory-directed discretization algorithm already generates the information about the two neighboring states for σ_i parallel to the corresponding input axis dimension s due to the orthogonalization algorithm. The input axis is described by the vector $\mathbf{i}_s$, being the s-th column vector of the identity matrix $\mathbf{I}_{n_d}$. Calculated from the transition vector $\mathbf{v}$ starting in $\mathbf{p}_i$, the orthogonal vector $\mathbf{b}_s$, generating the points $\mathbf{q}_s^{(a)}$ and $\mathbf{q}_s^{(b)}$, is parallel to the input axis. This is due to the trajectories being sampled with piecewise constant inputs, and the inputs are only changed between sample steps. Hence, the dynamic transition vectors have zero magnitude in the direction component of the input dimensions and therefore, the normalized $\mathbf{b}_s$ equals to $\mathbf{i}_s$.

For a state σ_i, every identified neighboring state σ_j for every input dimension is finally connected to σ_i by an undirected transition in the DATS, meaning that these input edges can be traveled in both directions corresponding to any external input variable change:

$$R = R \cup (\sigma_i, \sigma_j) \cup (\sigma_j, \sigma_i) \tag{2.71}$$

By definition, the transition time of such transitions shall be zero:

$$T(R(\sigma_i, \sigma_j)) = 0 \text{ and } T(R(\sigma_j, \sigma_i)) = 0 \tag{2.72}$$

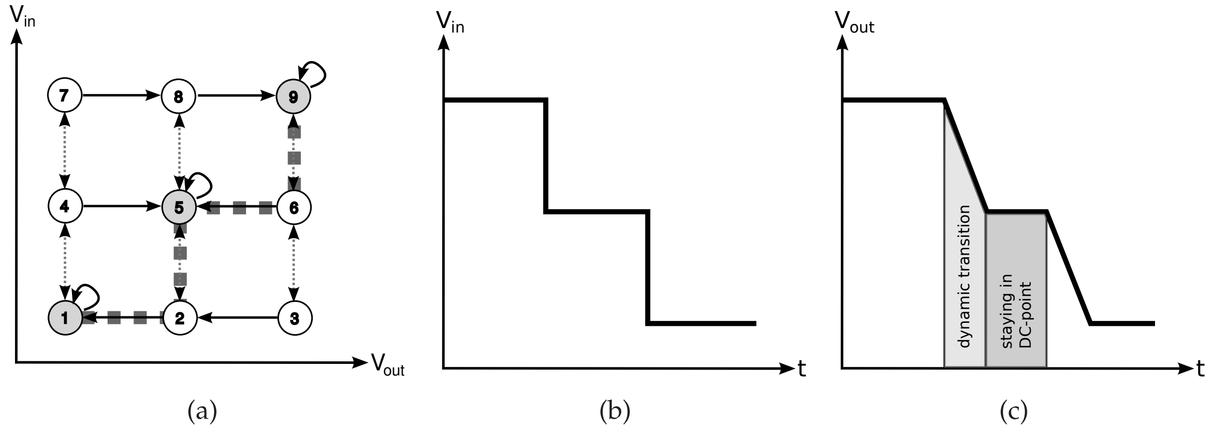


Figure 2.16: *Schematic illustration of a DATS with a possible input change-induced transition path (a). The corresponding input signal (b) and output signal (c) is assumed for this path.*

Finally, these transitions have to be identified as input edges in the DATS for offering the possibility of masking these states for verification algorithms such as oscillation detection:

$$L_T(R(\sigma_i, \sigma_j)) = 1 \text{ and } L_T(R(\sigma_j, \sigma_i)) = 1 \tag{2.73}$$

Figure 2.16(a) illustrates an imaginary DATS with nine states with a highlighted input change-induced transition path. The input signal that can be traced in the DATS leading to this path is illustrated in Figure 2.16(b) with the corresponding output signal behavior illustrated in Figure 2.16(c). Consider the system to be in the DC-operating-point represented by state 9. The circuit can stay there infinitely. However, the first step of the input voltage brings the circuit to leave state 9 over an input edge into the state corresponding to the new input voltage represented by state 6. This is a dynamic state which means that immediately a dynamic timed transition to state 5 occurs which again represents a DC-operating-point. The process is repeated for the next input step, finally bringing the modeled circuit to stay in state 1. With the modeled input edges in the DATS, all possible input connections between states can be considered for verification.

A limitation of the edge steepness of input signal changes should be handled by the verification algorithms on the DATS, assigning a nonzero transition time to these input transitions when a limited input bandwidth has to be considered. Therewith, arbitrary piecewise linear input stimuli can be represented on the transition paths.

2.4.4.7 Runtime Complexity

Considering the discretization of an analog circuit to a DATS with respect to the direct mapping of the states to the sampled points in the state space as described in Section 2.4.4.5, the asymptotic worst-case runtime complexity of the trajectory-directed discretization algorithm is correlated to the number of points n_p sampled in the state space of the modeled circuit. This number of points increases exponentially with the number of dimensions n_d of the extended state space (variables of the energy storing elements and input dimensions). The base κ of the exponential function represents the average number of sampling points needed for covering an one-dimensional range, which is determined by the step length control of the transient steps between the sampled points in the state space:

$$n_p = \kappa^{n_d} \tag{2.74}$$

For every sampled point, the trajectory-directed discretization algorithm computes the transient simulation step contributing t_{tr}, the Gram-Schmidt orthogonalization contributing t_{gs} and the distance information contributing t_{pr} for the proximity criterion. The remaining parts of the algorithm with subordinate contribution to the complexity of the algorithm are summed up in t_{re}. An exact asymptotic complexity for transient analysis depends on the applied set of algorithms. However, the transient analysis algorithm contains parts with cubic asymptotic worst-case complexity with respect to the matrix of the circuit equations and the number of variables being related to n_d. The Gram-Schmidt algorithm has a complexity of $\mathcal{O}(n_d^3)$ [GVL96]. The proximity neighbor search conducted by a Kd-tree consumes $\mathcal{O}(n_p \log n_p)$ to be created and $\mathcal{O}(\log n_p)$ for the query [Ben90]. Hence, the runtime t_d of a discretization run can be estimated by:

$$t_d = \kappa^{n_d} \cdot \underbrace{(t_{tr} + t_{gs} + t_{pr} + t_{re})}_{t_p} \tag{2.75}$$

The overall computation time of a single sample point t_p is dominated by the large factor of transient analysis t_{tr}, with t_{pr} only becoming dominant for high state numbers. Anyhow, with respect to n_p, the asymptotic runtime complexity of the discretization algorithm is dominated by the proximity computation as it is the only component with direct dependency on n_p:

$$\mathcal{C}_{n_p} = \mathcal{O}(n_p \log n_p) \tag{2.76}$$

However, changing the perspective of complexity considerations to be relative to the number of state space dimensions, the asymptotic worst-case runtime complexity of the trajectory-directed discretization algorithm is dominated by the exponential growth of sample points with respect to the number of state space dimensions:

$$\mathcal{C}_{n_d} = \mathcal{O}(\kappa^{n_d}) \tag{2.77}$$

While this exponential runtime complexity in the number of extended state space dimensions is common to all discrete modeling approaches for analog circuits, relevant analog circuit blocks usually do not exceed a system order of eight, which can be handled well by this approach. Moreover, by application of an eigenvalue-based model order reduction of the DAE system [HKH04], an extension to circuits with parasitic capacitances and full BSIM3 transistor models would be possible. The reduction can be achieved by reducing the state space to the dominant state variables of a system and separating the parasitic ones which are mathematically proven not to affect the system behavior above a defined threshold.

2.4.4.8 Modeling Error Analysis

The goal of the previously introduced discrete modeling of analog systems is to obtain a circuit model with minimal discretization error, corresponding to the criteria presented in Section 2.4.2. In order to obtain an overall impression on the different modeling errors of analog systems, three major classes of errors have to be distinguished when comparing a physical circuit implementation with any type of mathematical model.

Firstly, the error in the following referred to as "physical modeling error" will describe all differences between the physical implementation and the mathematical DAE circuit model. Secondly, the results of transient analysis are affected by numerical computation errors which are referred to as "simulation error". This simulation error is common for all contemporary circuit analysis tools. Thirdly, the error introduced by the discrete modeling process will in the following be referred to as "discretization error". This discretization error distinguishes the different discretization approaches.

Physical Modeling Error The DAE model generated for transient circuit analysis by MNA, as described in Section 2.3.2, introduces an error due to the simplified BCEs of the circuit elements using device models not representing the complete physical effects down to the quantum level. Especially for transistor models in sub-micron technologies, high effort of modeling is spent on capturing their behavior.

However, as discussed in Section 2.3.1, even the most exact models available do not offer a complete model of the physical behavior. Often, modeling accuracy is traded in for faster runtimes of the algorithms operating on the DAE model, as the number of equations has a negative influence on the runtime of the analysis algorithms.

Simulation Error Not only the model itself contains an error but also the transient analysis algorithm described in Section 2.3.3, which is based on numerical integration, contributes an error when computing the system's behavior on the DAE model. Although the LTE of each step controls the time step length, in implementations of the SPICE algorithms there are user defined error thresholds such as RELTOL and ABSTOL which affect the accuracy of the transient analysis. However, by increasing the accuracy, the number of computed time steps and the number of iterations of the numerical integration in each step increases just as well.

Discretization Error The discretization error is introduced by the representation of the continuous dynamics of an analog circuit by a discrete transition system in which the behavior of the circuit has to be captured.

As a general benchmark for a discrete model of a circuit, the comparison of state space trajectories can be considered. On the one hand, they are computed by a transient analysis with the algorithm described in Section 2.3.3, on the other hand the trajectory is determined by a discrete set of states in the modeled DATS. Such practical evaluations will be made in Chapter 6 in order to compare the results of the implementation of the new trajectory-directed discretization to the state-of-the-art hyperbox discretization algorithm and to transient analysis.

Based on the five criteria to evaluate a discretization approach defined in Section 2.4.2, the theoretical modeling quality of the trajectory-directed discretization approach is discussed in the following in comparison to the hyperbox discretization approach. Table 2.2 summarizes the characteristics of both approaches. Besides these theoretic considerations, a comparison of experimental results for the successor relation error and the determinism of the successor relation of the trajectory-directed discretization compared to the hyperbox discretization will be discussed in Section 6.3.4.

Direction Error The direction error $\epsilon_\theta^{(\mathcal{R}_j)}$ within the partitions $\mathcal{R}_j$ of the trajectory-directed approach is controlled by a user-defined maximum r_θ. Hence, the algorithm controls the size and structure of the partitions to be below this bound. However, around singularities such as the attractors in the state space introduced by DC-operating-points, a threshold value controls the minimum partition size not be decreased infinitely. This comes at the cost of not meeting the error threshold in this particular case.

In order to obtain a DATS model of a size that can be checked well by verification algorithms, the integration time of the transient simulations used for computing the transition vectors determining the partition size can be set to a user-defined minimum and maximum. The same applies to the hyperbox discretization approach. Due to the higher degree of freedom for deciding the partitioning of the state space, the trajectory-directed approach can be expected to create less partitions for the same ϵ_θ compared to the hyperbox discretization.

Length Error Just like the direction error, the length error $\epsilon_\Delta^{(\mathcal{R}_j)}$ within the partitions $\mathcal{R}_j$ is controlled by a user-defined maximum r_Δ. The same considerations as for the direction error apply correspondingly.

Number of Partitions The number of partitions k of the trajectory-directed approach is indirectly determined by the user-defined bounds r_θ and r_Δ. This is due to the size of the partitions within a selected part of the state space to be discretized being correlated to the allowed direction and length errors or the minimum and maximum allowed partition size.

Determinism of the Successor Relation The trajectory-directed discretization creates the successor relation parallel to the flow of the state space dynamics. Therewith, the out-degree of the states in the DATS is mostly 1. The only possible exception was discussed in Section 2.4.4.3. Moreover, the dual representation of the partitioning assures the out-degree to be exactly 1 when no borders of the state space are reached that cause the determination of successors to be stopped.

In contrast, the hyperbox discretization can over-estimate the angle of the successor relation up to 90 degrees. Therewith, depending on the number n_d of dimensions of the state space, the out-degree can be up to $2^{n_d} - 1$. This is due to the paraxial slicing and therewith, in a worst-case scenario as previously illustrated in Figure 2.7(d), in all dimensions all adjacent boxes within an angle of 90 degrees can be selected as successors. Those nondeterministic paths are weakening the expressiveness of the model as an over-approximated set of possible trajectories is reported.

Successor Relation Error With the arguments given in the previous paragraph, the successor relation error ϵ_{suc} of the trajectory-directed discretization is almost 0 degrees as most of the successors shall be directly determined by transient steps when using the initially sampled points as center vectors of the states of the DATS. Even when using the centers of the hypercells for the center vectors, the possible successor relation error between two states is bounded by the user defined r_θ. This is due to the enclosing vectors of the hypercell, that have been calculated by transient analysis, are controlled not to have an angle difference above r_θ. Corresponding to the previous considerations, the only exception applies to states representing DC-operating-points.

For the hyperbox discretization approach, ϵ_{suc} can be up to 90 degrees. This can again be concluded from the example shown in Figure 2.7(d).

Table 2.2: *Comparison of the trajectory-directed discretization and the hyperbox discretization approaches.*

Criterion	**Trajectory-directed**	**Hyperbox**
Direction error	$< r_\theta$	$< r_\theta$
Length error	$< r_\Delta$	$< r_\Delta$
Partitions	determined by r_θ, r_Δ	determined by r_θ, r_Δ
Successor determinism	≈ 1	$\leq 2^{n_d} - 1$
Successor relation error	$\approx 0^\circ$	$< 90^\circ$

3

Property Specification for Verification

In the previous chapter, the system representation for verification with emphasis on developing a DATS model of analog systems has been presented. Based on this introduced discrete system representation, property specification approaches will be introduced in the following. Starting with a definition of the three elementary concepts of property, performance and specification, an advanced approach for analog property specification is systematically developed by discussing existing approaches.

3.1 Basic Definitions

A set of properties can be defined for a system that are relevant to evaluate the system behavior.

Definition 3.1.1 (Property)
A system's property can be any function that can be calculated on the system's variables. All properties of a system span the property space P*.*

Within the property space, the system exhibits a characteristic behavior that constrains the property space to nominal performances that the system can exhibit by system analysis.

Definition 3.1.2 (Performance)
A system performance $\mathbf{f}(S, P)$ *is the result of an evaluation of system properties and hence represents a point in the property space.*

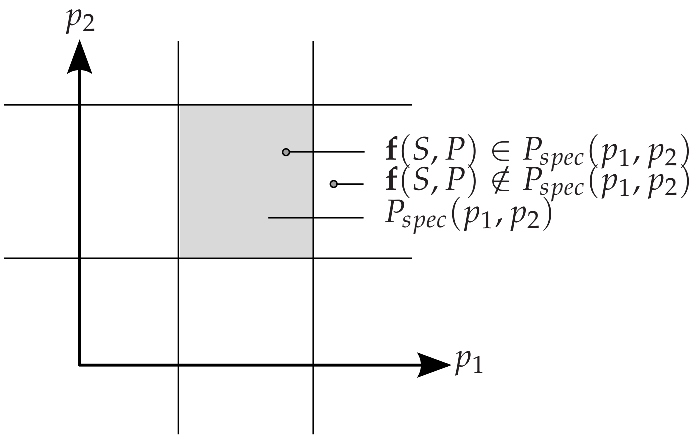

p_2
$\mathbf{f}(S,P) \in P_{spec}(p_1,p_2)$
$\mathbf{f}(S,P) \notin P_{spec}(p_1,p_2)$
$P_{spec}(p_1,p_2)$
p_1

Figure 3.1: *Property space for properties p_1, p_2 with specification $P_{spec}(p_1, p_2)$ and performances satisfying and violating the specification.*

Definition 3.1.3 (Specification)
A specification $P_{spec} \subset P$ defines a subspace of the property space by constraining it to required system performances. Figure 3.1 illustrates a property space with a system performance satisfying and a performance violating the specification.

3.2 Operational and Declarative Specification

In order to verify a system, it is mandatory to define under which performance constraints it will be considered as fully functional. The property specification used to evaluate these performance constraints has several aspects. While definition 3.1.3 describes the abstract characteristics of a specification, practical requirements to specifications are complex.

Initially, the functional requirements are often defined informally in a natural language specification in the system design process. Based on these requirements, a technical specification is created which then has to be transferred into a property specification that can be evaluated during the design process. For automated approaches, this specification has to be formalized to be machine-readable and therewith can be evaluated by verification algorithms. Moreover, the property values not only have to be specified. Additionally, a formal and well-defined specification of how these properties are evaluated within the verification environment has to be made. This can be achieved by defining consistent verification semantics for the available measurements. With the level of formality of the verification approach, the requirements in the formality of the specification are increasing as well.

Formal specifications can either be operational or declarative [Lam00]. An operational specification describes a model of the system to be designed as a collection of processes that the system shall incorporate. In the area of digital system specification, an operational specification would be in form of a finite state machine describing the desired operation of an implementation of this machine. While operational specifications are common in the area of software design using a component model, operational models for specification of digital systems quickly grow in complexity and hence are not widely used.

Operational specification in analog hardware design is applied when a top down system design flow is specified by a set of simplified behavioral models to specify the high-level design of the system. By checking the functional equivalence between such operational behavioral specifications and a lower abstraction level implementation, much of the design flow verification is accomplished.

Declarative specification uses a logic-based reasoning on elementary properties of a system from which more complex relations are specified in a recursive approach. For digital systems, declarative property specification can be connected with Boolean logic that is enhanced by a temporal reasoning layer. In contrast, analog declarative specification needs a translation from the complex analog system properties to a consistent specification language with a logic foundation for a well-defined semantic definition. There is a wide gap between declarative temporal logic-based specification that will be discussed in the following section and the designer's intent of informal specification of analog circuit properties. Hence, the characteristics of existing logic-based formal specification approaches have to be analyzed in order to develop a formal specification and verification methodology for analog circuit properties in Section 5.

3.3 Property Specification for Discontinuous Systems

The foundations of formal system property specification have been developed in conjunction with the application of temporal logic reasoning to program verification. Linear Temporal Logic (LTL) [Pnu77] and Computation Tree Logic (CTL) [CE82] specifications are evaluated on transition systems represented as Kripke structures which are directed labeled graphs. This proof-based verification of system properties formulated in temporal logics on system models is called model checking.

Temporal logic specification applied to models of digital circuits enabled CTL-based model checking approaches. The breakthrough was marked by the introduction of symbolic approaches of modeling transition systems and transition relations using Binary Decision Diagrams (BDDs), solving the problem of state space explosion due to its explicit representation, handling up to 10^{20} states [BCM^{+}90] and introducing the Symbolic Model Verifier (SMV) [McM92]. In order to overcome the problem of very abstract specification formulation, approaches to classification of common property patterns in CTL [DAC98] and application of natural language specification by translating English natural language specifications to SMV code [Hol99] attempted to align formal property specification and verification engineers' thoughts.

With formal verification gaining more and more importance, a consortium of IC-design and EDA-companies developed the proposal of the Property Specification Language (PSL), finally becoming IEEE standard 1850 in 2005 [FMW05].

3.3.1 Linear Temporal Logic (LTL)

Linear temporal logic (LTL) [Pnu77] has a linear, non-branching time model with temporal modal operators operating on propositional variables as atomic propositions connected by the logical connectives *negation*, *and*, *or* and *implication*. The syntax of well-formed LTL formulas is defined by the following context-free grammar:

$$\begin{aligned} \phi = \quad a \; & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\ & \mid \diamond\, \phi \mid \phi_1 \,\mathrm{U}\, \phi_2 \end{aligned} \tag{3.1}$$

An explanation of the placeholders for the language symbols of LTL is given in Table 3.1 and Figure 3.2 visualizes the semantics of the temporal operators of LTL. The semantics of LTL are defined as follows with respect to a path $\pi = \sigma_0, \sigma_1, \sigma_2, ..., \sigma_n$ in the Kripke structure M and properties ϕ and ψ of atomic propositions a:

$$M, \sigma_0 \vDash a \quad \Leftrightarrow \quad a \in L(\sigma_0) \tag{3.2}$$

$$M, \sigma_0 \vDash \neg\phi \quad \Leftrightarrow \quad \text{logical negation of } (\sigma_0 \vDash \phi) \tag{3.3}$$

$$M, \sigma_0 \vDash \phi \wedge \psi \quad \Leftrightarrow \quad \sigma_0 \vDash \phi \text{ and } \sigma_0 \vDash \psi \tag{3.4}$$

$$M, \sigma_0 \vDash \phi \vee \psi \quad \Leftrightarrow \quad \sigma_0 \vDash \phi \text{ or } \sigma_0 \vDash \psi \tag{3.5}$$

$$M, \sigma_0 \vDash \mathrm{X}\, \phi \quad \Leftrightarrow \quad \sigma_1 \vDash \phi \tag{3.6}$$

$$M, \sigma_0 \vDash \mathrm{G}\, \phi \quad \Leftrightarrow \quad \forall\, i \geq 0 : \sigma_i \vDash \phi \tag{3.7}$$

$$M, \sigma_0 \vDash \mathrm{F}\, \phi \quad \Leftrightarrow \quad \exists\, i \geq 0 : \sigma_i \vDash \phi \tag{3.8}$$

$$M, \sigma_0 \vDash \phi \,\mathrm{U}\, \psi \quad \Leftrightarrow \quad \exists\, i \geq 0 : \sigma_i \vDash \psi \text{ and } \forall\, 0 \leq j < i : \sigma_j \vDash \phi \tag{3.9}$$

The temporal operators of LTL allow to specify important system characteristics such as safety and liveness. Safety assumes that something bad never happens, which maps to the following LTL formula when considering the states with the atomic proposition ϕ as bad: G $\neg\phi$. This specification states that generally, i.e. for all future states of the design under verification (DUV), states labeled with ϕ will never be reached. Liveness assumes that something good, represented by states where the atomic proposition ψ is true, keeps happening, which maps to the LTL formula GFψ.

Table 3.1: *Explanation of LTL syntax.*

a	atomic proposition	
$\diamond$	temporal operator	F = eventually
		G = generally
		X = next
U		U = until

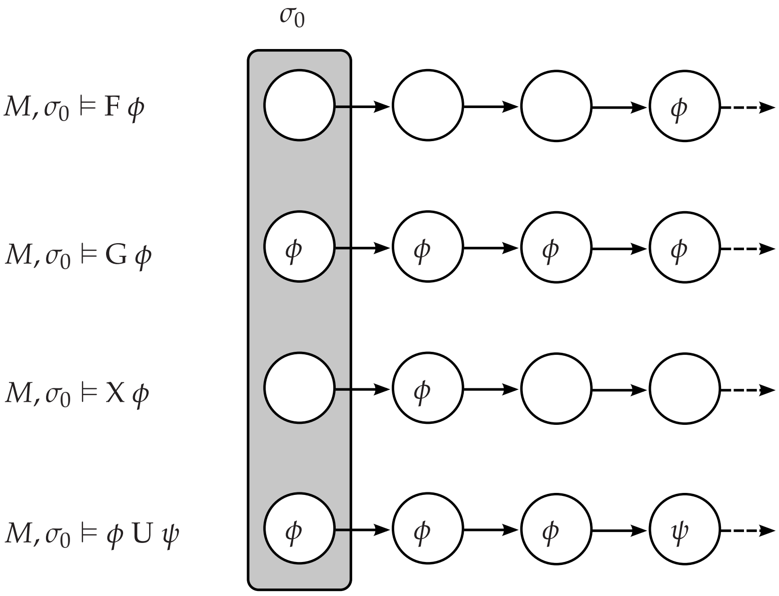

σ_0
$M, \sigma_0 \models \mathrm{F}\,\phi$
ϕ
$M, \sigma_0 \models \mathrm{G}\,\phi$
ϕ
$M, \sigma_0 \models \mathrm{X}\,\phi$
$M, \sigma_0 \models \phi\,\mathrm{U}\,\psi$
ψ

Figure 3.2: *Illustration of the LTL operation semantics.*

3.3.2 Computation Tree Logic (CTL)

Computation Tree Logic (CTL) [CE82] is branching time logic where, in contrast to the linear non-branching time domain of LTL, path quantifiers specify the validity of formulas on a branching infinite computation tree of future states obtained from unwinding the Kripke structure into this tree. The semantics of the LTL temporal operators remain unchanged but for each temporal operator, an associated path quantifier defines whether the expression shall hold on all possible paths of the computation tree or whether one path of the computation tree for which the temporal operator holds is sufficient for satisfying the CTL formula. The universal path quantifier “A” defines the former and the existential path quantifier “E” the latter requirement. A CTL formula can be a nested expression of operations as shown in the following CTL syntax grammar.

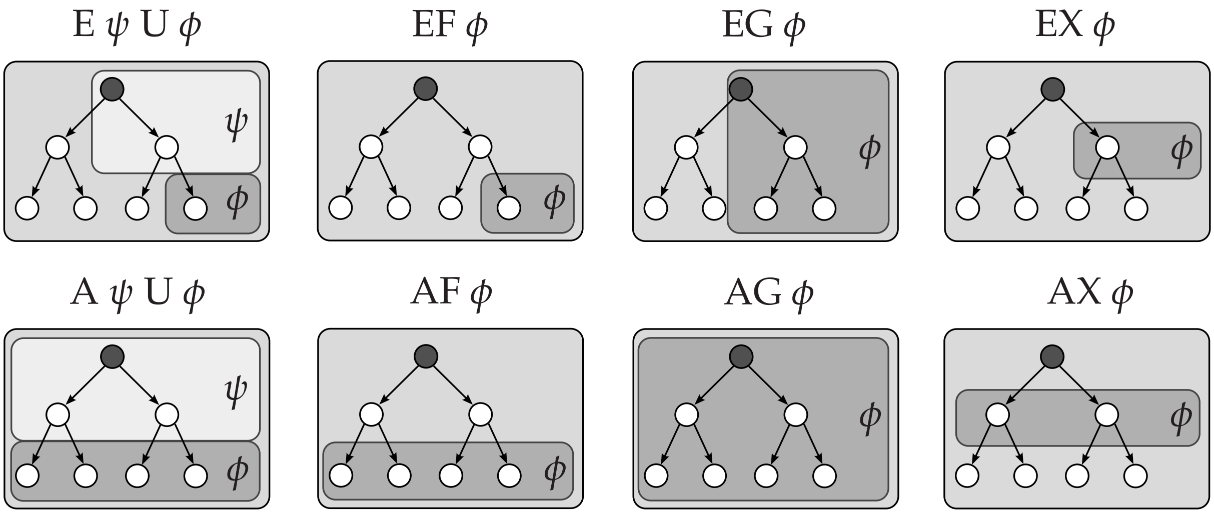


Figure 3.3: *Illustration of the CTL operation semantics.*

The syntax of a formula ϕ in CTL with atomic propositions a is defined by:

$$\begin{aligned} \phi = \quad a \quad & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\ & \mid \triangleright \diamond \phi \mid \triangleright \phi_1 \,\mathrm{U}\, \phi_2 \end{aligned} \tag{3.10}$$

An explanation of the placeholders for the language symbols of CTL is given in Table 3.2, and Figure 3.3 describes the operations visually such that the respective formula is satisfied for the root node.

Table 3.2: *Explanation of CTL syntax.*

a	atomic proposition	
$\triangleright$	path quantifier	A = on all paths (universal quantifier)
		E = on at least one path (existential quantifier)
$\diamond$	temporal operator	F = eventually
		G = generally
		X = next
U		U = until

Due to the logical duality of the universal and the existential quantifier, the universally quantified CTL-operations can be composed from the operations EX, EG and EU.

The universal quantifier postulates the validity of a proposition with respect to the temporal operator on all paths. The dual statement is that there exists no path, on which this proposition is not valid:

$$\forall a \equiv \neg\exists\neg a \;\; \Rightarrow \;\; \mathrm{AG}\phi \equiv \neg\mathrm{EF}\neg\phi \tag{3.11}$$

3.3.3 Property Specification Language (PSL)

Property specification of concurrent systems using CTL paved the way for introduction of formal methods in system design. Especially in the area of digital circuit design, model checking approaches using CTL were applied at the end of the 1990s. Following the first verification successes, the demand for a more designer friendly specification emerged. CTL turned out to be hard to write and read by non-specialists and an improved syntactic layer on top of CTL was introduced by IBM with their in-house specification language called "sugar" [BBDE$^+$01]. In addition to CTL operations, sugar was extended by sequential extended regular expressions (SEREs) to reason about more complex state transitions within Kripke structures. LTL was added as a basic temporal foundation, as digital simulation traces are strictly linear in time. Sugar was extensively used in verification of these simulation traces.

An industry consortium selected IBM's sugar to be the basis for a new standardized property specification language (PSL) and in 2005 PSL became IEEE standard 1850 [FMW05]. PSL builds a common specification and verification layer for direct interfacing with digital hardware description languages (HDLs) such as VHDL, Verilog, SystemVerilog and SystemC. PSL expressions are composed from a Boolean layer, a temporal layer and a verification layer.

The Boolean layer forms the atomic propositions using simple Boolean expressions interfaced with variables of the underlying HDL. In addition, local temporal operators are introduced in the Boolean layer for further refining propositions using operations such as `rose(a)` and `fell(a)`, which are true if in the previous step the variable `a` was 0 and now is 1 for `rose()` or vice versa for `fell()`.

The temporal layer defines behavior over time, forming properties from timed reasoning about Boolean layer expressions. Temporal operators can be the extended LTL semantics resulting in `always`, `never`, `eventually`, `next`, `before` and `until`. In addition, SEREs allow to specify more complex timed behavior. The expression $a; b; c$ for example specifies the non-overlapping sequence of b directly following to a and c directly following to b. For a detailed explanation of the expressiveness of SEREs please refer to [Con04] or [EF06].

The verification layer finally consists of directives specifying how the properties specified on the temporal layer should be evaluated by the verification back-end. The main verification layer directives are:

- `assert` for telling the verification back-end that a temporal layer expression shall hold,
- `assume` for constraining the verification to those states where the temporal layer expression is true,
- `restrict` for constraining design input sequences to get to a specific state before checking assertions,
- `cover` for directing the verification back-end to check if a specified path was covered by the verification and
- `fairness` for assumptions corresponding to liveness properties.

3.4 Existing Approaches to Specification of Analog System Properties

In contemporary analog design flows, verification is not formalized and so is the specification. Test bench-based characterization of circuit properties is comparing an informal specification, often given in form of a table, of allowed ranges of certain circuit properties such as maximum power consumption, slew rate, startup time, etc. to experimental evaluation of the DUV in a circuit test bench. Hence, there is no machine readable formalized specification methodology that guarantees a standardized verification approach. In fact, the incomplete and indirect specification of circuit properties leaves considerable freedom of interpretation to transfer the designer's intent into a circuit design as well as how to verify the specified properties. While in modern test benches, a set of simulation measurements is predefined in order to quickly characterize a given circuit, global quality management of the specification and verification is not yet widely introduced into design flows. Nevertheless, without a formalized detailed specification, several sources for design errors exist.

There are two fundamental domains of property specification which have a high impact on how the verification of the properties can be performed. On the one hand, for today's design flows, a signal-oriented temporal property specification is needed in order to describe the desired behavior of simulation results. On the other hand, formal verification methodologies for analog circuits consider circuit behavior to be analyzed in the circuit's state space. Therefore, specification of dynamic temporal analog properties for application in formal verification flows is required to consider the state space domain.

In the following, existing approaches to formalizing property specification of analog circuits in the signal domain as well as in the state space domain are discussed. In the subsequent section a synthesis of the findings will lead to the development of a new

Analog Specification Language (ASL) which is capable of specifying analog behavior in both the signal and the state space domain of the circuit.

3.4.1 Specification of Assertions within Analog Hardware Description Languages

Property specification and property verification are closely linked as the verification can be considered to be the evaluation of the specification. In the context of this thesis, a property specification is a stand-alone definition of desired system behavior that is created independently from the implementation to be verified. A possible application of hardware description languages (HDLs) for operational analog property specification is to have a specifying system set up for a circuit implementation and then comparing both implementations using an analog equivalence checking approach.

However, in the terminology of formal verification, declarative property specifications in a machine-readable language are distinguished from operational specifying implementations, as the former shall be more formal and lightweight. Moreover, the property specification shall define properties in a specific but sufficiently abstract way that the type of implementation is not restricted by the property specification. In general, these requirements are not satisfied by specifying implementations in HDLs.

While HDLs hence cannot be applied to formal specification of analog circuit properties directly, they incorporate internal concepts to specify properties of implemented behavioral description code blocks using assertions [Syn03, CVK05]. Therewith, statements can be inserted into the behavioral modeling code that can monitor internal properties of the design whether they comply with specified values [KZ04]. While this does not satisfy the requirement of independent specification, a set of assertions inserted into the HDL is a step towards property checking of the system during simulation runs. For example in VHDL-AMS the `assert` statement allows to have Boolean conditions trigger events controlled by a level of severity ranging from “note” to “failure” to which the simulator can react [APT03]. Moreover, information can be reported to the user giving debug information.

As the Boolean conditions can be generated from logical combinations of the comparison of time or signal values to specified constraints, complex assertions can be formulated. However, while motivating the development of assertion evaluations for analog circuits, the VHDL-AMS approach does not target an independent formal property specification of analog circuits. In Section 4.4.3 the verification part of the assertion-based approaches will be discussed in detail.

3.4.2 PSL for Analog Signal Property Specification

With the growing acceptance of PSL in the digital domain, approaches to apply the PSL specification methodology to analog signals for developing automated evaluation tools have emerged.

A syntax extension called Signal Temporal Logic (STL) to PSL was developed in order to verify properties on transient simulation waveforms [NM07, MNP08]. STL connects to the Boolean layer where threshold crossings of an analog signal represent Boolean events. Hence, the expressiveness of PSL/STL is mostly directed to timing verification and was demonstrated on a flash memory case-study [JKN10]. Another approach formulating analog specifications in form of recurrence equations directly by incorporating PSL sequential extended regular expressions as proposed in [SZDT07] can again express only very abstract, timing-oriented specifications.

3.4.3 CTL Specification of Analog Properties in the State Space

In order to have a rigorous specification of analog circuit properties to be formally checked on a discrete state space representation, an analog extension to CTL has been introduced in [HHB02a]. For application of the temporal methodology of CTL specifications to such discrete state space models of analog circuits, the atomic propositions have to be generated as they are not given by the modeling process. In the first approach to CTL-based analog specification, state sets are selected by constraining state space variable values to intervals using the operators "$<$" and "$>$". The inclusion of a state in a set is described by the Boolean variable labels which form the atomic propositions. If a state is a member of a set, its labeling by the Boolean variable of the set is true and otherwise false. For example, an atomic proposition can be generated from $V_{in} > 0.75$. Hence, all states for which the extended state space variable V_{in} has a value greater 0.75 are selected. The temporal logic CTL, extended by the operators "$<$" and "$>$", is defined as CTL-A in [HHB02a]. In order to specify explicit timed behavior for CTL-A operations, CTL-AT was introduced as an extension, offering to constrain the validity of CTL-formulas to a time interval[GPHB05]. This refined specification of temporal properties is derived from the concepts of Real-Time CTL (RTCTL) [EMSS92], where, for example, AF[1, 3](ϕ) selects only those states from the result of the operation that run into the set ϕ within the time interval [1, 3]. The syntax of formulas ϕ in CTL-AT is described in Equation 3.12 with the placeholders defined in Table 3.3.

$$
\begin{aligned}
\phi = \quad a \quad & \mid z * v \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
& \mid \triangleright \diamond \phi \mid \triangleright \phi_1 \,\mathrm{U}\, \phi_2 \\
& \mid \triangleright \diamond^{-1} \phi \mid \triangleright \phi_1 \,\mathrm{U}^{-1}\, \phi_2 \\
& \mid \triangleright \circ \Box \phi \mid \triangleright \phi_1 \,\mathrm{U}\, \Box \phi_2 \\
& \mid \triangleright \circ^{-1} \Box \phi \mid \triangleright \phi_1 \,\mathrm{U}^{-1} \Box \phi_2
\end{aligned}
\tag{3.12}
$$

Another important feature is the possibility to reverse the direction in which paths in the state space are processed by the CTL algorithms. Therefore, the syntax of a temporal logic expression can be extended by "$^{-1}$" which implies the reversal of all edge directions for this operation. With the additional labelings of the edges remaining unchanged, the temporal logic expression is then evaluated on the inverted transition relation R^{-1}:

$$\forall\,(\sigma_i,\sigma_j) \in R\,:\, R^{-1} = \bigcup(\sigma_j,\sigma_i) \tag{3.13}$$

Table 3.3: *Explanation of CTL-AT syntax.*

a	atomic proposition	
z	continuous state space variable	
v	number in $\mathbb{R}$	
$*$	analog operator	$<$ = smaller $>$ = greater
$\triangleright$	path quantifier	A = on all paths (universal quantifier)
		E = on at least one path (existential quantifier)
$\diamond$	temporal operator	F = eventually
		G = generally
		X = next
$\circ$		F = eventually
		G = generally
U		U = until
$^{-1}$	time reversal	paths are traveled in reverse direction
$\square$	time interval	$[t_{low}, t_{high}]$ = continuous time interval with $t_{low} \in \mathbb{R}_0^+, t_{high} \in \mathbb{R}_0^+ \cup \infty, t_{low} \leq t_{high}$

3.5 Analog Specification Language (ASL)

Property specification for analog circuits is fundamentally different to property specification for digital circuits. While in the digital domain logic functions and abstract properties like fairness have to be verified, in the analog domain properties such as slew rate, oscillation, startup times, etc. have to be considered. Temporal logics are not sufficient to express such properties.

Although analog operators in CTL-AT in combination with time constraints allow to specify some basic properties such as reachability of states from DC-operating-points, advanced specifications for analog properties cannot be formulated in a systematic way. This is on the one hand due to CTL-AT not allowing to acquire additional information about the execution of the operations. Therewith, intermediate results of the operations cannot be taken into account. This leads to the impossibility of sequential specification with operating on values measured during the evaluation of preceding operations. On the other hand, the solely temporal specification methodology itself is not suited for specifying complex analog behavior, where additional operations are needed to determine state space sets more sophisticatedly.

Therefore, a specification language is necessary that on the one hand allows to specify complex analog properties in a designer-oriented way and on the other hand can be mapped to formal verification algorithms that operate on a discrete state space model of the analog circuit under verification.

Due to its origin in temporal logics, CTL is not capable of offering a designer-oriented specification methodology. In order to gain acceptance for formal approaches in analog verification, a new methodology of property specification is necessary. While PSL offered this step towards designer-oriented specification in the digital domain, the first approaches to analog specification with PSL, as discussed in Section 3.4.2, are only covering signal-based properties for assertion-based verification. Hence, they are not suited for describing properties of analog systems in the state space.

In the following, a new Analog Specification Language (ASL) for state space-based property specification of analog systems is introduced, which, in combination with the corresponding ASL verification algorithms, will allow to extend the complexity of analog circuit properties specifiable for formal property verification.

ASL syntax shall be designed to be semantically deductive and therewith it shall reduce the time needed for understanding existing specifications in comparison to temporal logic specification. This is achieved by providing the possibility of creating parameterized macro functions involving a macro preprocessor. Hence, specification code can be sourced out to macro libraries allowing encapsulation and reuse of specification code. As will be presented, the application of ASL specifications and algorithms is not restricted to state space models generated with the approaches described in Section 2.4, so it can be adopted to other finite state machine-based modeling approaches like [FSS06]. Moreover, as will be shown in Section 5.3, ASL can be used to specify and verify properties on conventional transient simulation waveforms by transferring them to a state space representation.

3.5.1 Language Concept

From the point of view of analog circuit developers, analog properties are represented by continuous physical values and their alteration over time. Thus, it is necessary to select states by calculations on their state space parameter values. Whether a state belongs to a set is decided by comparing the result of an arithmetic calculation to a specified interval. Extended path operations abstract from the reachability analysis concept of temporal logics and allow examination of more complex properties on paths within the state space. Assigning measured values of the operations such as maximum and minimum path times to number variables allows to sequentially specify complex properties.

A high level assertion layer shall allow to evaluate a verification run without interpretation effort, additionally generating a verification report where all intermediate results can be inspected.

Based on the previously introduced concept, the following subsection defines a condensed Extended Backus Naur Form (EBNF) grammar for the syntax of ASL. Terminal symbols are printed in bold capital letters, user-defined variables are printed italic. Some epsilon-productions are omitted for the purpose of clarity. In Section 3.5.3, the semantics of the operations will be described.

3.5.2 EBNF Grammar of ASL

ASL_Specification :=

Spec_Sequence **QUIT**; | **QUIT**;

Spec_Sequence :=

Spec_Expression | Spec_Sequence Spec_Expression

Spec_Expression :=

SETVAR *Set_Variable*;

| **NUMVAR** *Number_Variable*; | *Set_Variable* = Set_Expression;

| *Number_Variable* = Number;

| **CALCULATION** *Calc_Name* (Calc_Expression);

| **FOR** Set_Expression **ASSERT** Set_Expression;

| **FOR** Number **ASSERT** Interval;

| *Number_Variable* = **FOR** Set_Expression **ASSERT** Set_Expression;

| *Number_Variable* = **FOR** Number **ASSERT** Interval;

Set_Expression :=

ON Base_Set Operation_Set | Operation_Set

Base_Set :=

Elementary_Set | *Set_Variable* | *State_Space_Variable* Interval

| (Base_Set) | **NOT** Base_Set | Base_Set **AND** Base_Set

| Base_Set **OR** Base_Set

Operation_Set :=

Elementary_Set | *Set_Variable* | *State_Space_Variable* Interval

| (Operation_Set) | **NOT** Operation_Set

| Operation_Set **AND** Operation_Set

| Operation_Set **OR** Operation_Set

| *Calc_Name* (calc_parameters) Interval

| **VALUE** (*State_Space_Variable*) Interval

| **ASSIGN** (*Number_Variable*, Assign_Type) Operation_Set

| **SELECT** Operation_Set

| **OSCILLATION** | Temporal_Logic_Expression Operation_Set

| **DELTA_COMPARE** (*State_Space_Variable*) Interval **FROM** Operation_Set **TO** Operation_Set

| **DELTA_COMPARE** (*State_Space_Variable_1, State_Space_Variable_2*) Interval **FROM** Operation_Set **TO** Operation_Set

| **TRANSITION FROM** Operation_Set **TO** Operation_Set

| **COUNTEREXAMPLE FROM** Operation_Set **TO** Operation_Set

| **INPUTSTIMULI FROM** Operation_Set

Elementary_Set :=

ALL | **DCPOINTS**

Interval :=

[Number, Number] | [< Number] | [<= Number] | [> Number] | [>= Number] | ε

Temporal_Logic_Expression :=

Temporal_Logic_Operator Interval Direction Operation_Set

| **ALWAYS** Direction Operation_Set **UNTIL** Interval Operation_Set

| **A** Direction Operation_Set **U** Interval Operation_Set

| **EXISTS** Direction Operation_Set **UNTIL** Interval Operation_Set

| **E** Direction Operation_Set **U** Interval Operation_Set

Temporal_Logic_Operator :=

UNIVERSALLY | **AG** | **EVENTUALLY** | **AF**

| **STAY** | **EG** | **REACH** | **EF**

Direction :=

ε | **FROM** | **ˆ -1**

Calc_Expression :=

Number | calc_parameter N | (Calc_Expression)

| Calc_Expression Math_Operator Calc_Expression

Number :=

Floatingpoint_Constant | *Number_Variable*

| (Number) | Number Math_Operator Number | **ABS** (Number)

Assign_Type :=

MAX | **MIN** | **AVERAGE** | **RANGE**

Math_Operator :=

+ | - | * | /

3.5.3 Semantics of ASL Operations

In the following, the semantics of the main ASL operations are described. The methodology descriptions of Section 5.2 will provide additional insights into the application of ASL specifications.

There are two types of sets that influence the evaluation of ASL operations: Base_Set and Operation_Set. A Set_Expression in ASL can either be evaluated on all states of the DATS or the evaluation can be constrained to operate only on a subset of the states of the DATS which is then selected by a Base_Set. Hence, a Set_Expression that has the Syntax “ON Base_Set Operation_Set” will determine the results of the Operation_Set only on the set of states identified by Base_Set. The keyword “SELECT” has no semantic function and is only used for syntax beautification.

VALUE: With the operation “VALUE”, a set of states ϕ can be selected on the whole state space or on another set by a state space variable constrained to a specified interval. Algorithmically, for each state σ_i is decided whether it is included within the given interval by a comparison of the actual value $p_i^{(s)}$ of the entry in position s in the state space variable value vector $\mathbf{p}_i = L_V(\sigma_i)$ and the interval boundaries $\underline{r}$ and $\overline{r}$:

$$\sigma_i \in \phi \Leftrightarrow p_i^{(s)} \in [\underline{r}, \overline{r}] \tag{3.14}$$

The worst-case runtime complexity of the value operation is $\mathcal{O}(n)$ with n being the number of states in the Base_Set the operation is applied to. This is due to a single loop iterating over the set of states and deciding the inclusion in the interval boundaries.

Temporal logic expressions: Although the temporal logic operations of CTL-AT can be used in ASL, as a language convention, their natural language equivalents shall be used for better understandability of the ASL syntax. Therefore, for the six basic combinations of path quantifiers and temporal operators in CTL, a natural language equivalent has been selected in order to correspond to the syntax style of ASL. The ASL syntax grammar in the previous section introduces the natural language operation syntax, each followed by its CTL equivalent. For example, the ASL operation "UNIVERSALLY" maps to its CTL equivalent "AG".

The worst-case runtime complexity of the temporal logic operations is closely connected to graph traversal algorithms such as depth-first search or breath-first search used for implementing the temporal logic operations in ASL. In the worst-case, for every state, every path to all other states in the graph has to be checked. Due to a marking of visited states and edges with the already acquired information, in the worst-case due to the reuse of information, a quadratic runtime complexity $\mathcal{O}(n^2)$ with n being the maximum of the number of states and edges is given. A nesting of operations introduces an addition of the complexities of the single operations and hence does not increase the asymptotic complexity. For using time intervals in the operations, the semantics of CTL-AT apply.

TRANSITION: Previous approaches to property specification of analog systems were directly derived from temporal logics and mostly perform some kind of reachability analysis. Although CTL-AT operations are still possible in ASL, the abstract reachability operation of ASL is called "TRANSITION" and selects states on paths between two state areas. It determines the minimum, maximum, average and the range of the transition times detected on the transition paths. These values can be assigned to numeric variables using the "ASSIGN" command. The sum of edge weights on a path between two vertices i, j is defined as distance and is calculated by Dijkstra's algorithm once as shortest path and once as longest path by inversion of the edge weights. The longest path computation is possible efficiently as the path algorithms are modified to consider the DATS to be a directed acyclic graph where a labeling function assures not to travel loops. A pseudocode definition is given in Algorithm 2 and an illustration of the states identified by the transition operation is shown in Figure 3.4(a).

Algorithm 2: ASL Transition Algorithm: ON Base_Set TRANSITION FROM Start_Set TO Dest_Set

```
foreach vertex i in (Start_Set ∩ Base_Set) do
    foreach vertex j in (Dest_Set ∩ Base_Set) do
        if shortest and longest distance($i \to j$) $< \infty$ then
            add shortest and longest distance($i \to j$) to transition times;
            add vertices on paths between $i \to j$ to Transition_Set;
        end
    end
    report minimum, maximum, average and range of the detected transition
    times.
end
```

Algorithm 3: ASL Oscillation Algorithm: ON Base_Set SELECT OSCILLATION

```
foreach vertex i in Base_Set do
    if (shortest and longest distance($i \to i$) $> 0$ and $< \infty$) then
        add shortest and longest distance($i \to i$) to oscillation periods;
        add vertices on paths $i \to i$ to Oscillation_Set;
    end
    report times of minimum, maximum, average and range of the detected
    oscillation periods.
end
```

When considering the edges to be all labeled with a transition time of 1, the minimum transition time is 4 and the maximum transition time is 6.

As can be concluded from Algorithm 2, the worst-case runtime complexity of the transition operation is dominated by the all-pairs shortest path problem which can be computed in $\mathcal{O}(n^3)$ with n being the number of vertices of the DATS.

OSCILLATION: The operation "OSCILLATION" identifies states on cycles in the state space and calculates the corresponding minimum, maximum and average oscillation period. The pseudocode definition is presented in Algorithm 3. In Figure 3.4(b) a schematic illustration of a set of states of an oscillation cycle is shown with an shortest oscillation period of 11 and longest period of 12 when considering the edges all to be labeled with a weight of 1. The cycle detected from state i is identified due to i being reachable from i with a path length greater zero and less than infinity when a single self-transition is ignored using a modified path traversal algorithm. The worst-case runtime complexity of the oscillation detection algorithm is $\mathcal{O}(n^3)$ with n being the number of states in the Base_Set the operation is applied to. This is due to the oscillation operation relying on distance information generated by a modified

Algorithm 4: ASL Delta_compare Algorithm: ON Base_Set DELTA_COMPARE (State_Space_Variable) Interval FROM Start_Set TO Dest_Set

```
foreach vertex i in (Start_Set $\cap$ Base_Set) do
    foreach vertex j in (Dest_Set $\cap$ Base_Set) do
        if shortest and longest distance($i \rightarrow j$) $< \infty$ then
            add vertices on paths $i \rightarrow j$ to Transition_Set;
        end
    end
end
foreach vertex pair $i, j$ with $(\sigma_i, \sigma_j) \in R$ in (Transition_Set) do
    if $\frac{\Delta value(State\_Space\_Variable)_{ij}}{\Delta time_{ij}} \in Interval$ then
        add $i, j$ to Result_Set;
    end
end
report minimum, maximum, average and range of $\frac{\Delta value(State\_Space\_Variable)}{\Delta time}$
```

path traversal algorithm such as Dijkstra's with, in the worst case, n calls to the path algorithm with a quadratic runtime complexity. Corresponding to the explanation for the transition operation, the longest path detection operates on inversed edge weights and does not travel loops due to a labeling of visited edges. However, closing a loop from vertex i along a path to i is allowed by a modification of the algorithm.

DELTA_COMPARE: The operation "DELTA_COMPARE" evaluates $\frac{\Delta value}{\Delta time}$ between all pairs of consecutive states of detected paths within a given transition area. Hence, it is possible to measure the rate of change of a state space variable value over time on paths. This is of particular use for determining slew rates on transition paths. A pseudocode definition is introduced in Algorithm 4. Considering the two state space variables a and b in Figure 3.4(c). For variable b, the algorithm calculates $\frac{b_2 - b_1}{2}$ and $\frac{b_3 - b_2}{3}$.

The worst-case runtime complexity is dominated by the computation of the transition set with $\mathcal{O}(n^3)$. The calculation of the derivative only takes linear time, hence its worst-case runtime complexity is $\mathcal{O}(n)$.

Moreover, by passing two variables to the operation, the partial derivative $\frac{\Delta variable1}{\Delta variable2}$ between the two variables can be measured. Referring again to Figure 3.4(c), the algorithm calculates $\frac{b_2 - b_1}{a_2 - a_1}$ and $\frac{b_3 - b_2}{a_3 - a_2}$.

CALCULATION: By defining a calculation formula, a set of states can be selected by evaluating an arithmetic property on the state space variable values of each state in the Base_Set. Additionally, the numerical calculation results can be used by subsequent

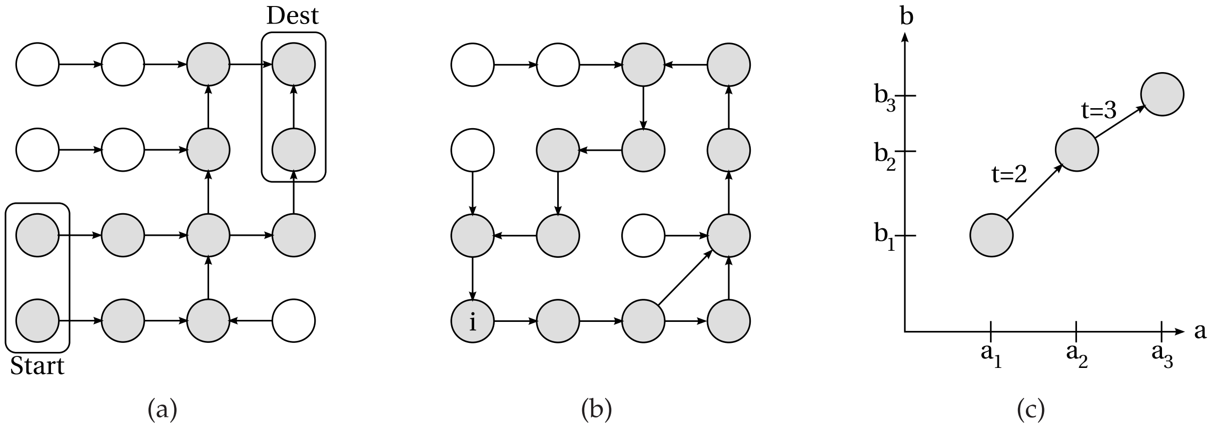


Figure 3.4: *States in the transition set between sets "Start" and "Dest" (a). States of an oscillation set identified by the oscillation algorithm (b). Illustration of the delta_compare operation (c).*

operations. Algorithmically, the formula is calculated on the variable values of each state and therewith decided if it meets the given value constraint while recording the calculation results. A definition of a calculation contains placeholders for each parameter which will later be passed on to the calculation.

For example, the calculation formula

```
calculation fraction (calc_parameter2 / calc_parameter1 )
```

can be defined. An application in the body of the ASL specification could be:

```
result_set = on test_set fraction(V_C1,V_C2)[0.2,0.5]
```

In this code line, for the two state space variables V_{C1} and V_{C2} for every state in the set test_set, the division as presented in the definition of the formula is computed. If the result for a state is within the exemplary interval $[0.2, 0.5]$, the state is inserted into the result set. With such calculations, complex properties can be defined.

The call to a calculation has a worst-case runtime complexity of $\mathcal{O}(n)$ with n being the number of states in the Base_Set as the evaluation of the calculation formula can be computed by a single loop iterating over the set of states.

DCPOINTS: The operation "DCPOINTS" returns the steady states represented by DC-operating-points of the state space. For different input values, the corresponding DC-operating-points determine the set ϕ_{DC} of steady states of the system. This set is directly identified by the discretization process. The DATS represents DC-operating-points with a self-transition and no other outgoing dynamic edges:

$$\sigma_i \in \phi_{DC} \Leftrightarrow (\sigma_i, \sigma_i) \in R \wedge \deg(\sigma_i) = 1 \tag{3.15}$$

ASSIGN: The operation "ASSIGN" allows to assign a numerical value returned by another ASL operation to a number variable. The possible values are minimum, maximum, average and range of the set of single values determined on a set of states.

ASSERT: To complete the verification of a property, it is necessary to include assertions in the specification code. These assertions evaluate to either the Boolean result true (1) or false (0), which is always printed to the verification report and in addition can be optionally assigned to a number variable for further evaluation. For sets, the operation "ASSERT" checks whether a given set is the subset of the reference set. Hence, "FOR ϕ ASSERT ψ" reports true if the following equation holds:

$$\phi \cap \psi = \phi \tag{3.16}$$

Numeric assertions check values determined during the verification process with respect to a given interval. Therewith, the assertion returns true if the number variable in the expression is within the specified interval boundaries.

The worst-case runtime complexity of assertions is $\mathcal{O}(n)$ with n being the number of states in the Base_Set for the evaluation of set assertions and $\mathcal{O}(1)$ for the evaluation of number assertions.

COUNTEREXAMPLE: The operation "COUNTEREXAMPLE" starts the counterexample generation algorithm which will be detailed in Section 5.4.

INPUTSTIMULI: The operation "INPUTSTIMULI" starts the complete state space-covering input stimulus generation algorithm which will be detailed in Section 5.5.

4

Verification of Systems

The goal of electronic circuit design is to create an implementation that completely satisfies the circuit specification. Therefore, at several stages within the design process, the current state of the design has to be checked whether it satisfies the initial specification for the circuit. This task can be accomplished in different ways.

After presenting the general definitions and methods that distinguish verification concepts, existing approaches to analog non-formal and formal verification are discussed in this chapter.

Definition 4.0.1 (Verification)
Verification of a system compares the performances of a model of the system with specified properties either in order to detect design errors (non-formal verification) or to prove the correctness of the system with regard to the specified properties (formal verification). The general term verification refers to non-formal verification.

The level of formality of the specification as well as of the conducted verification efforts ranges from informal and incomplete with experimental character to proof-based formal approaches with absolute certainty of the verification results. Consequently, a classification of the different verification approaches is necessary, distinguishing non-formal from formal verification and introducing the corresponding approaches and terminologies.

In the following, definitions and existing methodologies are discussed in order to develop a hierarchy of verification approaches to prepare the introduction of new analog formal verification methodologies in the next chapter that allow formal *and* analog design-oriented verification.

4.1 Non-Formal Verification

Under the notion of non-formal verification all those approaches to verification are summarized that do not guarantee completeness of the verification.

Definition 4.1.1 (Non-Formal Verification)
Non-formal verification of a circuit conducts a finite number of simulations in order to detect circuit performances that do not meet the specification. A successful verification only holds for the specific input conditions and internal states of the circuit that were covered by the n simulation runs:

$$\neg\exists\, \mathbf{f}_i(S,P) \nvDash P_{spec} \text{ for all } 1 \leq i \leq n \tag{4.1}$$

A successful verification using a non-formal approach does not prove the absence of errors as it is only a sequence of incomplete experiments. Experiments can remain for which the investigated property causes the verification to fail.

4.1.1 Simulation-Based Verification

The most common way to check whether a system conforms to its specification is to simulate test cases in order to obtain performances that are considered representative for its future operation. A set of simulation test benches is set up and by conducting the simulations, the system behavior is compared to a specification. While design errors can be detected by this approach, it is never known when enough test cases have been simulated to consider a system as error free. Moreover, the evaluation of simulation results and comparison with specified properties is a manual task. Because of the specification not necessarily being conformant to any specification standard, there is a high level of uncertainty whether the specification is interpreted correctly by the designer. The simulation flow is illustrated in Figure 4.1.

4.1.2 Assertion-Based Verification

In contrast to simulation-based verification using test benches with manual inspection of the results, assertion-based verification tries to overcome some downsides of the previous approach. This is accomplished by introducing formalization of the specification and the evaluation of simulation runs in order to reduce the manual effort and to increase standardization of the verification approach. The automation is achieved by having system properties defined in a machine readable specification which can

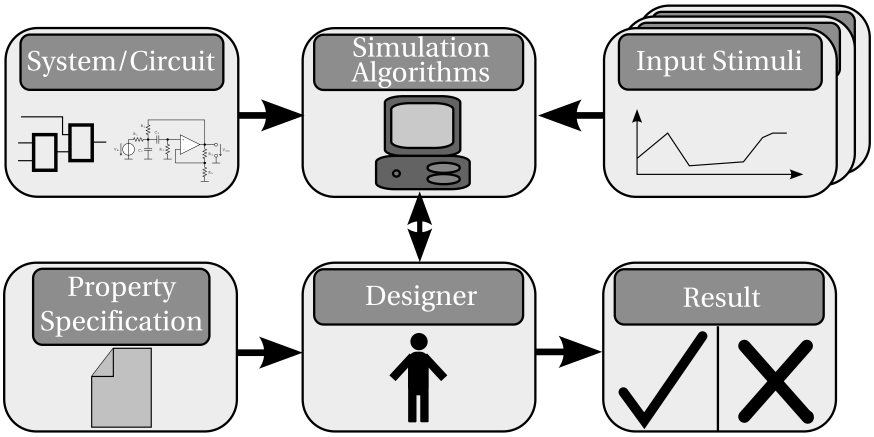


Figure 4.1: *Simulation-based verification flow. The system is simulated using simulation algorithms with user-defined input stimuli. The designer compares the simulation results with the property specification of the system and either considers the specification as satisfied or not.*

be processed by an evaluation tool on the simulation waveforms. This evaluation can be directly linked to the simulation environment, resulting in incrementally online-monitoring the simulation results. In the case of specification violations, the simulation can be interrupted immediately. Therewith, time can be saved by not finishing long simulation runs when an error is detected early during the simulation. The offline-monitoring approach receives the complete simulation results after finishing the simulation. While the advantage of online-monitoring is obvious, algorithmically, some propositions about the simulation results can only be checked when the complete results are available. This applies for example to averaged values or the number of crossings of a given threshold.

While the assertion-based verification approach does not cover all possible states of a system due to incorporating incomplete experiment-based simulation, using a machine-readable specification approach and automated evaluation improves the verification quality significantly. Figure 4.2 illustrates the assertion-based verification flow.

4.2 Verification Coverage

One common challenge among all non-formal verification approaches is verification coverage. The question when a verification is complete and a design can be considered as error-free cannot be answered confidently. As test bench-based simulation approaches are based on a finite set of simulation cases, the number of detected design errors should decrease with the number of simulations performed. The decision

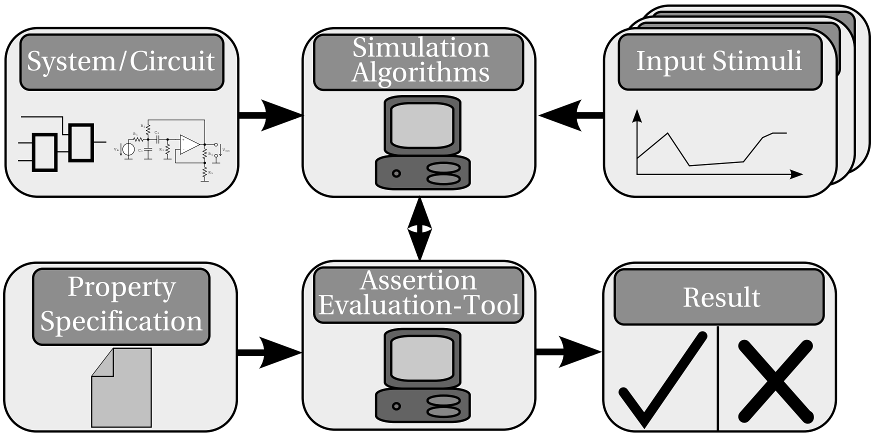


Figure 4.2: *Assertion-based verification flow. The system is simulated using simulation algorithms with user-defined input stimuli. The assertion-evaluation tool compares the simulation results with the machine-readable property specification of the system and reports whether the specification is satisfied by the simulation results.*

when enough simulation runs have been conducted depends on the designer's experience, and deducing that a design considered to be simulated sufficiently can be taken as error free is a common misconception. It can still contain numerous critical design errors that just have not been covered by the test cases designed by the verification engineer. For digital as well as for analog hardware verification, exhaustive simulation is considered not to be possible efficiently and therefore verification coverage of test bench-based approaches is not complete.

Theoretically, digital designs have a finite set of states that can be enumerated and test cases could be constructed that cover every possible state the system can adopt. However, even for very small designs this effort is infeasible due to the combinatorial explosion. Hence, measures for verification coverage are mandatory in order to rate the verification quality and actively control which parts of a design are not covered sufficiently.

Besides the aforementioned challenge of never knowing whether or not a given specification is completely satisfied by the DUV, another challenge even valid for formal verification approaches using declarative property specifications exists. It is posed by the question, whether the formal property specification is sufficient for capturing the design intent. Even a perfectly formally verified system can contain errors if the specification did not cover them. This problem is called specification coverage and will not be considered in the following as we assume that a verification is complete when the specification is satisfied by the DUV under any circumstances.

Analog verification coverage has not yet been subject of extensive research and only few approaches exist that try to transfer some of the digital coverage measures to mixed digital-analog verification [BCMP06, HLSS08], or in the area of hybrid systems using sensitivity analysis for coverage-directed simulation [DM07]. In the digital hardware domain, a taxonomy of verification coverage already exists. Tools supporting verification management methodologies, subsumed under the heading "coverage-driven verification", have emerged for handling the verification of large designs. Languages such as OpenVera [Syn03] and SystemVerilogAssertions [CVK05] include assertion management and coverage assistance and tools like Cadence Incisive Enterprise Specman [Cad07] offer a systematic verification coverage management within the verification flow. The corresponding ideas will be outlined in the following in order to motivate an analog verification coverage concept.

In the field of non-formal verification, there are two main coverage metrics: structural/code coverage and functional coverage [GRW05]. The main observation of these metrics is that only a subset is connected to direct specification checking and most of the coverage metrics are indirect, which means that they do not conclude specification conformance from their results but help to decide whether one can be confident in the verification results.

4.2.1 Structural Coverage

Structural coverage describes the implementation coverage of a verification. The objective is to measure which parts of the implementation have been covered by verification. Depending on the type of implementation, this can be performed in different ways. Structural coverage can vary from code statement coverage, including branch coverage and path coverage for HDL-like implementations, to finite state machine coverage, including state coverage and transition coverage in order to identify states and transitions visited by the verification.

While an automated evaluation of structural coverage can be obtained by applying simple algorithms, the results are not always very meaningful. Checking that every part of the implementation has been covered by a verification run does not mean that critical interactions of the parts have been covered. As an example, an abstract system consisting of four parts with directed edges representing a signal flow is illustrated in Figure 4.3. In a first simulation run labeled "a", parts 1, 3 and 4 are covered. The second run "b" covers parts 1, 2 and 4. However, the connection between part 2 and 3 was not covered by the verification. Although 100% of the parts have been covered, a possible problem in the interoperation of parts 2 and 3 would not have been detected by this verification. On the other hand, simulating all possible combinations is not feasible for larger designs. Therefore, coverage of corner cases cannot be concluded from high structural coverage metrics.

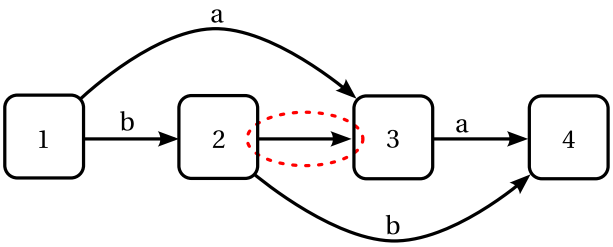


Figure 4.3: *Illustration of structural coverage.*

4.2.2 Functional Coverage

In contrast to the non-specific structural coverage, functional coverage is measured by checking whether predefined functional aspects have been part of the verification run. Therefore, explicit coverage points have to be specified that represent critical corner cases or requirements of the design, based on the knowledge of the verification engineer. Revisiting the example presented in the previous subsection, a verification engineer could consider the combination of parts 2 and 3 as important. Hence, a functional coverage measure is specified that reports whether this path has been covered by the verification. By covering user-defined functional requirements, the confidence in the verification results is improved.

Functional coverage measures are closely related to the assertions specified for assertion-based verification approaches. By specifying the functional requirements using such assertions and having all those assertions covered by the verification, a successful assertion-based verification is obtained. On the other hand, passing of assertions may not be confused with covering of assertions. While an assertion can pass without ever being triggered, a covered assertion has to be evaluated and either passes or fails. Moreover, additional information can be gathered by counting the evaluation of assertions and the internal value combinations leading to these evaluations.

4.3 Formal Verification

Formal verification describes the verification methods that are based on formal concepts allowing to prove that propositions made about a system model are valid. Such propositions can be in form of a declarative specification or an operational specifying system model as discussed in Section 3.3.

Definition 4.3.1 (Formal Verification)
Formal verification of a system proves that the performances satisfy the property specification for every possible input signal and internal state of the circuit:

$$\forall\, \mathbf{f}(S,P) \;:\; \mathbf{f}(S,P) \vDash P_{spec} \tag{4.2}$$

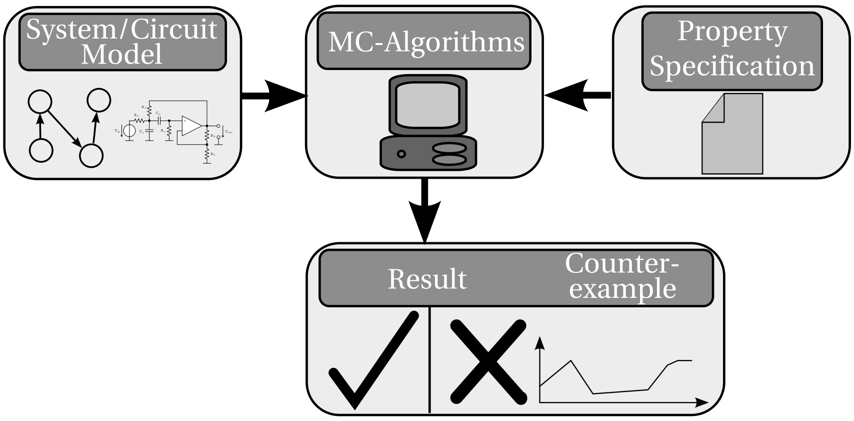


Figure 4.4: *Model checking flow. The machine readable property specification is checked against the system model using a model checking tool. The result is either true, i.e. the specification is satisfied, or a counterexample is returned.*

Historically grown, two classes of algorithmic approaches to formal verification are distinguished. The algorithmic approach of proving that a system model satisfies a specified declarative specification is called model checking. Proving the functional equivalence of a system model under verification and an operational specifying system model is referred to as equivalence checking. In the terminology of the coverage aspects of the previous section, formal verification has complete structural and functional coverage with respect to the specification.

4.3.1 Model Checking

Model checking is an approach to formal verification where a system model is compared with a functional specification [CGP99]. Hence, the task of model checking is to verify whether a finite state system model represented by a type of Kripke structure M as defined in Section 2.2.1 and 2.4.1 satisfies a given specification P_{spec}. The declarative languages for denoting the specification have been introduced in Chapter 3. Therewith, the verification problem to be solved can be stated as:

$$M, \sigma \models P_{spec} \text{ for all } \sigma \in \Sigma \tag{4.3}$$

The model checking algorithm reports true if P_{spec} holds for every state of the system model. Otherwise, states not satisfying the specification are identified. A transition path π_{ce} from a defined initial state to a state identified as not satisfying the specification is called counterexample. Figure 4.4 illustrates the model checking approach.

While the foundations of using propositional temporal logics for specifying system behavior originated in philosophy long before their application to algorithmic system verification, the combination of CTL property specification and a system model represented as an explicit state transition system in form of a Kripke structure made possible the first automated model checking of an abstract system model [CE82]. Before these algorithms for model checking have been proposed, all verification proofs were conducted by manual calculations and hence were infeasible for real world applications.

4.3.2 Equivalence Checking

While model checking is applicable to prove that a machine-readable specification of system properties is satisfied by a model of the system implementation, the task of equivalence checking is to compare two implementations of a system whether they are functionally equivalent. Hence, equivalence checking does not verify that a system conforms to a set of properties, but instead complete behavioral equivalence is proven.

In the digital domain, equivalence checking has a long history and there is a very strict formal definition for functional equivalence of static combinatorial circuits [MM04]:

Definition 4.3.2 (Boolean Equivalence of Combinatorial Circuits)
Given two representations d_f *and* d_g *of two Boolean functions* f, g*:* $\{0,1\}^n \rightarrow \{0,1\}^m$*, decide whether the Boolean functions* f *and* g *are equal, i.e. whether* $f(\alpha) = g(\alpha)$ *holds for all* $\alpha \in \{0,1\}^n$*.*

The general case of the Boolean equivalence checking problem is co-NP hard [GJ79] as it represents the complemented Boolean formula satisfiability problem which is known to be NP hard. The advances in the field of digital equivalence checking were made by finding representations of Boolean formulas such as reduced ordered binary decision diagrams (BDDs) [Ake78] or multiplicative binary moment diagrams (*BMDs) [CB95] that reduce the actual equivalence checking problem down to constant time, at the cost of creating the underlying representations.

For dynamic sequential circuits, the equivalence checking problem is related to checking the equivalence of the finite automata representations of the circuits. Hence, a state space traversal approach has to compare the reachable sequences of states of both automata for defined starting states [MM04].

In the scope of this thesis, the concept of functional equivalence of static and dynamic systems is generalized towards equal input/output behavior of analog systems with the following definition.

Definition 4.3.3 (Input/Output Equivalence of Static/Dynamic Systems A **and** B**)**
Two systems A *and* B *are considered as input/output-equivalent if for every possible*

input signal $\mathbf{u}(t)$*, the outputs of system* A *and* B *are equivalent with respect to the dynamic output behavior defined in Definition 2.4 in Section 2.1:*

$$A \equiv B \;\Leftrightarrow\; \mathbf{y}_A(t) \equiv \mathbf{y}_B(t) \;\text{ for all } \mathbf{u}(t) \tag{4.4}$$

with

$$\mathbf{y}_A(t) = S_A(\mathbf{u}(t), \mathbf{x}_A(t))\,;\, \mathbf{y}_B(t) = S_B(\mathbf{u}(t), \mathbf{x}_B(t)) \tag{4.5}$$

In addition to the question whether two systems are equal, which is a Boolean result of true or false, equivalence metrics have to be developed for equivalence checking of analog circuits. This is due to the fact that there is no total equivalence in real analog circuits as it exists for the Boolean equivalence of digital systems. Hence, equivalence of analog circuits is assumed when a given equivalence measure, reported by a metric to be defined, is above a given magnitude.

4.4 Existing Approaches to Non-Formal Analog Circuit Verification

While the presented general verification concepts for non-formal verification apply to analog circuits, there are several differences that have to be discussed. Especially the availability of automated tools for analog verification is significantly reduced compared to digital circuit verification. While there are mature verification methodologies for digital circuits that are already introduced in industrial design flows, analog verification is mainly done by manual investigation of simulation runs. Due to the complex properties of analog circuits, not only the specification of the desired behavior of the circuit is much more difficult than in the digital domain, but also the verification algorithms are far more complex due to operating on continuous systems represented by DAEs. Therefore, circuit simulations are the major tool for verification of analog circuits, nowadays partially augmented by automated evaluation of verification results.

This section addresses the concepts of formalizing non-formal verification of analog circuits, followed by a discussion of existing approaches to analog formal verification in the subsequent section.

4.4.1 Assertion-Based Approaches to Analog Verification

The first approaches to overcome the manual evaluation of analog simulation results were introduced in the area of automated circuit characterization [HGT91, EGG98]. Therewith, by developing reusable templates for circuit class specific properties, automated performance evaluations have been introduced into the analog design flow.

An emerging verification approach attempts to formalize the property specification and evaluation of conventional simulation results introducing assertions. Derived from the digital domain, assertion-based verification automates the evaluation of simulation results and hence enables regression testing. A recent approach to include analog assertion-based verification on commercial platforms proposes property specification implemented either as an analog extension to SystemVerilog Assertions (SVA) or a library of analog assertion objects for the Open Verification Library (OVL) [MPDG09].

The tool AMT proposed in [NM07] uses STL/PSL in order to verify properties on transient simulation waveforms offline and online. However, due to the limited expressiveness of STL/PSL, only timing oriented verifications are possible.

In Section 5.3 a new assertion-based verification methodology for complex analog properties such as slew rate, oscillation and overshoot by application of the Analog Specification Language (ASL) to transient simulation waveforms will be introduced. Due to the global evaluation methodology making possible the verification of complex properties on the waveforms, online monitoring is only reasonable for local properties.

4.4.2 Analog Verification Coverage

While formal verification methodologies automatically deliver complete coverage of the investigated state space of the designs under verification, the verification gap of today's design flows can simply be characterized by the idea of verification coverage. Test bench-based simulation approaches try to find critical input stimuli and external parameters that bring the circuit in as many critical states as possible, but the real verification coverage of the state space is uncertain. Therefore, in the area of post-production testing of analog circuits, several approaches to automatic test pattern generation (ATPG) have emerged for a systematization of the test procedure. Their common method is to start with a set of given faults and trying to compute a test stimulus that covers every element of the fault set using either sensitivity analysis [Bur01], controllability and observability computation [SHZ$^+$01], or statistical distance computation [VdPG97]. In the area of hybrid systems, coverage-guided test generation is emerging for linear systems, but applications to nonlinear analog circuits are not yet available [DN09].

Shifting the perspective back to design-time verification, automated approaches are very rare. An approach [BCMP06] of generating constrained randomized stimuli cannot guarantee to cover the complete state space of the design under verification (DUV). The same applies to another approach that simulates the system behavior with statistical chaotic excitation signals [MZXA08].

In the scope of this thesis, analog verification coverage can be connected to state space coverage. If every state of a system was investigated during a verification, the verification coverage is complete. Hence, in Section 5.5, an approach to complete state-space covering input stimuli generation will be proposed, allowing transient simula-

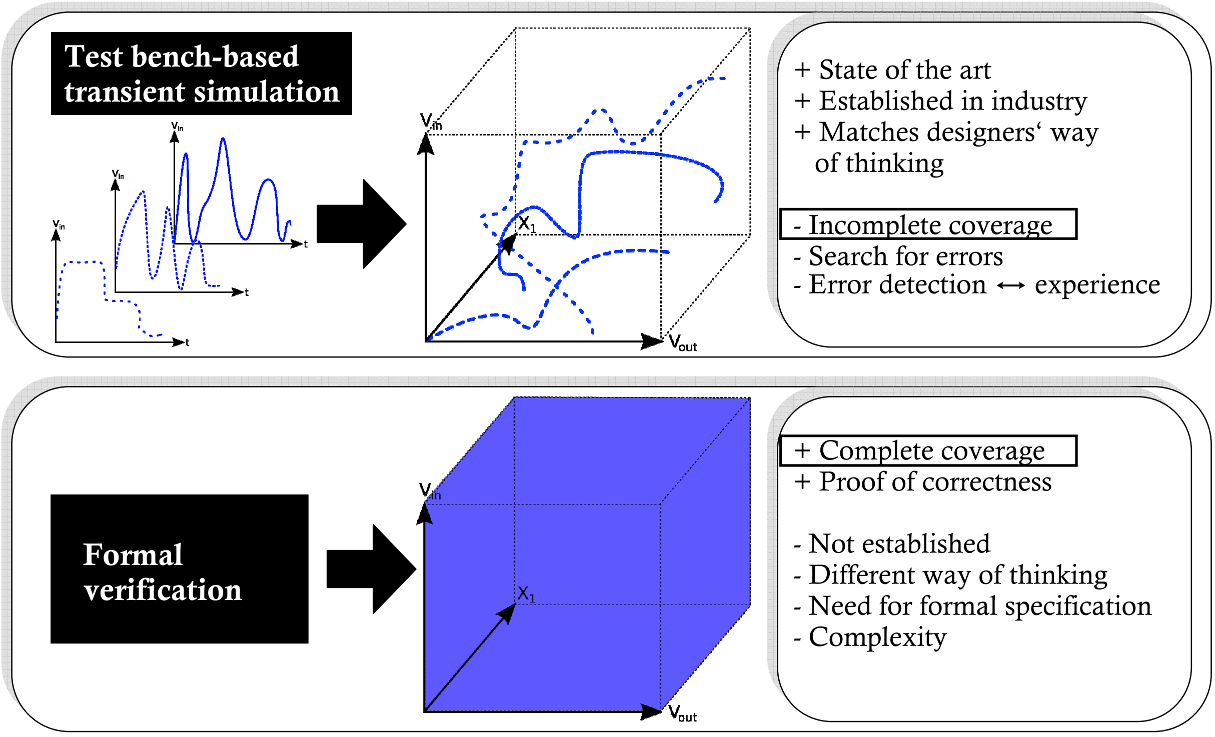

Test bench-based transient simulation
Vin
t
Vin
t
Vin
t
Vin
X1
Vout
+ State of the art
+ Established in industry
+ Matches designers' way of thinking
- Incomplete coverage
- Search for errors
- Error detection ↔ experience
Formal verification
Vin
X1
Vout
+ Complete coverage
+ Proof of correctness
- Not established
- Different way of thinking
- Need for formal specification
- Complexity

Figure 4.5: *Comparison of verification coverage in the state space by test bench-based transient simulation and formal verification.*

tion of nonlinear analog circuit blocks with guaranteed coverage of the reachable state space of the system under verification.

Figure 4.5 compares the state space coverage-characteristics of incomplete test bench-based non-formal verification to those of complete formal verification.

4.4.3 Formalizing the Analog Verification Flow

With the growing need for formalizing analog verification in industrial design flows and analog formal verification tools not yet being available, approaches that give up formality in favor of delivering practical solutions have emerged. Systematic verification based on verification plans that introduce a hierarchy of behavioral modeling and verification-oriented test benches target the problem in a conventional way by changing the methodology how existing tools are used [CK07]. On block level, formal methodologies could be introduced into industrial flows quickly with the support of EDA-vendors. However, increasing the complexity of the system under verification is continuously decreasing the applicability of formal methods, as presented in [BGG+09]. In the RF-domain, sophisticated behavioral modeling and therewith making possible more simulations within a given verification time is already considered as formalizing the verification in this domain [WJWH09].

As analog circuit simulators are the central tool all verification flows are based on, in [TGP^{+}09] a satisfiability (SAT) solver-based approach to SPICE-type circuit simulation for formal simulation results is presented in order to target the verification problem from a different perspective. By formulating the simulation problem as an input for a SAT-solver, the simulator can discover all possible solutions for a given simulation task. Due to the NP-completeness of the SAT-problem, the approach can exhibit exponential worst-case runtime complexity which is targeted by abstraction refinement.

4.5 Existing Formal Approaches to Analog Verification

In this section, existing formal approaches to analog verification are discussed in order to obtain an overview over the state of the art.

Depending on their scientific origin, the approaches to formal verification of analog circuits have very opposing basic principles. On the one hand, methods that have been developed within the theory of logic reasoning and abstract system theories are extended to the area of analog verification. These approaches feature a consistent logical and formal foundation but they share the problem of being far away from applicability to real world analog circuit verification problems. Often, complicated manual modeling to abstract representations is necessary in order to apply the verification algorithms. Property specification is forcing logic theory upon analog property specification instead of aligning to the semantical terminology of existing analog circuit properties.

On the other hand, approaches coming from the background of electrical engineering and analog EDA-research are very practical and straight forward but often lack formality, soundness and structuredness.

4.5.1 Reachability Analysis and Verification of Analog Circuits

Reachability-based verification approaches use state space exploration techniques in order to formally compute the conservatively approximated reachable area in the state space of a system represented by ODEs. In contrast to the discrete modeling presented in Section 2.4, verification approaches incorporating reachability analysis do not generate a complete model of the state space of the circuit that can be checked for more complex properties. Therefore, compared to model checking, these approaches do not offer a property evaluation methodology exceeding the question whether a specified state is reachable from a given starting state.

Originating from the theory of hybrid systems, first reachability verification approaches targeted simple nonlinear control systems by linearization [HH95]. Reachability analysis development towards nonlinear analog circuit verification introduced calculation of hybrid polyhedral outer approximations enclosing the flow derived from differential equation systems [GKR04] using the tool “Checkmate”. Investigated circuits were a simple tunnel diode oscillator and a Matlab model of a third-order delta-sigma modulator.

Another hybrid system verification approach for analog systems is based on the ”d/dt” tool, applied to different analog circuit examples such as a biquad lowpass filter and the model of a delta-sigma modulator [DDM04, ADG07, DN09]. Specializing on oscillator reachability verification and stability computation without emphasis on a structured specification, approaches were presented in [FKR06, GY08].

4.5.2 Analog Model Checking

The first approach to model checking of analog circuits introduced a discrete state space model of the circuit to be checked against a property specification given as ω-automata. The circuit investigated was a transistor level representation of a digital interlock, described using simplified transistor models implemented by a capacitor and a voltage controlled current source [KM91]. Although the discretization into a finite automaton is very rough, this is the first approach to discrete modeling of the state space of analog systems with application of an automata-based property verification.

Derived from the successful property specification of digital and software systems with the Computation Tree Logic (CTL) [CE82], the first approaches to CTL-based model checking of analog systems applied CTL extended with an analog operator to a discretized state space model of nonlinear analog circuits [HHB02a, HHB02b]. With an adaptive state space discretization that controls the size of the enclosures of state space regions depending on the level of homogeneity of the state space dynamics, as described in Section 2.4.3, properties of nonlinear circuits such as the oscillation of a tunnel diode oscillator, overshooting of a second-order lowpass filter and the switching behavior of a Schmitt-trigger circuit have been verified. The addition of time constraints to the CTL specification allowed to first time formally verify timed behavior of analog circuits [GPHB05].

Weakening the formality of the approach, in [DC05, DC07] a verification system applying an extended CTL derivative called AnaCTL to the transient response of analog circuits is proposed. By higher level modeling of analog circuits using labeled hybrid petri nets, verification of AMS-systems is introduced in [LSW$^+$06, MHW$^+$06], not offering an automated translation of transistor level circuits to the petri net representation. The same limitation applies to the approach presented in [ASZT07], where a stability verification of a symbolic mathematical representation of a delta-sigma modulator using recurrence equations is introduced.

In contrast to the industrial application of model checking of digital hardware systems, the approaches to model checking of analog systems have not yet been introduced into industrial flows. There are several challenges that need to be solved in order to develop analog model checking towards industrial applicability. These challenges will be outlined in the following.

- The continuous system description originating from the differential equation system of the analog BCEs can not be evaluated by direct theorem proving approaches. Therefore, abstractions have to be made that have to balance between model complexity and model accuracy.

- The discretization of an analog system to a state model suffers from the state space explosion problem. Due to the exponential growth of the number of states with the number of state space variables, only block-level analog circuits can be handled.

- The hyperbox discretization, which is part of the up to now most capable model checking approach, is not rotation-invariant and massively over-approximates the transition relation of the system. Hence, it does not deliver reliable results for complex state space dynamics structures.

- Up to now, approaches to model checking of analog circuits are based on property specifications closely related to the digital specification with CTL syntax and semantics. Even for digital specification, approaches like PSL have been necessary to facilitate the access to formal property specification for the designers. Moreover, the semantics of branching time temporal logics used in analog model checking are not expressive enough for the precise specification of complex analog behavior.

- Corresponding to the lack of specification expressiveness of analog CTL specification, the algorithmic evaluation of formal analog specifications needs a different methodology compared to the digital domain. Digital specifications are very time-oriented with abstract properties such as fairness and liveness that are easy to specify and evaluate on digital state models. In contrast, analog properties such as slew rate, oscillation, overshoot, etc. are based on the complex relation of different internal state variables and specifications have to reason on the discrete representations of their continuous magnitudes and their alteration over time.

Some of these challenges will be targeted by the ASL model checking approach presented in Section 5.2.

4.5.3 Analog Equivalence Checking

The basic goal of formal equivalence checking of nonlinear analog circuits is to verify if both systems have the same input-output behavior for every possible input signal, corresponding to Definition 4.3.3. The existing approaches are discussed in the following. Moreover, in order to obtain a benchmark for the new equivalence checking approach proposed in Section 5.6, the up to now only automated formal approach to analog equivalence checking for nonlinear analog circuits that can directly operate on transistor netlist representations and behavioral models in VHDL-AMS is outlined.

An approach to equivalence checking for linear analog circuits with parameter tolerances is proposed in [HB98]. Based on the computation of outer bounds of the transfer functions using complex interval arithmetics, the over-estimated implementation of the circuit is compared with the inner bounds of a therewith underestimated specification function. Hence, sound results are obtained. Another approach dealing with linear circuits described by their frequency domain transfer functions considering parameter variations is presented in [SA01]. The phase and magnitudes over a defined frequency range are equivalence-checked using an envelope for the response comparison. A restriction to linear circuits is however not suited for an universal application to real-world problems.

Based on the PVS theorem prover, in [GV99] the functional equivalence of behavioral descriptions in VHDL-AMS and their synthesized analog circuits is checked by evaluating piecewise linear approximations of the analog behavior in the DC and low frequency domain. Due to the simplifications, this approach is not a complete verification methodology.

A state space sampling based approach for equivalence checking of nonlinear analog circuits was initially introduced in [HB95] with application to different CMOS inverters and extended in [HKH04] for application to a Schmitt-trigger, implemented as transistor netlist and behavioral description, as well as to a bandpass transistor circuit. An extension to improve handling of strongly nonlinear circuits by introducing structural recognition and mapping of eigenvalues to circuit elements and a reachability analysis was presented in [SH10a]. This allowed to additionally apply equivalence checking to the analog behavior of a NAND gate and a mixer circuit.

For both system implementations under verification, using a local linear transformation for each sample point to a canonical representation (Kronecker's canonical form) and by numerically integrating these linear local transformations, an approximation of the nonlinear transformation for the system is obtained. The numerical differences between both internal transformed dynamics and the output variables give a direct measure for the equality/difference of both systems.

While this approach can be successfully applied to behavioral models with an internal structure not too different from the modeled transistor block, strong abstraction of the behavioral model can result in reporting complete inequality. This is due to the need for an internal mapping of state variables which is only possible with certain similarities in the systems' structure.

In the following subsection the VERA equivalence checking method from [HB95, HKH04, SH10a] is outlined, as it is the benchmark the new equivalence checking methodology developed in Section 5.6 will be compared to in Section 6.6.

4.5.3.1 The VERA Equivalence Checking Algorithm

The VERA equivalence checking algorithm focuses on circuit descriptions based on a system of implicit differential algebraic equations (DAEs) as introduced in Section 2.3.2.

In order to verify the equivalence of two circuits A and B represented as a DAE system, the following approach is conducted by the VERA algorithm:

- Sample the state space of both systems iteratively and perform the following transformation on each sample point.
- Transform the dynamics $\mathbf{f}$ in the variable space spanned by $\mathbf{x}$ and $\mathbf{u}$ of each system into a canonical state space spanned by a vector $\mathbf{z}^{(c)}(t)$ of n_c canonical state variables $z_i^{(c)}(t)$ with $1 \leq i \leq n_c$. The transformation is defined by:

$$\begin{aligned} \mathbf{x} &= \mathbf{F}(\mathbf{z}^{(c)}) \\ \mathbf{z}^{(c)} &= \mathbf{F}^{-1}(\mathbf{x}) \end{aligned} \tag{4.6}$$

- The dynamics will then be transformed according to:

$$\mathbf{h}(\mathbf{z}^{(c)}(t), \dot{\mathbf{z}}^{(c)}(t), \mathbf{u}(t)) = \mathbf{f}(\mathbf{F}(\mathbf{z}^{(c)}(t)), \tfrac{\partial \mathbf{F}(\mathbf{z}^{(c)}(t))}{\partial t}, \mathbf{u}(t)) \tag{4.7}$$

  If the transformation is well chosen, the resulting system $\mathbf{h}$ will have only n_z nontrivial dynamic equations describing the system behavior. The remaining algebraic equations should be trivial (e.g. $0 = 0$).

Canonical State Space Comparison After obtaining a canonical state space representation of both systems A and B, the transformed system functions $\mathbf{h}_A$ and $\mathbf{h}_B$ have to be compared in the canonical state space $\mathbf{z}^{(c)}$. This assumes that both circuits A and B are transformed to canonical state space variables $\mathbf{z}_A^{(c)}$ and $\mathbf{z}_B^{(c)}$ with equal size $n_{z_A^{(c)}} = n_{z_B^{(c)}} = n_c$ using dominant pole order reduction for the system with more state space variables. The dynamics of the system functions can be compared by checking the values of the state derivatives $\dot{\mathbf{z}}_A^{(c)}$, $\dot{\mathbf{z}}_B^{(c)}$ to be equal for each state in the state space. This comparison is performed numerically, resulting in an error value with an appropriate norm:

$$\epsilon_{\dot{z}} = \|\dot{\mathbf{z}}_A^{(c)} - \dot{\mathbf{z}}_B^{(c)}\| \tag{4.8}$$

Obviously, this error will never be zero for analog circuits. Therefore, the user has to define an error value limit. If the error is below this limit in the whole reachable state space, the circuits are regarded as equivalent. This error is very sensitive to differences in the circuits under verification. For example, a relative error limit of 10% can be considered as appropriate. Besides the internal dynamics, using a selection matrix **R**, the output variables of the systems

$$\begin{aligned} \mathbf{x}_A^o &= \mathbf{R}_A \cdot \mathbf{F}_A(\mathbf{z}_A^{(c)}) \\ \mathbf{x}_B^o &= \mathbf{R}_B \cdot \mathbf{F}_B(\mathbf{z}_B^{(c)}) \end{aligned} \tag{4.9}$$

can be compared with a similar error measure:

$$\epsilon_y = \|\mathbf{x}_A^o - \mathbf{x}_B^o\| \tag{4.10}$$

Comparing the dynamics of the systems, as well as their output variables, assures that the dynamic and static behavior of the systems under verification have been considered for equivalence checking, resulting in high confidence in the obtained results. The described VERA equivalence checking methodology is summarized in Figure 4.6.

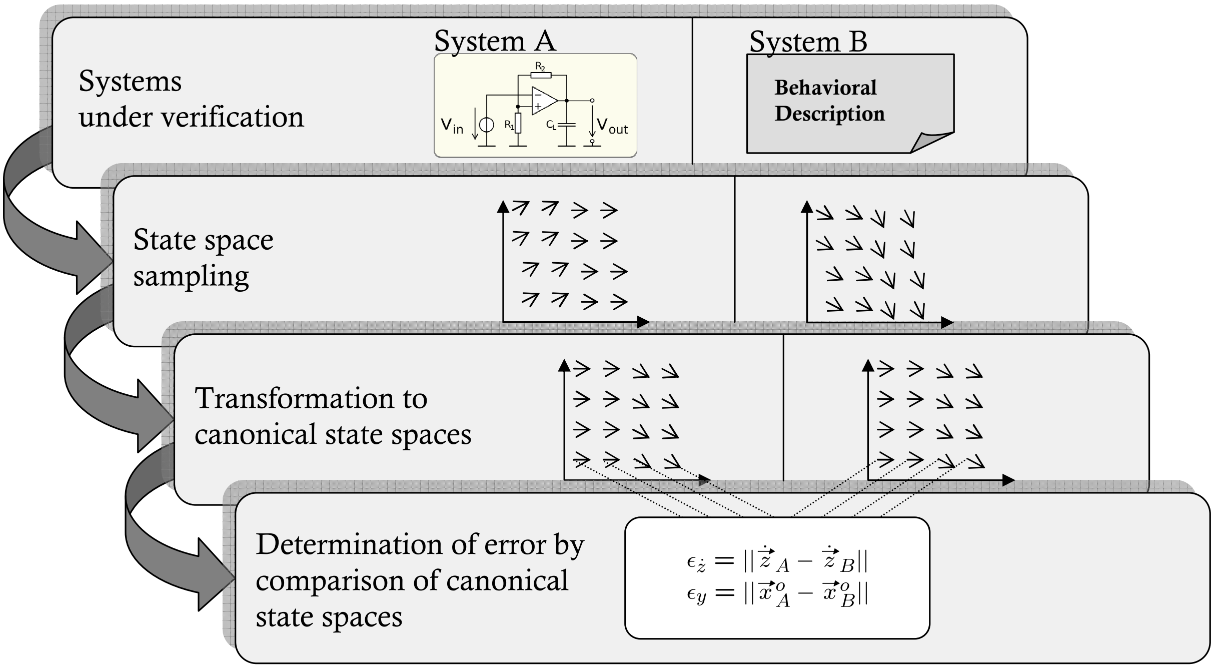

System A
System B
Systems under verification
R2
Vin
R1
CL
Vout
Behavioral Description
State space sampling
Transformation to canonical state spaces
Determination of error by comparison of canonical state spaces
$\epsilon_{\dot{z}} = ||\vec{\dot{z}}_A - \vec{\dot{z}}_B||$
$\epsilon_y = ||\vec{x}^o_A - \vec{x}^o_B||$

Figure 4.6: *Structure of the VERA equivalence checking methodology.*

5

Analog Formal Verification Methodologies

A framework of new analog formal verification methodologies has been developed around the newly introduced specification language ASL and the corresponding verification algorithms. As will be described in this chapter, fundamental formal verification methods such as model checking and equivalence checking are enhanced by the ASL verification algorithms and the improved discrete model generation using the newly introduced trajectory-directed discretization approach. Moreover, new verification concepts such as transient simulation with complete state space-covering input stimuli will be introduced. In combination with the counterexample generation, the application of ASL specifications to conventional transient simulation waveforms and new visualization approaches for the verification results, a set of new verification methodologies has been developed which will be combined into an analog verification framework (AVF).

5.1 New Verification Methodologies for the Analog Design Flow

Figure 5.1 illustrates the possibilities of the new AVF, offering several combinations of different modeling and verification approaches. The possible verification method-

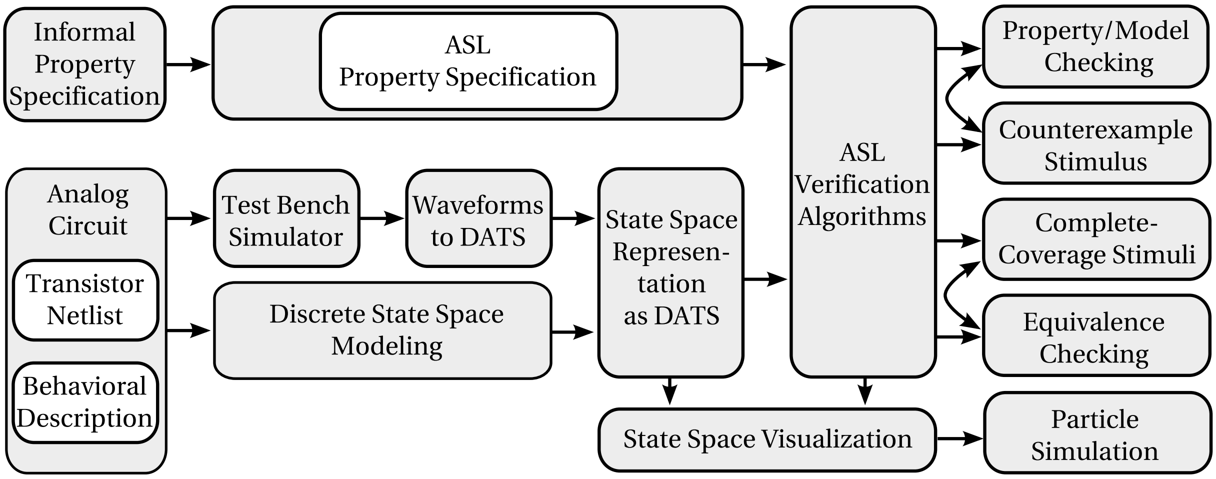

Informal Property Specification
ASL Property Specification
ASL Verification Algorithms
Property/Model Checking
Counterexample Stimulus
Analog Circuit
Transistor Netlist
Behavioral Description
Test Bench Simulator
Waveforms to DATS
State Space Represen- tation as DATS
Complete- Coverage Stimuli
Discrete State Space Modeling
Equivalence Checking
State Space Visualization
Particle Simulation

Figure 5.1: *Analog verification framework for different verification methodologies.*

ologies are outlined in the following and detailed in the subsequent sections of this chapter.

5.1.1 Verification Methodology Perspective

Starting from a behavioral or transistor netlist representation of an analog circuit, a discrete state space model for each circuit under verification is generated using the new approach presented in Section 2.4.4. Alternatively, conventional transient simulation results from test bench-based verification can be transferred into a partial state space representation which then can be processed by the verification algorithms accordingly, as will be described in Section 5.3. To the discrete state space model of the circuit, the following verification methodologies can be applied:

- The state space model can be checked against an ASL property specification which is the method of property checking or model checking (see Section 5.2).
- From identified state space regions that violate the specification, counterexample stimuli can be generated, which then in turn can be used in a test bench simulation environment to analyze the specification-violating behavior (see Section 5.4).
- By a systematic traversal of the DATS model, complete state space-covering piecewise linear input stimuli can be generated (see Section 5.5). Based on these complete-coverage input stimuli (CCIS), three sub-methodologies are available:

- The CCIS can be used in test bench-based simulation environments for manual analysis of the transient response to these stimuli that guide the simulator into every reachable state of the analog circuit.
- The transient simulation response to the CCIS can be re-transferred into a state space representation for evaluation of assertions specified in ASL. This is an alternative model checking approach with the advantage of not introducing a discrete modeling error due to directly operating on the results of the transient simulation, providing high accuracy of measured values (see Section 5.5.1).
- By generating CCIS for two circuits under verification and then comparing the deviation of the transient responses to the stimuli of both circuits using a specific ASL specification, a formal equivalence checking methodology is given due to the transient responses of both circuits being directed to every reachable state (see Section 5.6).

In order to interactively explore the state space dynamics of an analog circuit, an approach to multi-parallel particle simulation will be introduced in Section 5.7 which complements the insight into acquired verification results of the verification examples in Chapter 6.

5.1.2 Design Flow Perspective

Another perspective to application of the new verification methodologies in the analog circuit design flow is presented by considering a top-down design flow consisting of an ASL property specification that is transferred into an abstract behavioral model for the circuit design. Subsequently, a transistor circuit implementation for the circuit blocks is developed. Finally, after layout, an extracted version of the circuit can be considered.

Within this flow, the three major verification concepts can be applied to different design levels as illustrated in Figure 5.2. These concepts are assertion-based verification (ABV) by evaluating ASL specifications on transient simulation waveforms that can either be generated by user-defined input stimuli or CCIS, ASL model checking and equivalence checking with CCIS. A connection between two abstraction levels in the illustration corresponds to the possibility to apply the verification methodology.

ASL evaluation on transient simulation waveforms can be used to automatically check an ASL property specification against every level of abstraction. ASL model checking again can be applied to every level with the limitation that extracted blocks have to be processable by the discrete modeling approach. If too many parasitic capacitances occur, an approach to constrain the state space variables by an eigenvalue-based model order reduction, as discussed in Section 2.4.4.7, has to be applied. Equivalence checking using the methodology of automatically comparing transient simula-

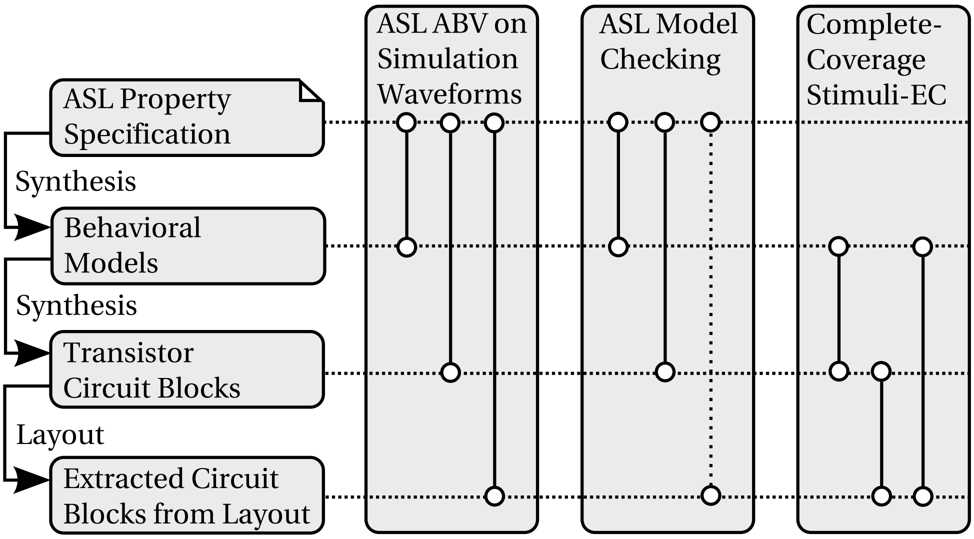

ASL ABV on Simulation Waveforms
ASL Model Checking
Complete-Coverage Stimuli-EC
ASL Property Specification
Synthesis
Behavioral Models
Synthesis
Transistor Circuit Blocks
Layout
Extracted Circuit Blocks from Layout

Figure 5.2: *New verification methodologies in the analog design flow.*

tion waveforms that have been generated using CCIS can be applied to compare any two levels of abstraction. However, at least one has to be modeled as a DATS for stimuli generation and both have to be processable by transient simulation.

5.2 ASL Property Specification and Verification Methodology

With the introduction of the trajectory-directed discretization generating DATS models of analog circuits with higher accuracy in Section 2.4.4 and the definition of ASL in Section 3.5, a property specification methodology for ASL shall be introduced in the following. Therewith, methodologies for circuit overshoot, advanced oscillation properties and for the startup time of autonomous circuits will be described. With these specification methodologies, the verification algorithms can automatically compute the model checking results without user interaction, reporting the acquired results in a verification report which will be demonstrated in Chapter 6.

5.2.1 Specification of Circuit Overshoot

For many types of circuits, the range of the output signal shall be constrained for a defined input range. However, there are circuits such as active filter circuits that tend to overshoot. This means that, depending on the input signal shape and frequency, output values higher than the expected output range can occur. Such a behavior is illustrated in Figure 5.3 where the dotted trajectory starting in a DC-operating-point of

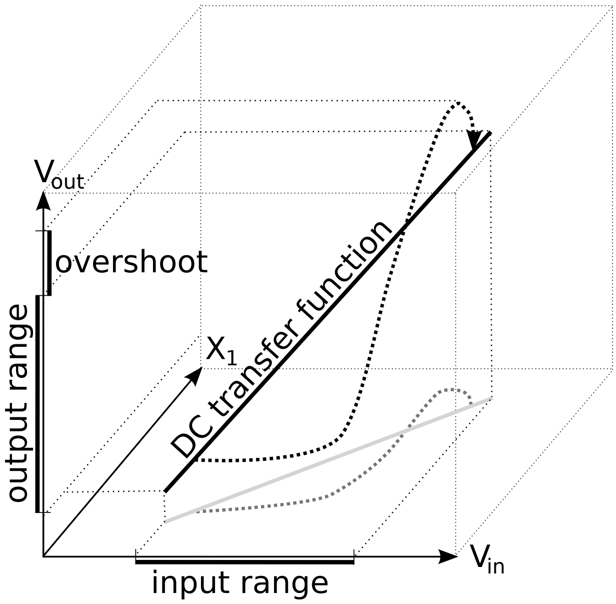


Figure 5.3: *Overshoot of the output voltage caused by a trajectory in the state space.*

the circuit exceeds the output range of the DC transfer function for the defined input voltage range. In the following, an ASL specification methodology for identifying such overshoot in a circuit's state space and measuring the overshoot ratio will be described. All verification steps shall be conducted on the reachable states of the circuit under verification. Hence, the set of states reachable from DC-operating-points is identified:

```
reachable = on all reach from DCpoints;
```

On this set `reachable` of reachable states, the minimum and maximum of the input and output voltages is assigned to number variables:

```
on reachable assign(%min_V_in,min) assign(%max_V_in,max) value(V_in);
on reachable assign(%min_V_out,min) assign(%max_V_out,max) value(V_out);
```

With these data, the overshoot ratio can be calculated by the division of the output voltage range by the input voltage range:

```
%overshoot_ratio = (%max_V_out-%min_V_out)/(%max_V_in-%min_V_in);
```

Finally, an assertion can be formulated that checks that the overshoot ratio stays between user defined specification values `%min_spec_value` and `%max_spec_value`:

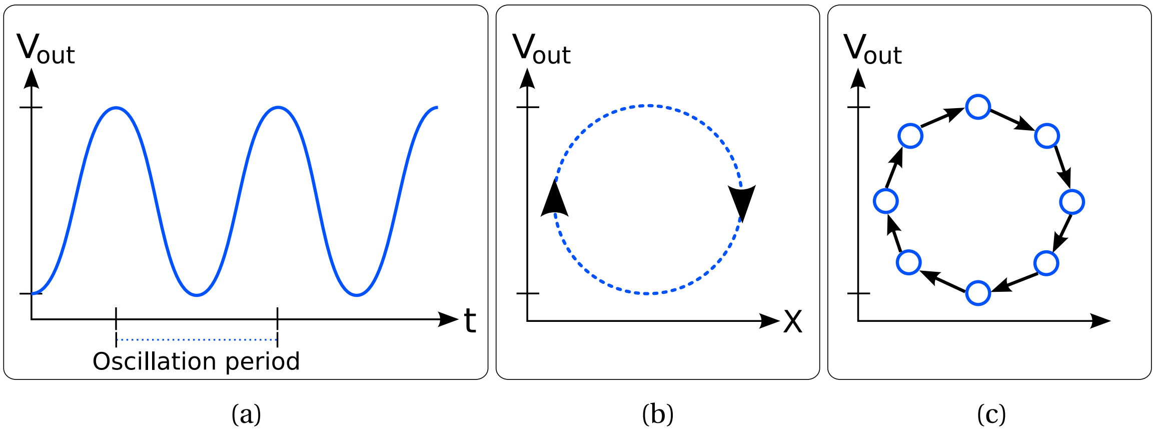
```
for %overshoot_ratio assert [%min_spec_value, %max_spec_value];
```

The result of the assertion as well as the measured voltage ranges are printed into the verification report for easy evaluation of the verification run.

5.2.2 Specification of Oscillation and Voltage Controlled Oscillator Gain K_{VCO}

For oscillator circuits, a specification methodology for verification of oscillation will be introduced in the following. Additionally, for voltage controlled oscillators (VCOs), a specification methodology for verification of the VCO gain K_{VCO} will be presented.

In the time domain, oscillation is represented by a periodic behavior of a circuit variable as shown in Figure 5.4(a). Transferred to the continuous state space, a cyclic path between at least two state space variables can be identified as illustrated in Figure 5.4(b). This results in a set of states connected to a cycle in the DATS as shown in Figure 5.4(c).



(a) (b) (c)

Figure 5.4: *Oscillation in the time domain (a), in the continuous state space (b) and in the DATS (c).*

The simple check whether there is an oscillation within a defined oscillation period range between `%spec_min` and `%spec_max` in the considered circuit model can be formulated in ASL as follows:

```
assign(%oscillation_period_min, min) oscillation;
assign(%oscillation_period_max, max) oscillation;

for %oscillation_period_min assert [>= %spec_min];
for %oscillation_period_max assert [<= %spec_max];
```

Now, a systematic methodology for verifying the oscillator gain K_{VCO}, represented by the sensitivity relation between the control input voltage and the oscillation frequency of a VCO, is introduced. Thus, the input voltage has to be constrained to different values and at each of these input voltages the oscillation frequency is determined as

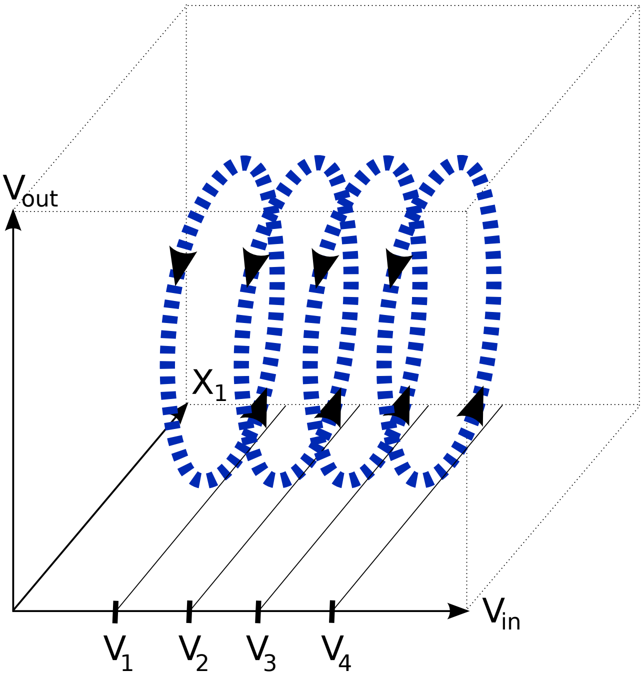


Figure 5.5: *Schematic illustration of oscillations for different input voltages* V_1*,* V_2*,* V_3 *and* V_4*.*

illustrated in Figure 5.5. Comparing each two oscillation frequencies f_{osc} of consecutive input values to their input voltage difference, the sensitivity

$$K_{VCO} = \frac{\partial f_{osc}}{\partial V_{in}} \tag{5.1}$$

can be determined. In the following methodology, only two different input values are considered for the purpose of clarity. Considering more than two input values, the deviation between the calculated local factors K_{VCO} provides information about the linearity of the VCO.

At first, the constrained input voltage areas in the state space have to be assigned to set variables as follows:

```
inp_set_1 = value(V_in)
  [%inp_voltage_1 - %range_size/2, %inp_voltage_1 + %range_size/2];

%inp_voltage_2 = %inp_voltage_1 + %inp_step;

inp_set_2 = value(V_in)
  [%inp_voltage_2 - %range_size/2, %inp_voltage_2 + %range_size/2];
```

On the selected state space slices, oscillation periods are determined. Although the average oscillation period for a given input voltage is considered, this approach is valid for the minimum or maximum oscillation period just as well:

```
osci_set_1 = on inp_set_1 assign(%osci_period_1, average) oscillation;

osci_set_2 = on inp_set_2 assign(%osci_period_2, average) oscillation;
```

Subsequently, the factor of change of the oscillation frequencies and the input voltages are determined:

```
%frequency_delta = (1/%osci_period_2) - (1/%osci_period_1);

%input_delta = %inp_voltage_2 - %inp_voltage_1;
```

In the final step, for the VCO gain property `%K_VCO`, calculated according to Equation 5.1, an assertion is specified. The gain is asserted to be within a percental range specified by the number variable `%tolerance` around the specified value `%K_VCO_spec`:

```
%K_VCO = %frequency_delta / %input_delta;

for %K_VCO assert [%K_VCO_spec - %tolerance/2, %K_VCO_spec + %tolerance/2];
```

5.2.3 Specification of the Startup Time of Autonomous Circuits

The startup time of a circuit is defined as the time from applying the supply voltage until the circuit reaches a desired output behavior, which is in the following considered equivalent to reaching a certain output voltage range.

In the state space, the maximum startup time is defined by the longest transition time among all possible paths leading from the initial state area to the destination area. Figure 5.6 illustrates the startup transition paths in a state space.

For a systematic specification of the startup time property in ASL, we define the initial conditions of the circuit. Hence, the corresponding start area in state space is selected by constraining the considered state space variables `V_out`, `X1` and `X2`. This resulting start area is assigned to the set variable `startarea`:

```
startarea = value(V_out)[< 0.1] and value(X1)[< 0.1] and value(X2)[< 0.1];
```

The area reachable from the start area represents all possible system states for the given initial condition. The states reachable from the set `startarea` are assigned to the set variable `reachable`:

```
reachable = reach from startarea;
```

In practical applications, the minimum required output voltage for the destination area could be directly assigned with a specification parameter. For a worst-case analysis of the startup time, 90% of the maximum output voltage within the reachable state space area are assigned to the number variable `%max_voltage` instead. This is considered as the worst-case upper bound for the steady state output behavior:

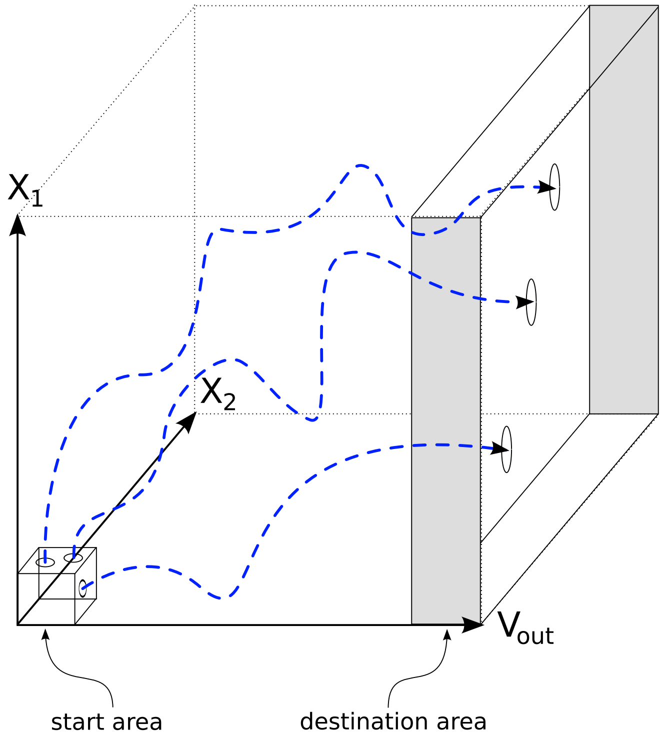


Figure 5.6: *Schematic illustration of startup transition paths.*

```
on reachable assign(%max_voltage, max) 0.9*value(V_out);
```

By calculating the maximum time needed for the transition from the start area to the area with the maximum output voltage, an upper bound for the startup time of the examined circuit is acquired:

```
assign(%startup_time, max) transition from startarea
  to value(V_out)[>= %max_voltage];
```

The final step asserts that the required output voltage is reached and that the startup time for reaching this area never exceeds the specified upper bound:

```
for %max_voltage assert [>= %spec_parameter1];
for %startup_time assert [<= %spec_parameter2];
```

5.3 ASL Property Verification on Transient Simulation Waveforms

In order to obtain a wider field of application for ASL property specification and evaluation, and to develop another formal property verification methodology in Section 5.5, evaluation of ASL property specifications shall be extended from discrete state space models to analog transient simulation waveforms. Therefore, the simulation waveforms have to be transferred into a state space representation as DATS to be introduced into the ASL verification toolchain.

Transient simulation data consists of data tuples containing a sequence of signal values and their corresponding time points for every investigated node voltage or branch current of the circuit under verification. A transient waveform for such a single signal s_i is a sequence $s_i(t_0), s_i(t_1), ..., s_i(t_n)$.

Definition 5.3.1 (Path π_{tr} in State Space generated from Transient Simulation Waveforms)
From a set of transient simulation waveforms for different signals, the sequence of states is a path π_{tr} in the DATS state space model determined by the vector of the m signal values $s_i(t_j)$ with $1 \leq i \leq m$ for each time point t_j:

$$\pi_{tr} = \sigma_{t_0}, \sigma_{t_1}, ..., \sigma_{t_n} \text{ with } L_V(\sigma_{t_j}) = \begin{bmatrix} s_1(t_j) \\ s_2(t_j) \\ \vdots \\ s_m(t_j) \end{bmatrix} \text{ for all } 0 \leq j \leq n \tag{5.2}$$

The transition relation is connecting consecutive states and the transition times are determined by the time steps of the transient simulation waveforms:

$$T(R(\sigma_{t_j}, \sigma_{t_{j+1}})) = t_{j+1} - t_j \tag{5.3}$$

The generation of a path π_{tr} in the DATS is illustrated in Figure 5.7. Due to the transition times now being defined by the labels of the graph transitions, a time axis is obsolete but plotted for better understanding.

As illustrated in Figures 5.8(a) and 5.8(b) for two periodic signals, their combined state space representation in Figure 5.8(c) forms a circle by mapping both signals to a plot over axis $s_1(t)$ and $s_2(t)$. For the state space representation of transient signals, a detection of periodic behavior is necessary for creating closed cycles for oscillation detection. With a defined tolerance interval ϵ, for each vertex σ_{t_r} it is checked whether its coordinate vector $\mathbf{p}^{(t_r)} = L_V(\sigma_{t_r})$ maps to the coordinate vector of another vertex $\mathbf{p}^{(t_s)} = L_V(\sigma_{t_s})$:

$$\sigma_{t_r} \equiv \sigma_{t_s} \Leftrightarrow \forall\, 1 \leq i \leq m : |p_i^{(t_r)} - p_i^{(t_s)}| < \epsilon \tag{5.4}$$

In this case, the transitions from predecessors and to successors of σ_{t_s} are connected to σ_{t_r}:

$$R = R \cup \{(\sigma_{t_{s-1}}, \sigma_{t_r}), (\sigma_{t_r}, \sigma_{t_{s+1}})\} \tag{5.5}$$

Finally, the transitions connecting σ_{t_s} are removed as well as σ_{t_s}:

$$R = R \setminus \{(\sigma_{t_{s-1}}, \sigma_{t_s}), (\sigma_{t_s}, \sigma_{t_{s+1}})\} \quad \wedge \quad \Sigma = \Sigma \setminus \sigma_{t_s} \tag{5.6}$$

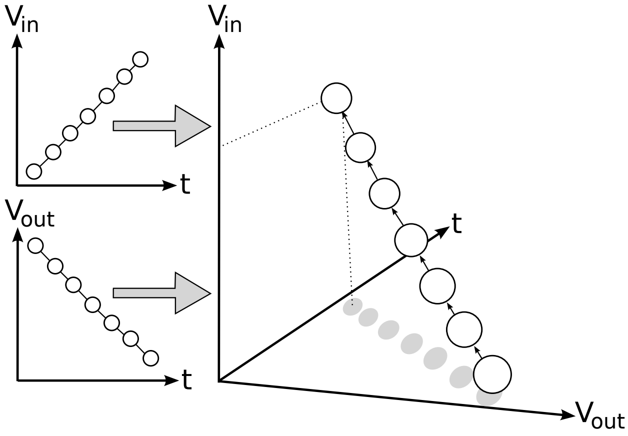


Figure 5.7: *Graph structure obtained by transient simulation waveforms.*

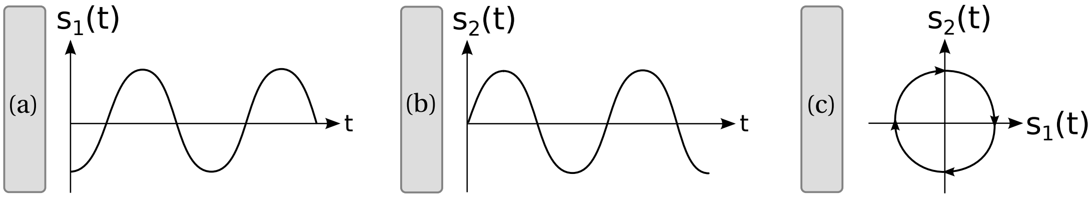


Figure 5.8: *Periodic transient signal waveforms $s_1(t)$ (a) and $s_2(t)$ (b). State space representation of periodic signals $s_1(t)$ and $s_2(t)$ (c).*

Figure 5.9 illustrates the ASL assertion-based verification flow for use with transient simulation waveforms.

As will be demonstrated by application to practical examples in Chapter 6, a joint ASL property specification can be used for automated transient signal evaluation as well as for a formal verification of the circuit's properties, sharing the same verification algorithms. For analog designers, this offers the possibility to get used to formalized property specification without changing their familiar design environment, increasing understanding and acceptance of a future application of formal verification.

5.4 Counterexample Generation for Model Checking

The specification of analog circuit properties using the ASL methodology introduced in Section 5.2 allows to identify regions in the state space of an analog circuit that violate the specification. In order to understand how the circuit behavior reaches such a region, a counterexample can be generated corresponding to the general concept of counterexample paths in model checking as defined in Section 4.3.1.

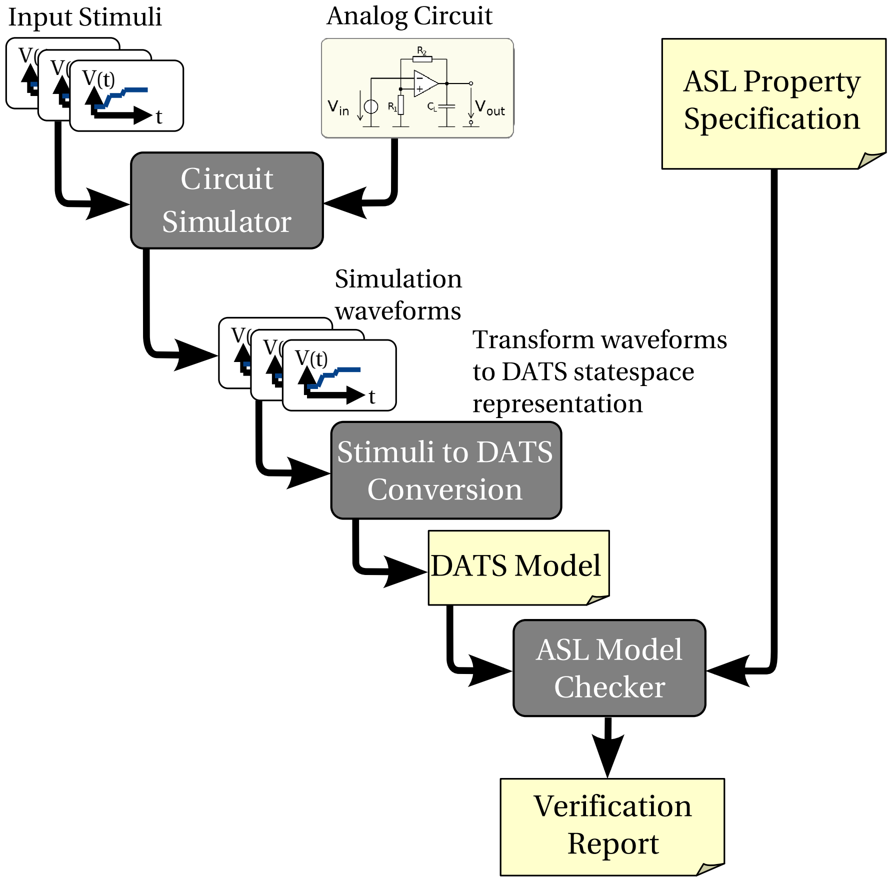


Figure 5.9: *ASL assertion-based verification flow for transient simulation results transferred to a state space representation.*

Definition 5.4.1 (Counterexample on a DATS)
A counterexample for the analog circuit model represented as DATS is a path π_{ce} from a defined starting state σ_0 to the set of states ϕ violating the specification:

$$\pi_{ce} = \sigma_0, ..., \sigma_i : \exists\, i \geq 0 \text{ with } \sigma_i \in \phi \tag{5.7}$$

On such paths, for every extended state space variable, the values of this variable and the corresponding transition times can be recorded for every state transition on the path due to the structure of the DATS. This yields a piecewise linear signal trace over time which can then be visually inspected by the verification engineer. Moreover, by generating such a signal trace for every input variable of the circuit, piecewise linear input stimuli are obtained which can be directly simulated in a circuit test bench.

The starting state for the counterexample input stimulus should be an initial condition that can be easily implemented in the simulation test bench, which for most circuit applications is provided by DC-operating-points. Given that the verification algorithms already checked the reachability of the states violating the specification from

DC-operating-points, any transition path starting in a DC-operating-point and ending in the destination set can be used for determining the piecewise linear signal trace. In the DATS, more than one path between the starting state and the destination set can exist. However, as the runtime of transient simulation increases with the time length of the input stimulus to simulate as the counterexample, an efficient counterexample generation algorithm operating on the DATS should report the path with the shortest overall path time. This path time is the sum of the individual transition times of the transitions visited on the path.

The shortest path in the DATS can be determined by Dijkstra's single source shortest path algorithm [Dij59] which directly outputs the shortest path from the starting state to every other reachable state of the graph structure. Hence, for every state of the set of specification violating states, a counterexample stimulus is generated with an overall worst-case runtime complexity of $\mathcal{O}(n^2)$ where n is the number of vertices in the DATS. Algorithm 5 denotes the method of counterexample waveform generation.

Algorithm 5: Counterexample Generation Algorithm.

```
Input: DATS modeling the analog circuit,
vertex $\sigma_i$ representing a DC-operating-point,
set $\phi$ of vertices identified as violating the specification
Output: List of tuples (value, time) representing piecewise linear waveforms for
        every input and state space variable of the circuit under verification
foreach vertex $\sigma_j$ in $\phi$ reachable from $\sigma_i$ do
    calculate shortest path $\sigma_i \rightarrow \sigma_j$ using Dijkstra's algorithm;
    foreach vertex $\sigma_k$ visited on path $\sigma_i \rightarrow \sigma_j$ do
        store value vector $L_V(\sigma_k)$ and accumulated transition path time;
    end
end
```

The introduced counterexample generation algorithm on the one hand directly allows to visually inspect the signal traces leading from DC-operating-points to states violating the specification. On the other hand, by using the piecewise linear waveforms for the input variables as input stimuli in a transient simulation test bench, the results from model checking on an analog circuit modeled as DATS can be transferred back into a conventional transient simulation-based verification flow. Therewith, the confidence in the obtained formal verification results can be increased as they can be examined in the framework the verification engineer is used to.

5.4.1 Counterexample Generation in the ASL Verification Flow

In the ASL verification flow, the operation `counterexample` is used to generate a set of piecewise linear signal traces. In the following example, the voltage over a capacitor C_1 shall be limited to 1 V. Hence, if there is an overshoot of the voltage, the input stimulus shall be examined that leads to this set `bad_states` of states where the voltage is above the specified value:

```
bad_states = select value(V_C1)[>1];
```

With this set, the counterexamples can now be generated by calling the operation `counterexample` with the set `DCpoints` of DC-operating-points as starting set and the set `bad_states` as destination set for the algorithm:

```
counterexample from DCpoints to bad_states;
```

5.5 Complete-Coverage Input Stimuli Generation

As has been discussed in previous sections, model checking proves the absence of faults in every possible state of a system, regardless of the input conditions. However, the need for a complete different way of thinking when dealing with model checking and formal property specification is a challenge that can only be overcome step by step. While the introduction of designer-oriented methodologies for model checking using ASL can facilitate the access to formal methods, gaining acceptance will be an incremental process.

Consequently, there is a need for formal approaches that seamlessly integrate in existent simulation-based design flows. For this purpose, a novel algorithm for formal automatic input stimuli generation will be proposed in this section. It is combining a formal approach and conventional transient circuit simulation into a verification methodology that overcomes the incompleteness of experiment-based transient simulation and the expected difficulties of analog designers to adapt to model checking approaches.

Derived from the counterexample generation approach introduced in the previous section, input stimuli covering the complete reachable area of the state space of the analog circuit can be computed by traversing the graph modeling an analog circuit as a DATS. Basically, the idea is to visit every reachable state of the graph structure and recording the input values and accumulated times of traveled edges during graph traversal. This concept corresponds to efficiently generating a piecewise linear counterexample input stimulus for every reachable state of the DATS model.

Definition 5.5.1 (Complete State Space-Covering Input Stimulus on a DATS)
A complete state space-covering input stimulus for the analog circuit model repre-

sented as DATS is a path π_{is} from a defined starting state σ_0 that visits every state of the set of reachable states ϕ of the DATS. The set Π_{is} consists of all π_{is} satisfying this requirement.

$$\Pi_{is} = \{\pi_{is} \,|\, \exists\, n : \pi_{is} = \sigma_0, ..., \sigma_n \wedge \bigcup_{0 \leq i \leq n} \sigma_i = \phi\} \tag{5.8}$$

$$\pi_{is} \in \Pi_{is} \tag{5.9}$$

Moreover, depending on the verification methodology where the complete state space-covering input stimuli will be applied, it can be necessary not only to cover every state of the DATS but also to cover every transition of the DATS. This corresponds to the idea not only conducting a subsequent transient simulation that brings the system under verification into every reachable state but also allowing to simulate every possibility how every state can be reached. In the following, the algorithm assumes the goal of complete state and dynamic transition coverage. In order to do this more efficiently than just generating a sequence of counterexamples, an algorithm is needed which satisfies the following requirements:

- When the algorithm terminates, every reachable state and dynamic transition of the circuit model, represented by the vertices and edges of the graph, must have been visited at least once.
- The number of travelled edges on the paths covering the complete state space shall be minimized as each timed transition taken between two vertices of the graph results in an increment of the time length of the input stimulus and is therefore affecting simulation time.
- During stimulus generation, if available, vertices representing DC-operating-points shall be visited periodically. This ensures that the circuit can recover from the traversal of corners of its dynamic behavior by starting and ending in its steady states.

Combining the above requirements into an algorithm reveals the NP-hardness of the optimization problem as it is a modification of the traveling salesperson problem [GJ79]. Accordingly, a heuristic approach is necessary for efficient computation. Due to the fact that any path that covers all reachable edges and states of the graph is a valid solution, an efficient algorithm using a heuristic approach will produce a valid solution with an assumed suboptimal path length. Algorithm 6 shows a possible efficient approach and is described in the following.

For a given DC-operating-point σ_i, the algorithm computes a list of tuples (value, time) representing piecewise linear stimuli for every input variable covering the complete state space. Initialization of variables includes setting the set *open* to all reachable edges and initializing the set *closed* as empty in lines 1 and 2. Starting in line 3, for each

Algorithm 6: Complete-Coverage Input Stimuli Generation Algorithm.

```
   Input: DATS modeling the analog circuit,
   vertex $\sigma_i$ representing a DC-operating-point
   Output: List of tuples (value, time) representing piecewise linear stimuli for
           every input variable covering the complete state space
1  open = all reachable dynamic edges;
2  closed = $\emptyset$;
3  foreach edge $j$ reachable from $\sigma_i$ in open do
4      calculate path covering as many edges as possible from $\sigma_i \rightarrow j \rightarrow \sigma_i$ avoiding
       edges in closed;
5      foreach edge $k$ visited on path $\sigma_i \rightarrow j \rightarrow \sigma_i$ do
6          put $k$ to closed;
7          remove $k$ from open;
8          store value vector $L_V(\sigma_l)$ of visited vertices $\sigma_l$ and path time;
9      end
10 end
```

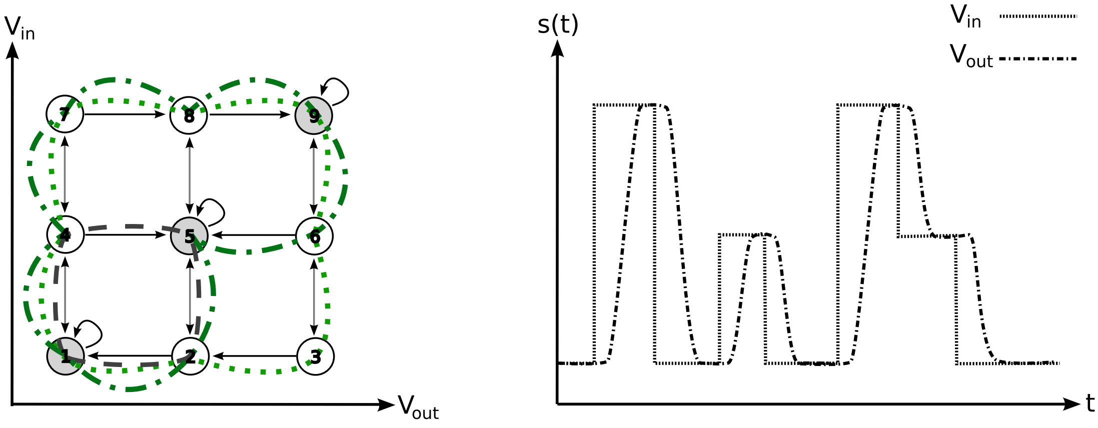
edge j in the set open, a path covering as many edges as possible from $\sigma_i \rightarrow j \rightarrow \sigma_i$ is computed avoiding edges in the set *closed* in line 4. Line 5 iterates over each edge k visited on the computed path $\sigma_i \rightarrow j \rightarrow \sigma_i$ and puts k to the set *closed* in line 6, removes it from set *open* in line 7 and stores the parameter tuple creating the piecewise linear stimulus to a file in line 8.

The implicitly mentioned path finding algorithm in line 4 can be any form of a modified longest path finding algorithm such as Dijkstra's applied with negative edge weights for longest path detection. Longest path detection is possible efficiently by modifying the path detection algorithm not to travel loops more than once and hence considering the DATS as an acyclic directed tree graph with the start vertex being added as a leaf vertex for closing single loop runs. For obtaining the longest path with respect to the number of vertices visited, edge weights have to be set to -1. Other optimization criteria are possible, such as considering euclidean vertex distance or the original edge weights containing transition time values. At first, computing the shortest time path might seem like an obvious solution, but as the goal of the stimuli generation algorithm is complete edge and vertex coverage, the number of inevitably revisited edges during single closed loop runs has shown to be high, resulting in worse overall stimuli time length.

In order to avoid revisiting edges, all edges in the set *closed* have to be assigned with a positive value. Trying to minimize the sum of edge weights, the path finding algorithm will automatically avoid those edges that are contrary to the optimization criterion.

The asymptotic runtime complexity of the stimuli generation algorithm for a graph with n reachable edges is dominated by the loop over each edge in the set *open* and the call to Dijkstra's algorithm having quadratic complexity inside this loop. This results in an asymptotic worst-case complexity of $\mathcal{O}(n^3)$.

Figure 5.10(a) shows a possible traversal result generated by the stimuli generation algorithm applied to the graph from Figure 2.6, starting from vertex 1. The first loop $1 \to 4 \to 7 \to 8 \to 9 \to 6 \to 3 \to 2 \to 1$ is created due to vertex 9 being the most distant vertex from 1, thus covering the most edges of the graph. Vertex 5 is still unvisited, therefore a second loop run is needed, traveling vertices $1 \to 4 \to 5 \to 2 \to 1$. In order to cover the last uncovered dynamic transition between vertices 6 and 5, a third run visits vertices $1 \to 4 \to 7 \to 8 \to 9 \to 6 \to 5 \to 2 \to 1$. The input-output behavior representing the stimulus and a possible transient response is illustrated in Figure 5.10(b). While any traversal policy covering the complete graph is valid, further investigation of better strategies is necessary as they directly result in shorter simulation times. The input stimuli generation and verification flow based on discrete state space modeling is illustrated in Figure 5.11.



(a) (b)

Figure 5.10: *Path generated by the stimuli generation algorithm (a) and the corresponding input/output behavior (b).*

5.5.1 Verification Methodology

The experience of an analog circuit designer is not only necessary for developing the circuit itself but for selecting the right test bench around the circuit for ensuring correct future circuit behavior under all expected circumstances.

As there are no written rules for the selection of appropriate input stimuli for transient analysis, a DUV matching the specification might just have not been simulated

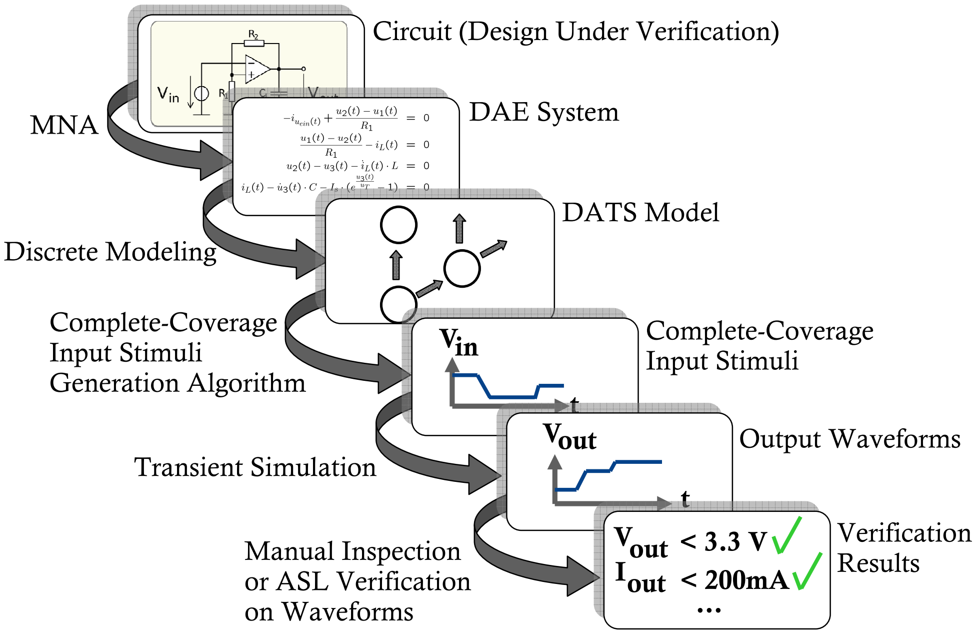

Circuit (Design Under Verification)
MNA
DAE System
Discrete Modeling
DATS Model
Complete-Coverage Input Stimuli Generation Algorithm
Complete-Coverage Input Stimuli
V_{in}
t
Transient Simulation
V_{out}
t
Output Waveforms
Manual Inspection or ASL Verification on Waveforms
V_{out} < 3.3 V ✓
I_{out} < 200mA ✓
...
Verification Results

Figure 5.11: *Complete-coverage input stimuli generation and verification flow.*

with the critical stimuli that would have taken it to violate the specification. Without having any knowledge about state space coverage of the selected input stimuli, the designer is searching for errors but can not be sure about how many are remaining undiscovered.

With the input stimuli generation algorithm introduced in this section, a verification methodology for a simulation-based design flow with guaranteed coverage of all corner cases is given. The generated stimuli can now be processed by an analog circuit simulator, computing an output response for every internal state of the system due to the special structure of the stimuli. With the system dynamics determining the structure of the discretized state space, the stimuli are inherently covering the frequency range of the system behavior at least up to the dynamics in the calculated state space. A user-specifiable maximum edge steepness of the stimuli allows for lowering the simulation effort caused by steep voltage steps. As assertion-based simulation methods are emerging in the analog domain, the complete state space-covering stimuli contribute to strengthen the significance of those approaches.

With the results from transient simulation using complete state space-covering input stimuli combined with the ASL property evaluation on transient simulation waveforms that was presented in Section 5.3, an alternative complete and therewith formal property verification methodology is given. The simulation results obtained from simulation with complete state space-covering input stimuli are representing the complete dynamic behavior of the DUV. Hence, any ASL-specified assertion that holds during an evaluation on such waveforms is guaranteed to hold for any input condition and any state the DUV can adopt. Therewith, a proof of correctness can be conducted which makes the presented approach equally expressive as the ASL model checking on the discrete state space model.

ASL verification on results obtained from complete state space-covering simulation has the overhead of stimuli generation and transient simulation compared to ASL model checking applied directly to the discrete state space model. However, the advantage of this approach is given by being an add-on to the conventional test bench-based simulation approach instead of being a replacement. Moreover, this stimuli-based ASL verification methodology is modular and it can be introduced incrementally into an existing design flow.

5.6 Equivalence Checking using Complete-Coverage Input Stimuli

The simulation of a single circuit using complete state space-covering input stimuli can reveal corner case behavior not identified by user-defined input stimuli. However, an equivalence checking methodology for analog circuits based on the new stimuli generation approach can be developed, giving certainty about the level of equality between two circuit implementations. This new approach for equivalence checking of analog circuits will retain formal completeness but, in contrast to previous approaches, will work well with any kind of circuit abstraction. Due to the application of conventional transient circuit simulation, each step of the equivalence checking process will be observable by the circuit designer and is mostly based on tools he already is used to.

The idea is to generate a stimulus for the system that covers the system's complete state space during a transient simulation. If another circuit is simulated using the same input stimulus, the level of equivalence of the two systems is determined by the level of deviation of the transient responses of the two circuits.

For each of the two circuits to compare, in the following referred to as circuit A and circuit B, complete state space-covering input stimuli are generated for every input of the circuit. Subsequently, four simulation runs are needed. Circuit A is simulated with stimuli of A and B, followed by simulating circuit B with stimuli of A and B. The simulation results are automatically compared using an error measure as described in Section 5.6.1, reporting equivalence if a user-specifiable maximum error value is not exceeded.

If circuits A and B show equivalent behavior when simulated with stimuli generated from circuit A, then the complete behavior of circuit A is included in circuit B:

$$A \subseteq B \tag{5.10}$$

If circuit A and B show equivalent behavior for simulation with stimuli generated from circuit B, then the complete behavior of circuit B is included in circuit A:

$$B \subseteq A \tag{5.11}$$

If both conditions (5.10) and (5.11) hold, circuit A and circuit B are considered as equivalent with respect to the user-defined maximum error, corresponding to the analog equivalence defined in Definition 4.3.3:

$$A \subseteq B \;\wedge\; B \subseteq A \implies A \equiv B \tag{5.12}$$

Figure 5.12 illustrates the described equivalence checking methodology.

In practical applications, often only one direction of the proof is necessary. Especially for reduced models generated with model order reduction techniques, the complete-coverage stimuli have to be generated only for the reduced model in order to prove that the transistor netlist behaves equal for the limited state space of the model during simulation. Of course, the other direction of the proof could fail as the reduced model intentionally does not cover all aspects of the transistor netlist, such as behavior above or below certain operating frequencies.

5.6.1 Error Measures for Waveform Comparison

For a complete automation of the equivalence checking flow, the differences between the simulation results of the two circuits under verification have to be computed by an error calculation algorithm. While there are several measures to calculate the error between two waveforms like Frechet distance [HA92], modified Hausdorff distance [PHHB98], etc., the most intuitive measure is the generation of a difference waveform of the signals. Therefore, for each time point of the two waveforms A and B, the value at this time point of the other waveform is calculated and the difference value is stored. The maximum difference between the waveforms is the reported error value and the results can be inspected by plotting the difference waveform. The difference error measure $\epsilon_{\text{difference}}$ is defined as:

$$\epsilon_{\text{difference}} = \max\left\{\frac{s_A(t_i) - s_B(t_i)}{r}\right\} \text{ for all } (s_A(t_i), s_B(t_i))$$

with:

r	-	maximum signal value range
$s_A(t_i), s_B(t_i)$	-	values at time point t_i for each waveform

The error can be normalized with the range of all values to obtain a deviation between 0 and 1. As the time points of two simulation waveforms generated by different simulation runs in general are not equal, an interpolation is necessary for obtaining the error values of the corresponding waveform at an arbitrary time point.

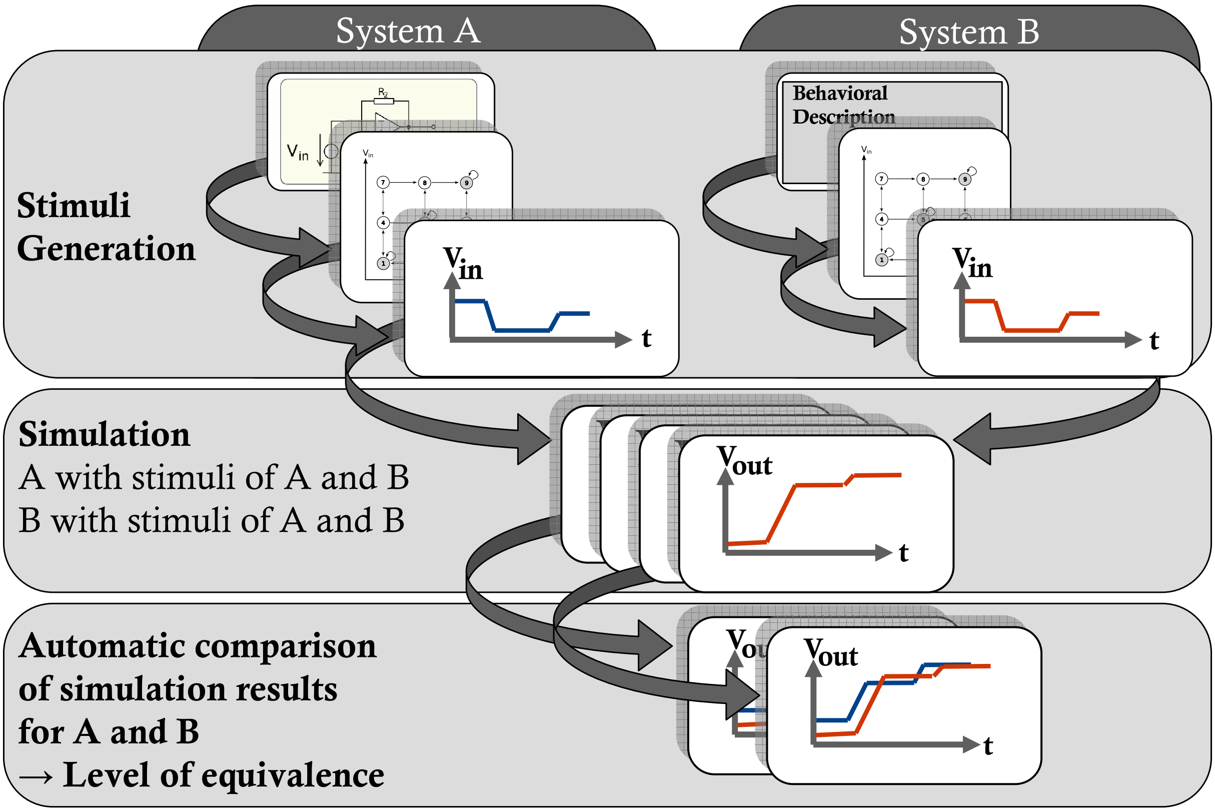

System A
System B
Stimuli Generation
Behavioral Description
Vin
t
Simulation
A with stimuli of A and B
B with stimuli of A and B
Vout
t
Automatic comparison of simulation results for A and B
→ Level of equivalence
Vout
t

Figure 5.12: *Equivalence checking flow using complete state space-covering input stimuli.*

5.6.2 Automation in the ASL Verification Flow

In order to include this new equivalence checking methodology completely into the automated ASL verification flow, the waveform comparison can be performed by the ASL verification algorithms, controllable from an ASL specification. Therefore, the simulation waveforms of both circuits under comparison are transferred into a state space representation as described in Section 5.3. Now, ASL can be used for an automated error determination with respect to an user-specifiable error bound. A calculation template is defined which corresponds to the desired error measure of the absolute difference error between the values of both waveforms at every time point:

```
calculation error_calc("abs(calc_par1-calc_par2)");
```

The maximum error is determined by applying this calculation template to Waveform_1 and Waveform_2 and storing the maximum value of the calculation to the number variable `%max_error`:

```
assign(%max_error, max) error_calc(Waveform_1, Waveform_2);
```

Finally, an ASL assertion determines whether the value assigned to `%max_error` is below a specified maximum error `%spec_error_max`:

```
for %max_error assert [<= %spec_error_max];
```

5.7 Multi-Parallel State Space Particle Simulation

Another perspective to the verification methodologies presented in the previous sections which especially well suits periodic circuit analysis can be obtained by visual inspection of the state space dynamics. A visualization of the dynamics of a n-dimensional vector field representing the analog circuit's behavior has not yet been considered as a possible approach to support circuit verification. However, this can be another option to investigate the complete state space dynamics due to complete coverage of the state space.

As dynamic structures can be difficult to identify in such high-dimensional vector fields originating from state space representation of analog circuits, the application of visual aids is mandatory. Approaches such as line integral convolution (LIC) [And04] or anaglyph stereo vision [McA93] facilitate the understanding of 2-D and 3-D visualization but are not covering dynamic transient behavior. Therefore, a novel methodology to consider analog circuits in a state space representation using visualized multi-parallel vector field particle simulation will be introduced in the following.

The dynamic behavior of a system under verification can be analyzed by visualizing the vector field of the system variable's derivatives in the state space. Such a vector field visualization is only possible with a restriction to three dimensions, while state space dimensions of common analog circuits can vary between two and more than four. Therefore, a selection of the main dimensions has to be made to project to the three-dimensional view.

Particle simulation is a common approach for vector field visualization and mature algorithms have been developed [DH96]. In contrast to a static approach such as line integral convolution, time-dependent motion of the particles visualizes the transient behavior of the circuit in an animation sequence.

Consider the vector field $\mathcal{V} : \mathbb{R}^{n_d} \rightarrow \mathbb{R}^{n_d}$ that was defined in Section 2.4.2 on which for the discrete set $Q = \{\mathbf{q}_1, ..., \mathbf{q}_m\}$ of m sample points $\mathbf{q}_i$ in the state space, the discrete vector field $\mathcal{V}_D : \mathbf{q}_i \rightarrow \mathbf{v}_i$ is defined:

$$\mathcal{V}_D(\mathbf{q}_i) = \mathcal{V}(\mathbf{z}^{(e)} = \mathbf{q}_i) = \mathbf{v}_i \tag{5.13}$$

In other words, the discrete vector field $\mathcal{V}_D$ is represented by position vectors $\mathbf{q}_i$ determining the sample points in the state space and the direction vectors $\mathbf{v}_i = \mathcal{V}_D(\mathbf{q}_i)$ giving the motion direction and speed within $\mathcal{V}$ at position $\mathbf{q}_i$.

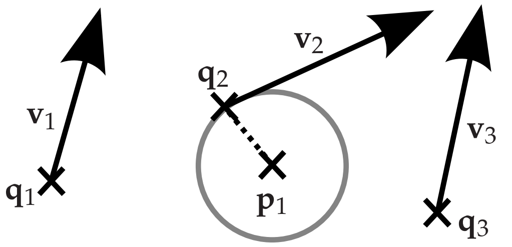


Figure 5.13: *Determining nearest sample point* $\mathbf{q}_2$ *in state space for particle* $\mathbf{p}_1$ *within discrete vector field* $\mathcal{V}_D$.

For the injected particles $\mathbf{p}_i \in \mathbb{R}^n$, their direction vector $\mathcal{V}(\mathbf{z}^{(e)} = \mathbf{p}_i)$ has to be approximated with respect to the discrete vector field $\mathcal{V}_D$. Hence, a mapping is necessary which assigns a nearest sample point $\mathbf{q}_j \in Q$ to each point $\mathbf{p}_i$ representing a particle from the set of particles, as illustrated in Figure 5.13:

$$\mathcal{M}(\mathbf{p}_i) = \{\underset{\mathbf{q}_j \in Q}{\arg\min} \|\mathbf{p}_i - \mathbf{q}_j\|\} \tag{5.14}$$

Therewith, for each particle $\mathbf{p}_i$, its next position can be calculated according to a time step Δt and the nearest direction vector $\mathbf{v}_j = \mathcal{V}_D(\mathcal{M}(\mathbf{p}_i))$:

$$\mathbf{p}_i(t + \Delta t) = \mathbf{p}_i(t) + \mathbf{v}_j \cdot \Delta t \tag{5.15}$$

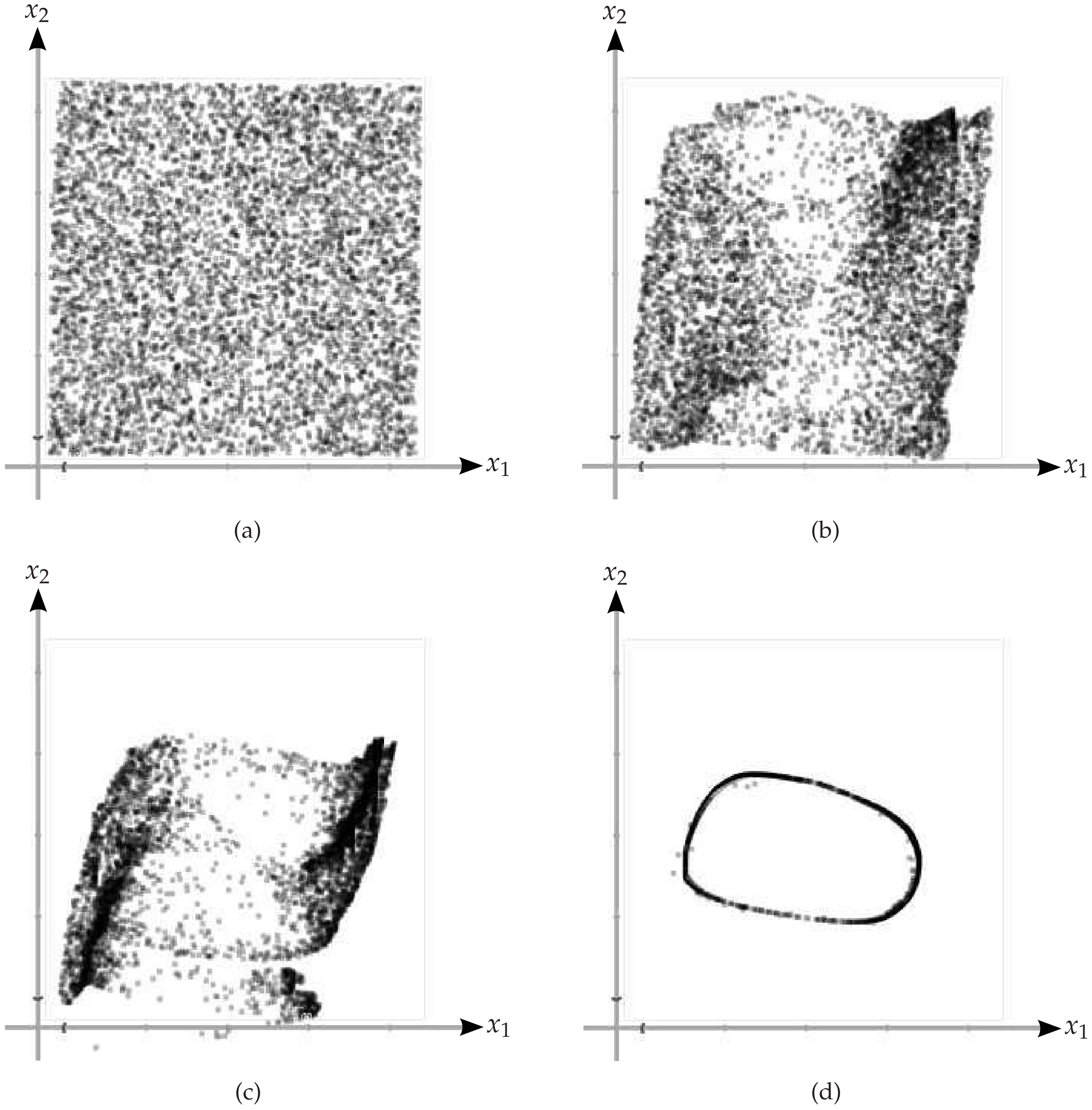
When starting the particle simulation, an equally distributed amount of particles is inserted into the vector field of the state space and for each particle, the nearest vector regarding euclidean distance determines its direction and speed of movement as stated above. Algorithm 7 recapitulates the introduced particle simulation algorithm.

Algorithm 7: Particle Simulation Algorithm.

```
while animation running do
    foreach each particle $\mathbf{p}_i$ in state space do
        detect nearest sample point $\mathbf{q}_j = \mathcal{M}(\mathbf{p}_i)$ with respect to euclidean distance;
        get direction vector $\mathbf{v}_j = \mathcal{V}_D(\mathbf{q}_j)$;
        $\mathbf{p}_i = \mathbf{p}_i + \mathbf{v}_j \cdot \Delta t$
    end
end
```

Each of the particles represents an independent simulation run with the starting position indicating its initial condition. While the visualization is projected to a three-dimensional representation, the motion vector of the particles is calculated with full dimensionality. Thus, the motion is determined by all dimensions of the state space, revealing additional information exceeding the three-dimensional plot.

An exemplary particle animation is illustrated in four steps in Figures 5.14(a) to 5.14(d) for a two-dimensional vector field containing an oscillation.



(a)

(b)

(c)

(d)

Figure 5.14: *Particle simulation for the two-dimensional state space of an oscillator circuit with increasing time from (a) to (d).*

6
Experimental Results

In this section the algorithms altogether forming the proposed new formal verification methodologies are applied to circuit examples. Different properties and circuits are analyzed in order to present practical results for all methodologies presented in Chapter 5.

6.1 Implementation

The trajectory-directed discrete modeling algorithm has been prototypically implemented using GNU Octave for the control flow including the mathematical operations such as the Gram-Schmidt orthogonalization. A prototypical implementation in C++ of a transient simulation back-end has been coupled with GNU Octave as a dynamic library using the SWIG interface compiler.

The ASL syntax grammar was implemented by a C++ LEX/YACC parser. The ASL verification algorithms as well as the other algorithms such as the complete state space-covering input stimuli generation and the transfer of transient simulation waveforms to a DATS have been coded in C++.

The multi-parallel particle simulation and visualization environment also used for all the vector field pictures of the DATS models shown in this thesis has been implemented in C++ using the open source 3D graphics engine (OGRE).

A common data interchange format serializing the DATS model including additional information such as state sets identified by the verification algorithms to a file has been implemented, allowing to transfer DATS models between the discrete modeling algorithm, the verification algorithms and the visualization tool. Additionally, an exporter to a DATS has been implemented into the VERA equivalence checking tool in order to apply the new equivalence checking methodology using complete state space-covering input stimuli to the same model data for comparison.

The runtimes of the experimental examples analyzed in the following have been computed on a single core of an Intel Core 2 Quad with 2.83 GHz and 8 GB of RAM.

6.2 Verification of Initial Conditions of a Ring Oscillator

The modified ring oscillator with an even number of inverter stages and cross-coupling [JKK08] was already presented in Figure 1.3 of Section 1.4 as a motivating example. The critical property to be verified of this circuit is the existence of initial conditions that cause the circuit not to run into an oscillation for certain ratios of the transistor sizes between the inverter chain and the bridges. While first results of the ASL model checking methodology were already demonstrated in the motivating example, the ASL property specification methodology will be discussed in the following.

The properties to verify are the existence of oscillation and proving that the circuit oscillates for every possible initial condition. Hence, the ASL specification shown in Listing 6.1 was developed.

Listing 6.1: *ASL specification for oscillator verification.*

```
# Assert that the circuit oscillates
osci_set=on all select oscillation;
for is_empty(osci_set) assert false;

# Assert that circuit has no non−periodic steady states
for is_empty(DCpoints) assert true;

# Which initial conditions lead into DCpoints?
bad_initial_conditions = reach DCpoints;
```

The macro `is_empty()` is checking if a set is empty, wrapping the following ASL statements for better understandability of the syntax:

```
#macro is_empty(set) −> returns 1 if assertion holds, 0 if not
macro is_empty
{
        parameter2 = for parameter1 assert not all;
}
```

The simple property specification in Listing 6.1 first checks if there is a periodic oscillation trajectory in the state space. The next assertion requires the oscillator's state space not to contain any non-periodic steady states. If those exist, a set `bad_initial_conditions` is assigned with the states that can reach these steady states. Every state from the set `bad_initial_conditions` represents an initial condition that causes the circuit to run into a steady state instead into the oscillation trajectory. The transient simulation in the motivating example, started with such a bad initial condition identified by the ASL model checking algorithms, showed that the circuit in fact does not oscillate.

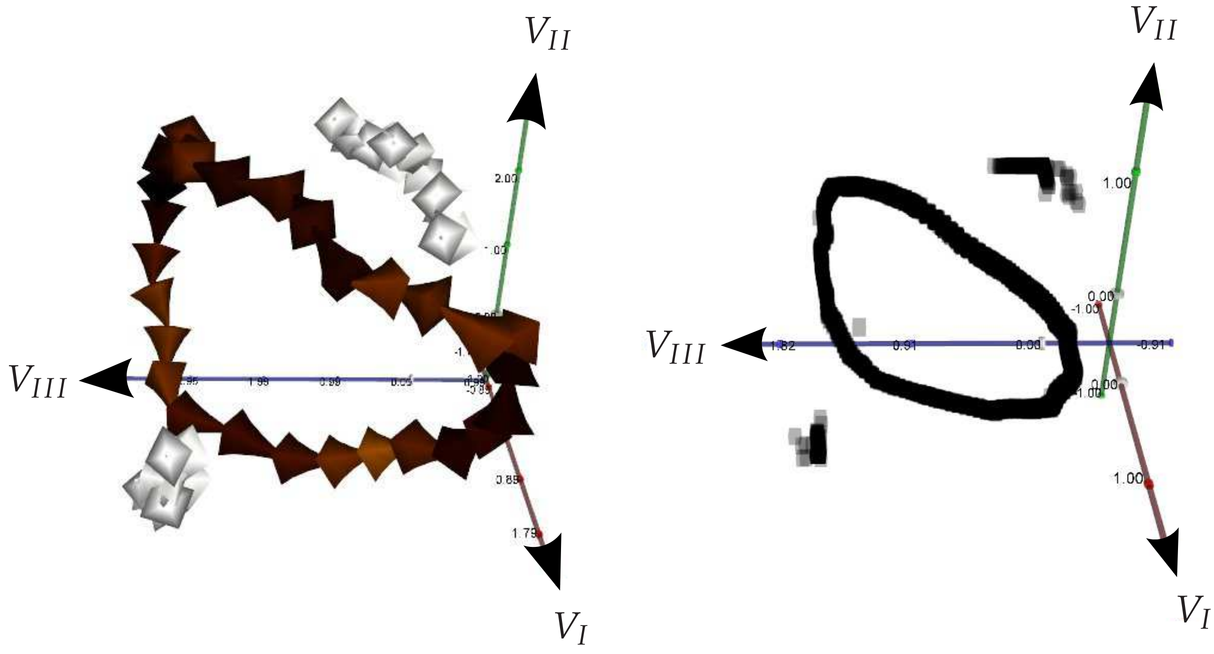
Furthermore, ASL model checking has been conducted for the α/β-ratios 1.05, 1.35, 1.65 and 1.95 in order to systematically check the circuit properties and to prove the consistency of the ASL model checking on the DATS model generated by the trajectory-directed discretization algorithm.

Table 6.1 summarizes the verification results including the oscillation periods reported by the ASL model checking algorithms as well as by transient analysis for $V_{DD} = 3.3$ V. If bad initial conditions were detected by model checking, a transient analysis run was conducted with these conditions in order to prove that the circuit shows the expected behavior. Additionally, information about modeling and model checking runtimes are denoted in the table.

Table 6.1: *Verification results for the modified ring oscillator with results obtained from ASL model checking (MC) and transient analysis (TRA).*

α/β-ratio	1.05	1.35	1.65	1.95
MC reports bad init. cond.	-	-	-	✓
# States DATS	16036	15810	14680	13288
Discretization Runtime	13 : 22 m	12 : 42 m	11 : 24 m	10 : 30 m
MC Runtime	7.5 s	7.1 s	6.0 s	5.4 s
Oscillation period MC	1.494 μs	1.224 μs	1.055 μs	0.961 μs
Oscillation period TRA	1.415 μs	1.159 μs	1.014 μs	0.936 μs

Figure 6.1(a) shows the transition vectors between states of the detected oscillation set and the non-periodic steady states identified by ASL model checking for an α/β-ratio of 1.95 projected to the state space dimensions V_I, V_{II} and V_{III}. Figure 1.5, presented in the motivating example, already illustrated the state space trajectories leading into the non-periodic steady states. Additionally, a multi-parallel state space particle simulation with the methodology introduced in Section 5.7 was executed. A snapshot of the dynamic motion flow is shown in Figure 6.1(b), allowing to visually obtain the same conclusions as from model checking.



(a)

(b)

Figure 6.1: *Transition vectors between states of the detected oscillation set and non-periodic steady states of the modified ring oscillator detected by model checking for α/β-ratio 1.95 (a). Particle simulation visualizing the oscillation trajectory and the non-periodic steady states (b).*

6.3 Verification of Active Lowpass Filter Overshoot

The Sallen-Key biquad lowpass filter shown in Figure 6.2 has a tendency to overshoot beyond the designed passband gain of 1.05 which is the property to be verified in the following. The overshoot around the cutoff frequency of 1 kHz may be a result of the complex conjugate poles in the transfer function and could have been overlooked in the design process. The parameters are $C_1 = 5$ nF, $C_2 = 50$ nF, $R_1 = R_2 = R_4 = 10$ kΩ, $R_3 = 500\ \Omega$. The input voltage range is ± 1 V and the operational amplifier represented by a behavioral model has a supply voltage of ± 3 V.

The overshoot property shall be analyzed using the methodologies of:

1. ASL model checking with counterexample generation and
2. transient simulation controlled by complete state space-covering input stimuli with the simulation results re-transferred to a DATS and evaluated by the ASL verification algorithms using the same ASL property specification as used for model checking.

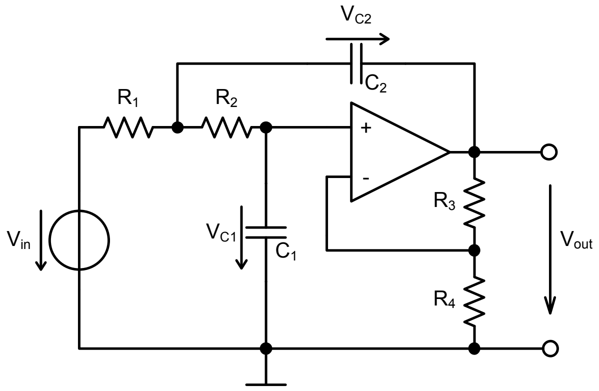


Figure 6.2: *Circuit schematic of Sallen-Key biquad lowpass filter.*

In order to apply the verification algorithms, a DATS model for the state space dimensions V_{in}, V_{C_1} and V_{C_2} has been generated using the trajectory-directed discretization approach. The DATS model in the following used for the results was generated with 3702 vertices and model generation requiring 142 seconds. While generating a higher number of states further increases modeling accuracy for model checking, the complete state space-covering stimulus covers all corners of the reachable area already with a significantly lower number of states but at the cost of then decreasing the density of the internal trajectories. Table 6.2 summarizes the discretization runtimes for different state counts. The number of states generated can be controlled in the discretization algorithm by constraining the length of transition vectors to a user defined interval.

The transient simulation steps needed for state space sampling have a constant runtime for a given circuit. Hence, the overall simulation runtime scales linearly with the number of generated states. However, the checking of the proximity criterion for the n sampled states, although consuming substantially less time than the transient step computation, has a runtime of $\mathcal{O}(n \log n)$ which can be identified in Table 6.2 by the slow nonlinear growth of the runtimes.

Table 6.2: *Runtimes of the trajectory-directed discretization algorithm for the Sallen-Key lowpass filter.*

Number of states	3702	6673	13288
Discretization Runtime	02 : 22 m	05 : 40 m	15 : 37 m

The set `reachable` of 1546 states reachable from DC-operating-points has been identified using the ASL statement:

```
reachable = on all reach from DCpoints;
```

The reachable transition vectors between states of the state space, spanned by the input voltage V_{in} and the voltages over the two capacitances V_{C_1} and V_{C_2}, are visualized in Figure 6.3.

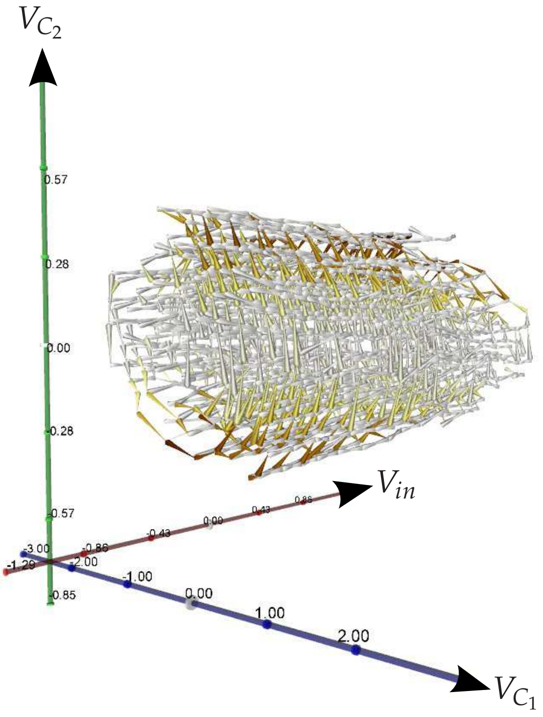


Figure 6.3: *Reachable state space of the Sallen-Key biquad lowpass filter.*

6.3.1 Model Checking

The ASL specification for the overshoot property is denoted in Listing 6.2, calculating the minimum and maximum of the input and output values of the reachable states. For the relation of the maximum and minimum output voltages to their respective input voltages, the overshoot ratio is calculated and an assertion is formulated, allowing a maximum overshoot ratio of 1.05 which represents the passband gain of the circuit.

The results calculated by the ASL model checking algorithms on the DATS are presented in Table 6.3, additionally giving the values for V_{C_1} and V_{C_2} that will be discussed in connection with the verification using the complete state space-covering input stimulus. An overshoot ratio of 1.734 has been calculated by the verification algorithms, causing the assertion not to be satisfied. The overall runtime of the model checking algorithms including reachability computation was 3.55 seconds.

Listing 6.2: *ASL specification for overshoot property.*

```
on reachable assign(%max_V_in,max) value(V_in);
on reachable assign(%min_V_in,min) value(V_in);
on reachable assign(%max_V_out,max) value(V_out);
on reachable assign(%min_V_out,min) value(V_out);

%overshoot_ratio = (%max_V_out-%min_V_out)/(%max_V_in-%min_V_in);

for %overshoot_ratio assert [<= 1.05];
```

Table 6.3: *Verification results calculated by the ASL model checking algorithms on the DATS for the Sallen-Key lowpass filter.*

Value	Minimum	Maximum
V_{out}	-1.714 V	1.754 V
V_{C_1}	-1.630 V	1.663 V
V_{C_2}	-0.562 V	0.512 V

6.3.2 Counterexample

Subsequently, a counterexample input stimulus shall be generated in order to analyze the erroneous behavior violating the specification in a transient simulation test bench using the following ASL statement to create a piecewise linear waveform for V_{in} and V_{out}:

```
counterexample from DCpoints to (reachable and V_out[< -1.5]);
```

While the piecewise linear signal for V_{in} represents the input stimulus that has to be applied to the circuit to reach an output voltage < -1.5 V, the signal waveform generated for V_{out} represents the expected behavior of the output voltage. Moreover, comparing this expected output behavior, in the following referred to as V_{out}^{expect}, with the transient response V_{out}^{sim} to the counterexample input stimulus allows to rate the modeling quality of the DATS and therewith the soundness of the verification results. As can be seen in Figure 6.4, the expected behavior from the DATS model and the transient response match quite well with the counterexample, reliably reaching the desired output voltage. The generation of the counterexample stimulus on the DATS took 0.65 seconds.

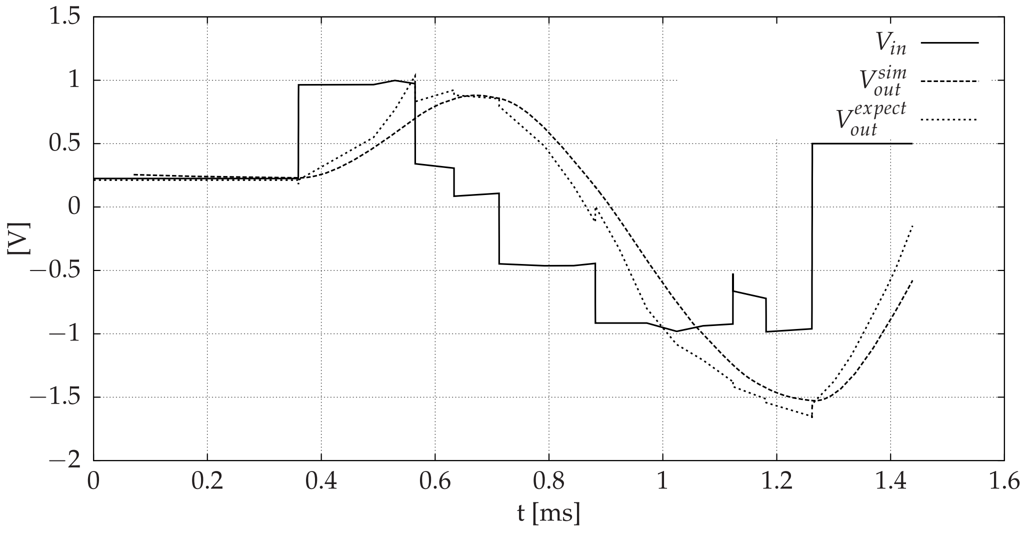


Figure 6.4: *Transient response V_{out}^{sim} to the counterexample input stimulus V_{in} of the Sallen-Key biquad lowpass filter with the expected response V_{out}^{expect} predicted by the counterexample algorithm on the DATS.*

6.3.3 Complete-Coverage Input Stimulus

As an alternative to the model checking methodology with counterexample generation, a complete state space-covering input stimulus as introduced in Section 5.5 was generated with a computation time of 3.93 seconds. The input stimulus consists of 31951 time and value tuples with an overall stimulus time length of 891 milliseconds.

Figure 6.5 provides an impression of the complete state space-covering input stimulus and the transient response of the circuit.

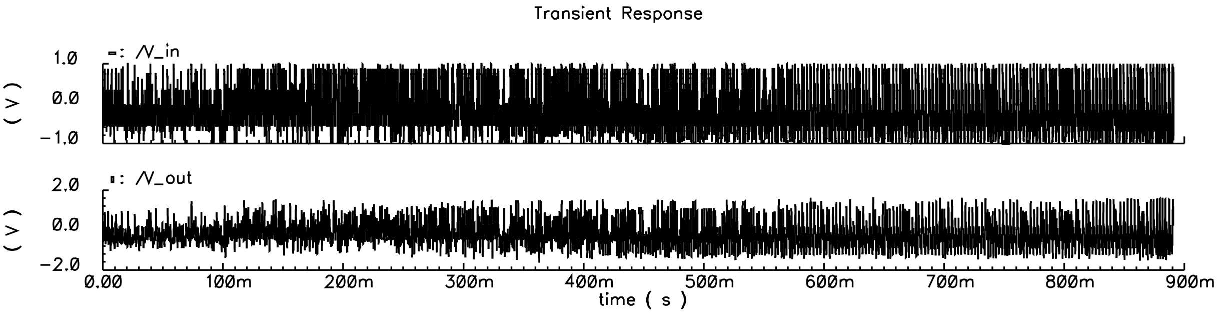


Figure 6.5: *Complete generated input stimulus and transient output response of the Sallen-Key biquad lowpass filter.*

While the input stimulus generation algorithm is designed to cover every vertex of the modeled circuit, which can easily be verified, it is necessary to prove that the generated stimulus really forces the circuit to adopt every reachable state of the system during transient simulation. In order to prove complete coverage of the system variables, in Figure 6.6 the transient circuit response to the input stimulus is plotted over V_{C_1} and V_{C_2}, showing that the trajectories completely cover the reachable coordinate pairs for V_{C_1} and V_{C_2}. The differences in the concentration of the trajectories are caused by the projection of the input variable's axis into the two-dimensional plot. Additionally, the number of sampled states of the state space discretization affects the density of the trajectories. In Figure 6.7 an excerpt of the simulated transient response V_{out}^{sim} and the signal V_{out}^{expect} computed on the DATS model generated by the new trajectory-directed discretization approach is shown, clearly illustrating that the behavior computed on the DATS model matches the transient response with high accuracy.

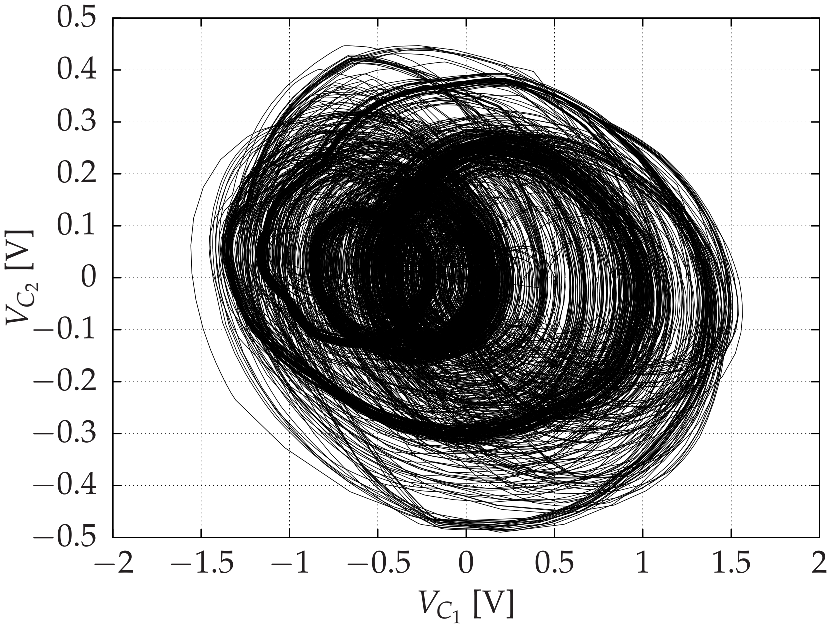


Figure 6.6: *Transient response to the generated input stimulus of the Sallen-Key biquad lowpass filter plotted over V_{C_1} and V_{C_2}.*

6.3.4 Comparison to Hyperbox Discretization

For comparison of the discretization quality, a DATS model with 3333 states of which 2060 are reachable from DC-operating-points has been generated using the hyperbox discretization approach [HHB02a] with a computation time of 87 seconds. On this model, a complete-coverage input stimulus was generated with an overall stimulus time length of 1.89 seconds. The transient response V_{out}^{sim} and the predicted behavior

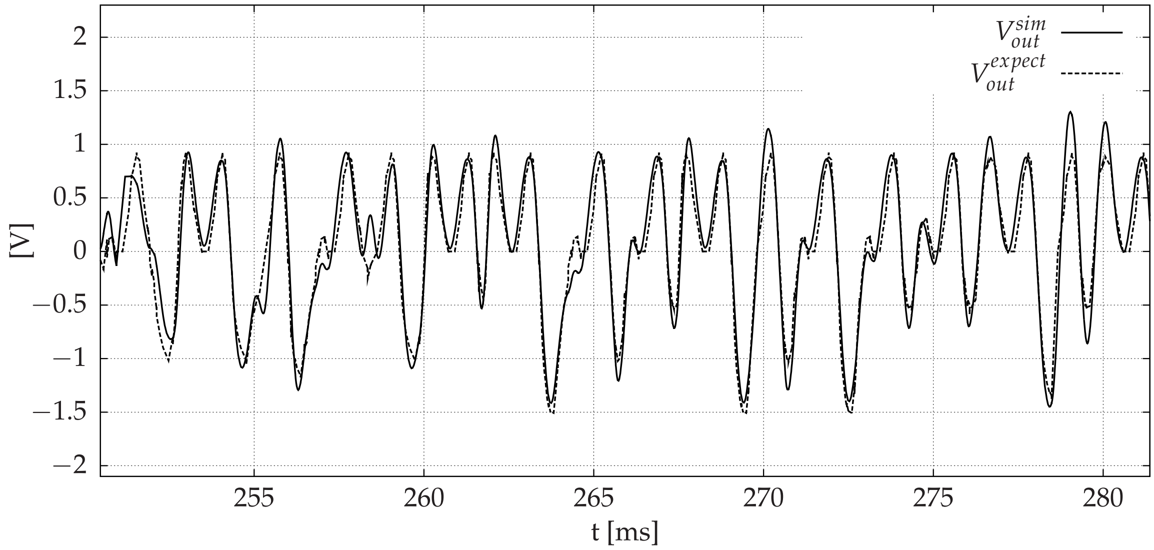


Figure 6.7: *Excerpt of the transient response* V_{out}^{sim} *to the complete-coverage input stimulus of the Sallen-Key biquad lowpass filter with the expected response* V_{out}^{expect} *predicted by the stimulus generation algorithm on the DATS, generated by the trajectory-directed discretization approach.*

V_{out}^{expect}, calculated on the DATS, are shown in Figure 6.8. In comparison to the matching waveforms computed by the trajectory-directed discretization, the results obtained by the hyperbox discretization show an over-approximation of the reachable area. Moreover, as all transition steps in the state space can only be either in the input direction or paraxial, the rectangular shape of the expected waveform V_{out}^{expect} does not equally well conform to the transient response. Model checking results of 2.915 for the overshoot ratio property correspondingly exhibit an over-approximation of the reachable area due to not matching the state space dynamics with the discrete transitions in the DATS.

A comparison of the successor relation error ϵ_{suc} as defined in Section 2.4.2 supports the conclusion of a large improvement of the trajectory-directed discretization over the hyperbox discretization. For the trajectory-directed discretization, ϵ_{suc} is 5.11° with an average out-degree of 0.94, compared to 29.58° for ϵ_{suc} of the hyperbox discretization with an average out-degree of 3.23.

6.3.5 ASL Verification on Simulation Waveforms

For application of the same ASL overshoot property specification that was introduced in connection with application of the model checking approach in Section 6.3.1, the transient simulation response to the complete state space-covering input stimulus was

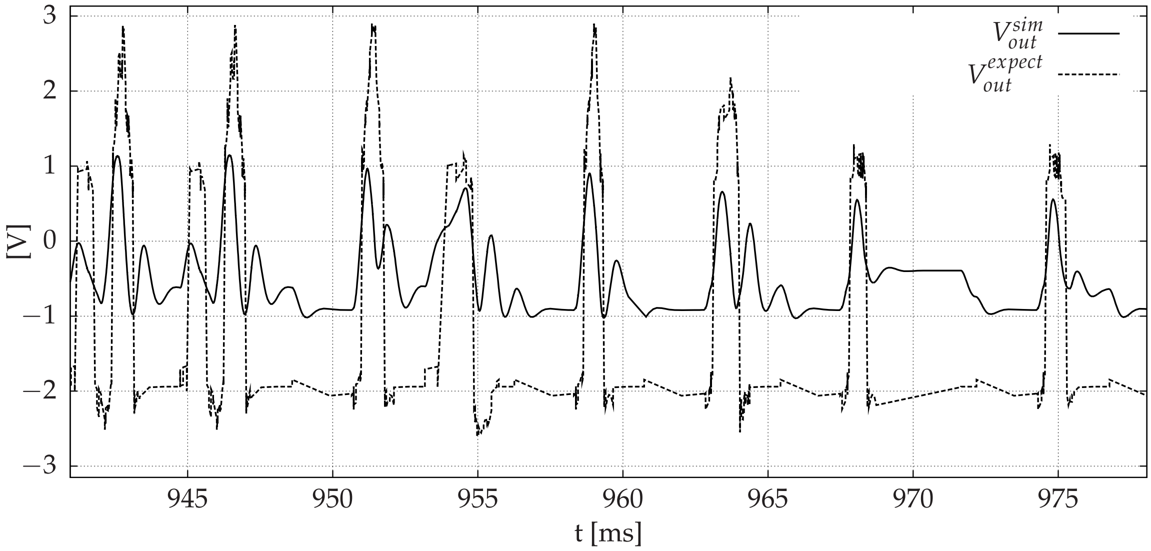


Figure 6.8: *Transient response V_{out}^{sim} to the complete-coverage input stimulus of the Sallen-Key biquad lowpass filter with the expected response V_{out}^{expect} predicted by the stimulus generation algorithm on the DATS, generated by the* hyperbox *discretization approach.*

transferred back into a DATS state space representation with the approach described in Section 5.3. The waveforms combined into a DATS model represent the signals V_{in}, V_{out}, V_{C_1} and V_{C_2}. The transfer to a DATS model took 59 seconds, resulting in 126337 states of the model, created from the 159052 simulation time steps. Table 6.4 compares the results of the ASL evaluation on the simulation waveforms after transferring them into a DATS (Min Sim, Max Sim) to the ASL model checking results on the discrete state space model generated by the trajectory-directed approach (Min MC, Max MC). The model checking on the DATS slightly over-approximates the reachable area compared to the transient simulation with the complete-coverage stimulus. Such differences can be the result of zero time input signal changes in the discrete model, while the complete-coverage stimuli have been restricted regarding their maximum signal edge steepness.

As the transient response covers the complete reachable state space of the circuit under verification, another automated verification methodology is given that can detect the overshooting behavior of the Sallen-Key circuit.

Table 6.4: *Verification results calculated by the ASL model checking (MC) algorithms on the DATS for the Sallen-Key lowpass filter compared to the ASL evaluation on the transient simulation results (Sim) using a complete-coverage input stimulus.*

Value	Min Sim	Min MC	Max Sim	Max MC
V_{out}	−1.636 V	−1.714 V	1.64212 V	1.754 V
V_{C_1}	−1.558 V	−1.630 V	1.563 V	1.663 V
V_{C_2}	−0.489 V	−0.562 V	0.447 V	0.512 V

6.4 Verification of CMOS Charge Pump Startup Time

The charge pump circuit shown in Figure 6.9 is a clocked step-up voltage converter allowing output voltages nearly twice the supply voltage. The parameters are $V_{DD} = 3$ V, $clk = 50$ kHz, $R_{load} = 1\ \mathrm{M\Omega}$, $C_1 = C_{load} = 1$ nF.

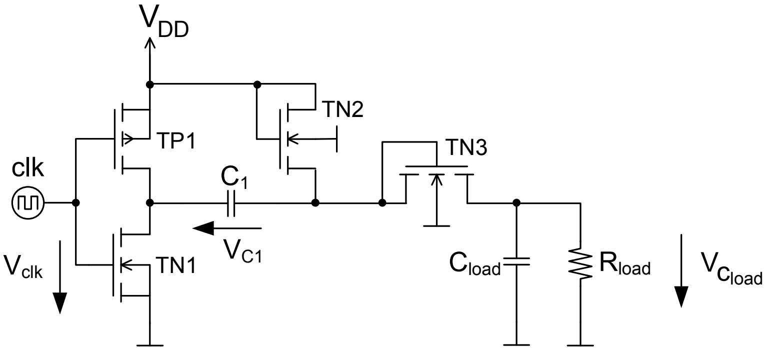


Figure 6.9: *Circuit schematic of the CMOS charge pump.*

A characteristic output waveform for $V_{C_{load}}$, as shown in Figure 6.10, is calculated by a transient analysis starting from the initial condition of $V_{C_1} = V_{C_{load}} = 0$ V, showing the typical ripple of the clocked switching behavior.

6.4.1 Model Checking

A DATS model of the charge pump circuit shall be generated using the trajectory-directed discretization method. The circuit has four state space dimensions: V_{C_1}, $V_{C_{load}}$ and two additional dimensions for a behavioral model of a clock generator that has to be included in the DATS model in order to obtain verification results for the defined clock frequency. If the clock was considered as an input, all possible clock frequencies would be contained in the model, not allowing to obtain expressive results.

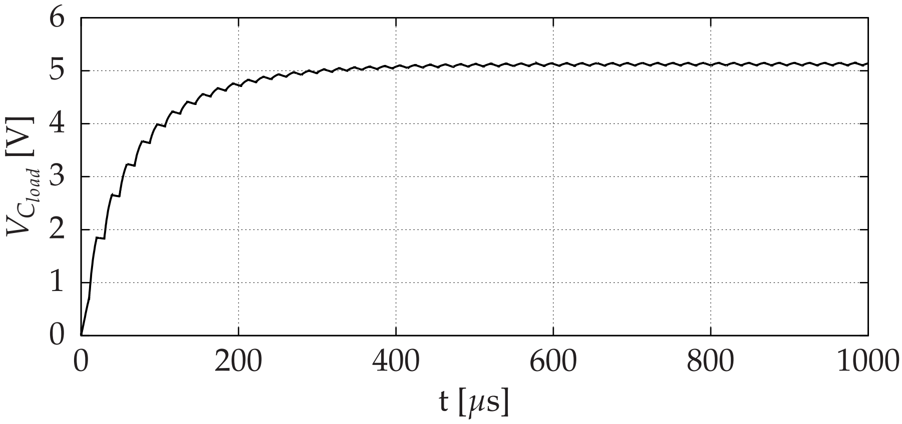


Figure 6.10: *Transient output waveform of the CMOS charge pump circuit.*

The generated DATS model selected for further analysis consists of 13055 states which consumed a model generation time of 18 minutes and 39 seconds. Table 6.5 summarizes the discretization times for other state numbers for comparison. While a higher number of states hardly improves the accuracy of the verified properties, a lower number of states, although well capturing the startup behavior, is not sufficient to model the periodic output ripple.

Table 6.5: *Runtimes of the trajectory-directed discretization algorithm for the CMOS charge pump.*

Number of States	5533	13055	28892
Discretization Runtime [mm:ss]	05:08	18:39	61:07

On the DATS model, the ASL property specification methodology for startup time verification of autonomous circuits presented in Section 5.2.3 is now applied. Therefore, the start area for the startup time verification has to be selected:

```
startarea = (value(V_C_load)[<0.1]) and (value(V_C_1)[>-0.1]);
```

The states reachable from this start area contain the dynamic startup behavior of the circuit on which the maximum output voltage can be measured and assigned to the number variable `%max_V_C_load`:

```
reachable = reach from startarea;
on reachable assign(%max_V_C_load,max) value(V_C_load);
```

The value assigned to `%max_V_C_load` is 5.15885 V. Within the states reachable from the start area, the minimum and maximum startup time is measured using the transition operation, assigning the computed states of the startup paths to the set `startup`:

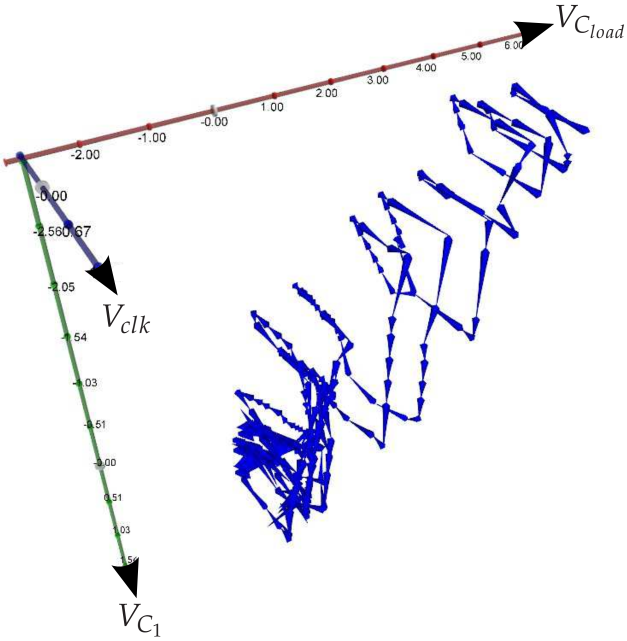


Figure 6.11: *Startup trajectory projected to V_{C_1}, $V_{C_{load}}$ and V_{clk}, identified by evaluating the ASL property specification on the DATS of the charge pump.*

```
startup = assign(%startup_time_min,min) assign(%startup_time_max,max)
  transition startarea to value(V_C_load)[>= 0.9 * max_V_C_load];
```

The set `startup` contains 648 states and the minimum startup time reported is 114 μs. The maximum startup time assigned to the number variable `%startup_time_max` is 172 μs.

By applying the ASL algorithms to the circuit model, the set `startup` is acquired as visualized in Figure 6.11.

The periodic steady states where the circuit reaches its maximum output voltage over $V_{C_{load}}$ with a periodic signal ripple shall be analyzed. Therefore, these periodic steady states, which form a cycle of transitions in the state space, are identified and the minimum and maximum values of $V_{C_{load}}$ are measured by the following ASL statements:

```
periodic_output_set = on reachable select oscillation;
on periodic_output_set
  assign(%min_V_C_load_ripple,min) assign(%max_V_C_load_ripple,max)
    value(V_C_load);
```

The oscillation detection algorithm reports a ripple period of 22.0234 μs which slightly deviates from the clock cycle time of 20 μs. The number variable

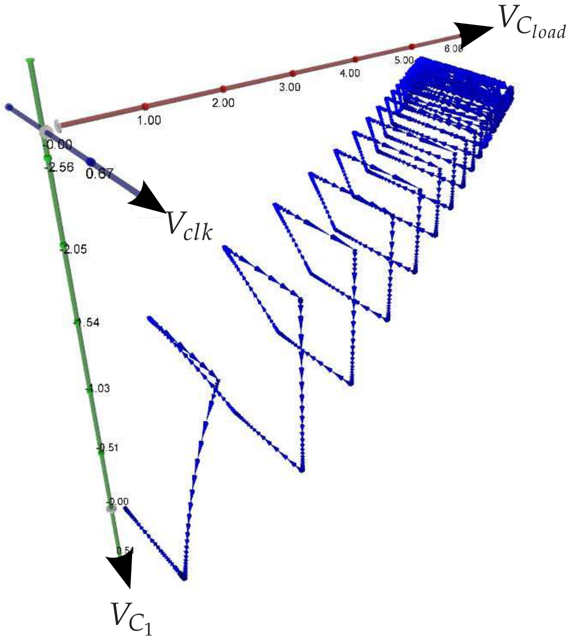


Figure 6.12: *Transient simulation waveforms of the charge pump for V_{C_1}, $V_{C_{load}}$ and V_{clk} transferred into a DATS representation for application of ASL verification algorithms.*

`%min_V_C_load_ripple` is assigned with 5.0889 V and `%max_V_C_load_ripple` contains 5.15885 V.

The model checking algorithms for evaluating all the described properties have a runtime of 18.28 seconds.

6.4.2 ASL Verification on Simulation Waveforms

A transient simulation for the startup process was conducted and the simulation waveforms for all four state space variables of the charge pump circuit were transferred into a DATS model in order to apply the previously described ASL specification on these waveforms. The transient simulation contains 8424 time steps for a simulation time of 1 millisecond. The transfer to a DATS model consisting of 3528 vertices and 4539 transitions took 0.35 seconds. The lower number of states compared to the simulation time steps is due to the ripple behavior at the maximum output voltage mapping to a periodic cycle of states. Figure 6.12 illustrates the DATS model generated from the transient simulation data, showing a similar structure compared to the startup trajectory identified in the DATS generated by the trajectory-directed discretization method in Figure 6.11. The application of the same ASL specification as used for model checking takes 10.6 seconds and the ASL verification results for model checking and ASL property checking on transient simulation waveforms are summarized in Table 6.6. As can be concluded from the comparison of the results, the model checking on a discrete state space model can verify the specified properties with high accuracy.

Table 6.6: *Comparison between ASL property evaluation on a DATS generated by the trajectory-directed discretization (TDD) method and on simulation waveforms obtained from transient analysis (TRA).*

Property	TDD	TRA
Max. output voltage	5.1589 V	5.1503 V
Startup time	[114 μs, 172 μs]	169 μs
Periodic output ripple	[5.0889 V, 5.1589 V]	[5.1008 V, 5.1503 V]

6.5 Model Checking of VCO Gain K_{VCO}

The considered VCO circuit illustrated in Figure 6.13 is an opamp-based CMOS design for demonstrating the ASL model checking methodology for verification of K_{VCO} and the linearity of the VCO that was introduced in Section 5.2.2. The circuit parameters are $V_{DD} = 2.5$ V, $V_{SS} = -2.5$ V, $g = 10^{-6}$, $R_1 = 20$ kΩ, $R_2 = 1$ MΩ, $C_1 = 100$ nF, $C_2 = 1$ nF. The oscillation of the circuit is caused by the inverter feedback loop alternately charging C_2. Thus, the oscillation period is controlled by the ideal voltage controlled current source gV_{in} determining the charge current of capacitor C_2 via current mirrors.

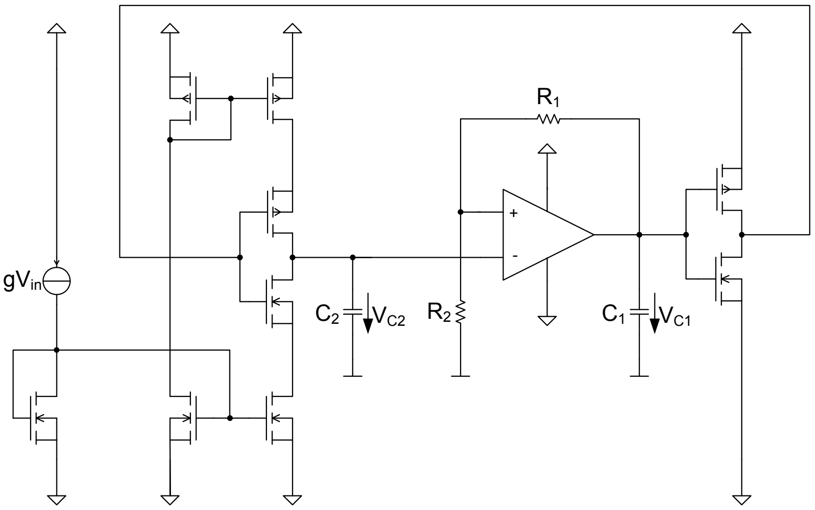


Figure 6.13: *Circuit schematic of the voltage controlled oscillator.*

The considered input voltages for gain verification are 0.66 V, 0.83 V, 1.00 V and 1.17 V. Figure 6.14 shows the corresponding oscillation areas in state space isolated by applying the methodology from Section 5.2.2 to the DATS circuit model generated by the trajectory-directed discretization algorithm. The DATS model consisting of 15660 states was generated in 16 minutes and 57 seconds. Although the circuit only has a three-dimensional state space, such a high number of states is necessary in order to accurately capture the oscillation period for gain verification. The state space slices contained in one single DATS with all state transitions for the four input voltage values are illustrated in Figure A.1 in Appendix A.1. From this transition structure, the oscillation sets are identified by the ASL algorithms. The adaption of the trajectory-directed discretization to the changed angles and dynamics of the transition structure at the different input voltages can be registered clearly.

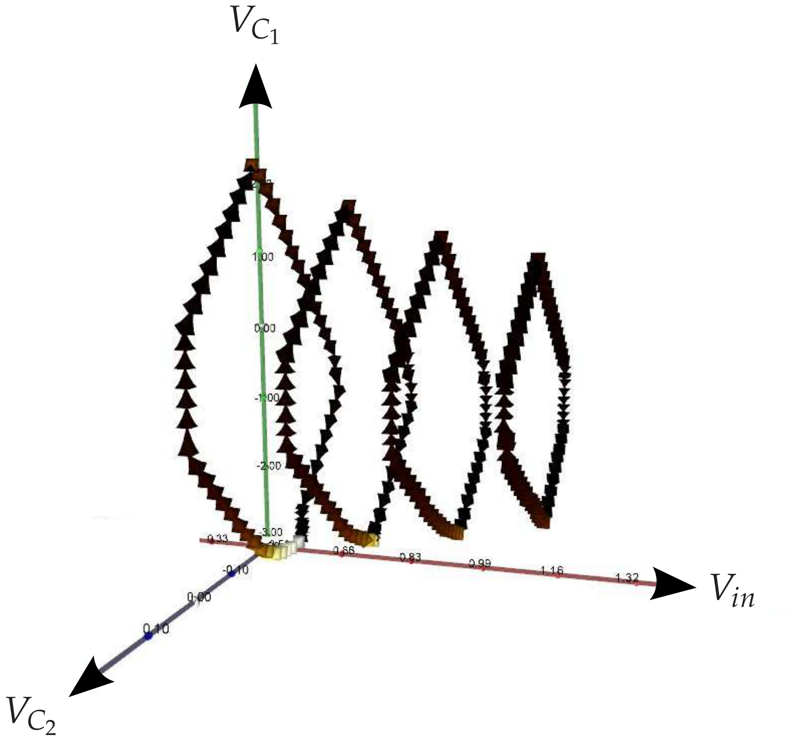


Figure 6.14: *Oscillation areas of the voltage controlled oscillator at different control voltages.*

The ASL verification run takes 14 seconds and the acquired results for the oscillation period of the VCO at the specified input voltage steps are summarized in Table 6.7 with a comparison to transient analysis results. In Table 6.8, the results for the gain property of the VCO are detailed in comparison to transient analysis results.

Table 6.7: *Comparison of the results between ASL model checking and transient analysis for the oscillation period of the VCO at different input voltages.*

V_{in}	**ASL MC**	**Transient Analysis**
0.66 V	1391 μs	1394 μs
0.83 V	1139 μs	1147 μs
1.00 V	972 μs	969 μs
1.17 V	835 μs	840 μs

Table 6.8: *Comparison of the results between ASL model checking and transient analysis for the gain K_{VCO} of the VCO.*

V_{in} **Range**	**ASL MC**	**Transient Analysis**
0.66 V, 0.83 V	959 Hz/V	931 Hz/V
0.83 V, 1.00 V	909 Hz/V	967 Hz/V
1.00 V, 1.17 V	1020 Hz/V	955 Hz/V

Compared to transient analysis, a small error of the values calculated by model checking can be noticed. The deviation of the relative gain for the different input voltage tuples between model checking and transient analysis lies within the discretization induced error range.

6.6 Equivalence Checking with Complete-Coverage Stimuli

In this section, the new equivalence checking methodology based on complete-coverage stimuli, in the following referred to as stimEC, is applied to example circuits and corresponding behavioral models. Where possible, the obtained results for each of the circuits are compared to a verification that has been performed using the VERA equivalence checking methodology [HKH04] described in Section 4.5.3.1. The VERA methodology directly compares the dynamic behavior in the state spaces of the systems under verification by mapping the state spaces using a nonlinear transformation to a canonical representation for each system. This is only possible for implementations that do not differ substantially in their internal structure. Besides the comparison of the two approaches with common circuit examples, the new stimEC approach will be applied to a delta-sigma modulator circuit that cannot be processed by the VERA tool due to the comparison to a highly abstracted behavioral model.

The DATS models needed for application of the stimEC stimuli generation algorithm are directly exported from the state space sampled by the VERA algorithms in order to compare the verification methodologies based on equal input data. The DATS export is implemented into VERA by exporting the sampled points in the state space. The successor relation of the DATS is determined by a local search for each point $\mathbf{p}_i$ which selects an adjacent state $\mathbf{p}_j$ as successor if a transient step vector $\mathbf{v}_i$ starting in $\mathbf{p}_i$ points towards $\mathbf{p}_j$ more than towards any other state. Although transition paths using this DATS modeling are not corresponding to state space trajectories equally well as those generated by trajectory-directed discrete modeling, the transition structure is sufficient for covering the reachable area of the circuits under verification.

The DATS model for complete-coverage input stimulus generation of the delta-sigma modulator is generated using the trajectory-directed discretization approach. The 8-dimensional state space is sufficiently captured to generate a stimulus that covers the reachable state space of the circuit.

6.6.1 Biquad Bandpass Filter

The first example circuit considered is a biquad bandpass filter illustrated in Figure 6.15 with $C_1 = C_2 = 1\ \mu\text{F}$, $R_1 = R_2 = R_4 = 10\ \text{k}\Omega$, $R_3 = 30\ \text{k}\Omega$, $R_5 = 20\ \text{k}\Omega$, $V_{DD} = 2.5$ V, $V_{SS} = -2.5$ V, $V_{in} = \pm 0.7$ V. The used op-amp is a simple 8-transistor CMOS design. This transistor netlist representation shall be compared with a VHDL-AMS behavioral model partially illustrated in Listing 6.3 in order to show that the behavioral model can be used for faster system simulation. Therefore, for the transistor netlist of this circuit, a complete-coverage stimulus containing 8648 time and value pairs with a time length of 300 ms is created and the circuit netlist as well as the behavioral model are simulated with this stimulus. The input stimulus and the simulation results are shown in Figure 6.16. Using the difference error measure, an error value of 11% is reported. The VERA equivalence checking method reports 1.32% of difference. The higher error reported by the stimEC method is due to the differing high-frequency behavior in the beginning of the simulation not equally being captured by the VERA method.

The state space of the biquad bandpass circuit transistor netlist is spanned by the input and the voltages over the capacitors. Hence, a complete coverage of the state space can be proven by plotting the transient response to the input stimulus over V_{C_1} and V_{C_2}. As illustrated in Figure 6.17, the transient response is covering the reachable states of the circuit.

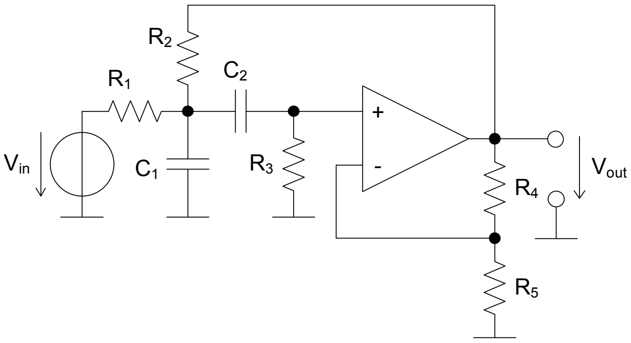


Figure 6.15: *Circuit schematic of the biquad bandpass filter circuit netlist.*

Listing 6.3: *Excerpt from VHDL-AMS behavioral model of the bandpass filter.*

```
ENTITY bandpass IS
  GENERIC( den0 : real := ... );
  PORT( terminal inp, outp, gnd : electrical );
end BANDPASS;

ARCHITECTURE behave OF bandpass IS
  QUANTITY uout ACROSS iout THROUGH outp TO gnd;
  ...
BEGIN
  uint == 1E-3*i_uint'dot+4E-7*i_uint;
  uint == 1.0/den1*(-den0*0.001*i_uint-den2*uint'dot+num1*uin);
  uout == uint+Rout*iout;
  ...
END behave;
```

6.6.2 Second-Order Delta-Sigma Modulator

Due to the high clock frequencies of delta-sigma modulators, they require plenty of simulation time. Hence, for faster simulation of mixed-signal systems containing delta-sigma modulators, behavioral models are used. With the new stimEC approach, the comparison of a transistor netlist implementation of a second-order delta-sigma modulator versus a simple unclocked behavioral model using an allpass filter as a delay component is performed. Due to the massive abstraction differences between the two implementations, the VERA approach cannot be applied to this task as it cannot identify common state space characteristics for mapping the transformation.

Figure 6.18 shows the transistor netlist implementation of the second-order delta-sigma modulator with $C_1 = C_2 = 200$ pF, $R_1 = R_2 = R_5 = R_6 = 100$ kΩ, $R_3 = R_4 = 5$ kΩ, $V_{DD} = 1$ V, $f_{clk} = 1$ MHz. The sequential bitstream is directed into a lowpass

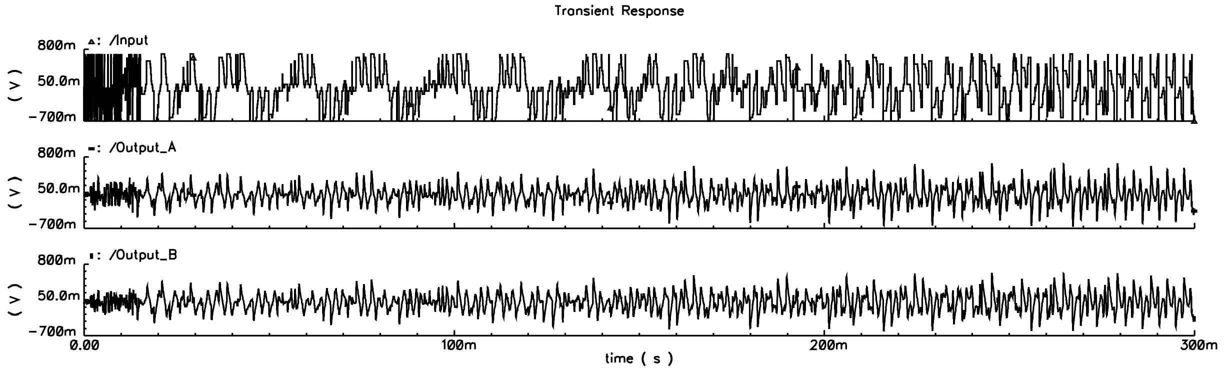


Figure 6.16: *Complete generated input stimulus and transient output response of the biquad bandpass filter transistor netlist (Output_A) and VHDL-AMS behavioral model (Output_B).*

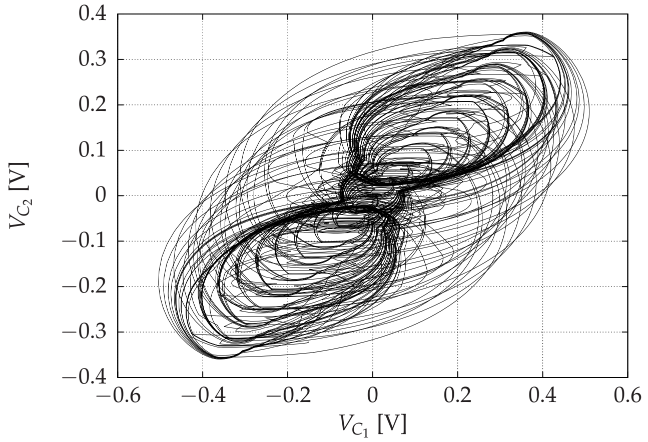


Figure 6.17: *Transient response to the generated input stimulus of the biquad bandpass filter plotted over V_{C_1} and V_{C_2}.*

filter for further processing. A simple behavioral model for this circuit is implemented by an allpass as a delay circuit followed by the same lowpass filter as illustrated in Figure 6.19 with $C_1 = 10$ nF, $R_1 = R_2 = R_3 = 1$ kΩ.

The complete state space covering input stimulus generated for the transistor netlist implementation of the delta-sigma modulator contains 32690 time and value pairs and a system simulation time of 4.34 ms. After simulation of both circuit implementations with the stimulus, the difference error measure is applied to the output voltage waveforms, reporting an output error of 10.67% when excluding the startup time of 0.1 ms where the netlist implementation needs to lock to the feedback chain.

Although the model used for stimulus generation with 21001 states generated by the trajectory-directed discretization algorithm in 3 hours and 29 minutes suffers from

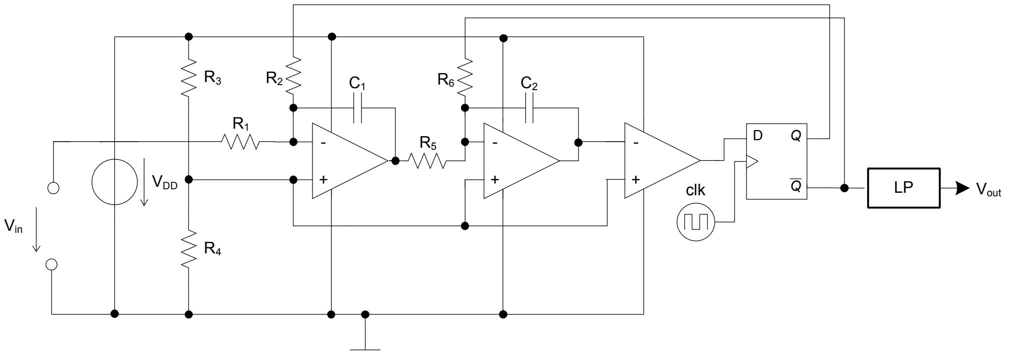


Figure 6.18: *Circuit schematic of the second-order delta-sigma modulator.*

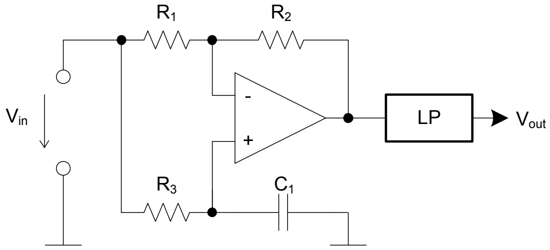


Figure 6.19: *Simple behavioral model for the second-order delta-sigma modulator using an allpass filter for signal delay modeling.*

the few states sampled for every of the 8 dimensions, the reachable state space of the internal capacitors C_1 and C_2 of the netlist implementation is covered quite well. The transient response to the input stimulus is plotted over V_{C_1} and V_{C_2} as depicted in Figure 6.20. The trapezoidal appearance is caused by the correlation of the inverting integrators, restricting the reachable value combinations of V_{C_1} and V_{C_2}.

6.6.3 Further Circuit Examples

Table 6.9 summarizes the equivalence checking results including runtime information, the number of state space dimensions and the number of states in the graph structure used for stimuli generation of the aforementioned biquad bandpass and the delta-sigma modulator, comparing the new stimEC approach to the VERA approach where applicable. In addition, three other circuit examples have been processed with stimuli generated for the transistor netlists. The results for a log domain filter, a Schmitt-trigger and a transistor switch with enable and power down functionality all indicate the feasibility of the new stimEC approach, producing results very similar to the VERA method. Runtimes of the stimEC approach on the VERA-generated DATS are dominated by the transient simulation runs with the generated stimuli, while the stimuli

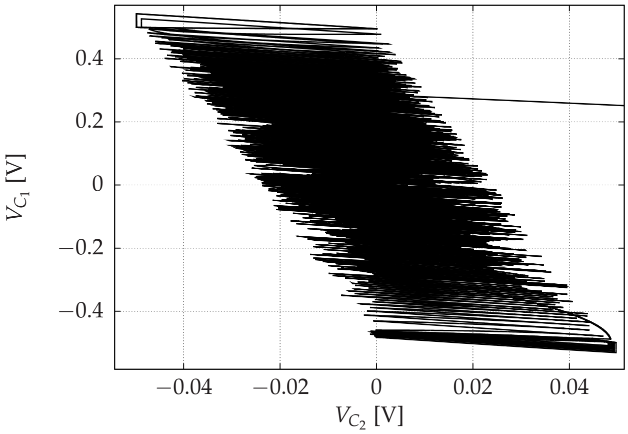


Figure 6.20: *Transient response to the generated input stimulus of the delta-sigma modulator transistor netlist plotted over* V_{C_1} *and* V_{C_2}.

generation and the error computation on the waveforms only require a small fraction of the overall runtime.

In order to compare the runtime results to the new stimEC approach, approximated runtimes of a conventional systematic simulation have been calculated for the five example circuits. While systematic simulation cannot guarantee the complete coverage of the circuits' behavior, it is a common approach for circuit characterization. For the delta-sigma modulator with a clock frequency of 1 MHz, a systematic simulation with sine waves over three decades with 10 amplitude levels and 10 samples per decade starting at 100 Hz could be performed. A single sine wave at 100 Hz already needs a system time of 10 ms with time steps being very small due to the high clock frequency. Without having performed the remaining simulations for higher frequencies, the 10 amplitude levels result in a system time of 100 ms, which is already over 20 times the length of the transient response generated by the stimEC approach. However, the high modeling time for the delta-sigma simulator dominates the runtime of 3 hours and 29 minutes while the stimulus generation, transient simulation and automatic error calculation for both circuit implementations is only consuming less than 3 minutes.

For the remaining circuit examples, performing such systematic simulations leads to higher simulation times than those of the stimEC approach. Table 6.10 summarizes the number of input time steps, the transient response time steps and the runtimes of the stimEC approach compared to the approximated runtimes of a conventional systematic simulation.

6.7 Assessment

The experimental results presented in this section showed the applicability of the formal verification methodologies developed in the scope of this thesis. Although the circuit examples are limited to block level, complex circuit properties have been verified. With the detection of the ring oscillator circuit's bad initial conditions, the advantage of state space-based complete verification methods could be emphasized. By applying different methodologies to the same example circuit, the consistency and exchangeability of the new methodologies have been demonstrated. For property verification, the presented methodologies were model checking, counterexample generation, application of the ASL specification to transient simulation waveforms obtained by a simulation controlled by complete state space-covering input stimuli and multi-parallel state space particle simulation.

The splitting of the verification process into a systematic ASL property specification, followed by a completely automated formal verification using the new verification algorithms, offers a structurized verification flow with minimal user interaction. Hence, not only the formality of the verification process but also its automation can be enhanced by the new methodologies.

The new trajectory-directed discretization algorithm significantly improves modeling accuracy, for the first time allowing to capture the dynamics of the state space in a discrete model with an accuracy that deviates less than 10% from transient analysis. The presented runtimes of the discretization can be considered as acceptable, due to a model checking run on a discrete model replacing a large number of conventional transient simulation runs. However, the exponential growth in states with the number of state space dimensions for generating models with a constant accuracy still limits the applicability to larger than block level circuits.

Equivalence checking using transient simulation with complete state space-covering input stimuli has proven to compare well with the state-of-the-art approach of transformed state space comparison. The wider area of application is a result of the stimuli generation being possible even for abstraction levels where the previous approach cannot be applied. This, however, comes at the cost of slightly higher verification runtimes.

Table 6.9: *Comparison of the results between equivalence checking by complete state space-covering input stimuli (stimEC) and the transformed state space comparison approach (VERA). The number of state space dimensions and the number of discrete states of the state space graph is given for the transistor netlist implementations.*

Circuit (Compared implementation)	#Dimensions	#States	Error stimEC	Runtime	Error VERA	Runtime
Biquad Bandpass (Netl. vs. VHDL-AMS)	3	2756	11%	48.13 s	1.32%	16.28 s
Delta-Sigma Modulator (Netl. vs. simple beh. model)	8	21001	10.67%	3 : 32 h	n.a.	n.a.
Log Domain Filter (Netl. vs. transfer function)	2	13866	5.73%	19.64 s	7.26%	9.49 s
Schmitt-trigger (Netl. vs. VHDL-AMS)	2	754	3.37%	18.19 s	3.51%	5.50 s
Transistor Switch (Netl. vs. beh. model)	5	320	0.88%	14.32 s	0.61%	8.47 s

Table 6.10: *Input time steps, transient simulation response time steps and runtimes of the stimEC equivalence checking approach compared to the approximated simulation runtimes of circuit comparison by systematic simulation.*

Circuit (Compared implementation)	Input time steps	Simulation time steps	stimEC	Systematic Simulation
Biquad Bandpass (Netlist vs. VHDL-AMS)	8648	144295	48.13 s	≈ 64 s
Delta-Sigma Modulator (Netlist vs. simple beh. model)	32690	468996	3 : 32 h	≈ 800 s
Log Domain Filter (Netlist vs. transfer function)	13910	65926	19.64 s	≈ 44 s
Schmitt-trigger (Netlist vs. VHDL-AMS)	4373	61337	18.19 s	≈ 44 s
Transistor Switch (Netlist vs. beh. model)	79	8322	14.32 s	≈ 258 s

7

Conclusions

Design verification of analog circuits is lacking formal methodologies in order to keep up with customer needs such as reducing the occurrence of redesigns, safety and reliability issues. Hence, this thesis has the goal of contributing new formal methodologies for analog circuit verification to target this verification gap.

7.1 Summary

The analog formal verification methodologies for nonlinear analog circuits proposed in this thesis consist of three areas:

- Discrete modeling
- Property specification
- Formal verification algorithms

To each of these areas, new approaches have been contributed in order to finally compose a set of new formal verification methodologies with the design goals of increasing accuracy and usability as well as back-propagation of formal verification results into today's test bench-based verification flows.

Discrete modeling Starting from the basics of system representation for verification, discrete modeling of analog circuits has been motivated in order to apply formal verification algorithms. The state of the art of state space discretization using a hyperbox-based state space partitioning exhibits downsides such as not being rotation invariant to the state space dynamics flow and therewith lacking accuracy and overapproximating the successor relation of the states. Hence, by analyzing these downsides, the requirements for a new discretization approach have been discussed. This resulted in the proposal of a trajectory-directed discretization algorithm, where the trajectories of the state space dynamics control the partitioning of the state space. While the geometric objects of the partitions are complex structures, a mapping to a discrete analog transition structure (DATS) was possible on a dual representation of the partitioning structure. To this discrete model, analog formal verification algorithms can be applied, delivering results of higher accuracy than previous approaches. This has been demonstrated by experimental evaluations, comparing the results of the new discretization approach to those of the hyperbox discretization and to transient analysis.

Property specification Property specification on discrete transition structures is originating from temporal logics which have been discussed in this thesis. Up to now, there have been no approaches allowing to specify complex analog properties for formal verification in the state space and only few approaches for signal-based analog specification exist. In this thesis, after analyzing the existing approaches and developing requirements for improvements, this analog property specification gap has been targeted by the development of the Analog Specification Language (ASL). ASL allows new specification methodologies for property verification in the state space of analog systems represented as a DATS. The feasibility of this model checking approach has been demonstrated in the experimental results chapter by applying ASL specifications for properties such as advanced oscillation, startup time of autonomous circuits and overshooting behavior to example circuits. Counterexamples can be generated for specification-violating states and investigated in a transient simulation test bench. Moreover, ASL specifications have been evaluated on transient simulation waveforms without modification, building a bridge for applying formal property specification to today's simulation flows in preparation of the future introduction of formal model checking.

Formal verification algorithms Besides the already mentioned model checking methodology applying ASL specifications to a DATS generated by the new trajectory-directed discretization algorithm, a set of additional formal verification methodologies has been developed in the scope of this thesis. Motivated by the introduced approach of counterexamples for analog model checking, the generation of complete state space-covering input stimuli by complete traversal of a DATS was presented. This offered to simulate analog circuit blocks in conventional transient simulation environments with guaranteed complete coverage of every reachable state the circuit can adopt. Based on the new stimuli generation algorithm, a methodology of formal assertion-based verification has been developed by evaluating ASL specifications on the waveform results obtained from transient simulation with complete state space-covering input stimuli. In addition, a formal equivalence checking methodology was introduced by comparing the transient simulation responses of two circuit implementations to such complete-coverage stimuli. The new state space particle simulation methodology augmented the verification insights gained by the introduced property verification approaches.

7.2 Challenges and Future Work

While this thesis presents approaches how the analog verification gap can be targeted by new formal verification methodologies, there are some challenges to solve until an industrial application could be considered.

Enhancing the semantics of the discrete model The trajectory-directed discrete modeling approach increased the accuracy of the state space discretization. However, although this approach reduces the number of states needed for modeling a circuit with equal discretization accuracy compared to previous approaches, the state space explosion problem still persists. Hence, a higher circuit complexity than block level could only be achieved by introducing modeling approaches that further decrease the number of states and finally end up in a sub-exponential complexity with respect to the number of state space variables. A possible approach could be giving up the homogeneity criterion for the state space partitioning and describing the flow of the state space partitions with additional semantics such as simplified polynomial or differential equations. While the partitions could be significantly larger due to more semantics put into the symbolic description of a partition, such a symbolic approach would require substantial changes to all verification algorithms operating on the enriched semantics of the discrete structure that no longer could be represented by a graph-based DATS.

Merging ASL with an established specification language The introduction of ASL showed how a minimal set of operations tailored to the purpose of analog property specification can be sufficient for complex specification tasks in the circuit's state space. Especially with the possibility of ASL specification evaluation on transient signal waveforms, a future combination with established specification approaches by adjusting the syntax for example to PSL could increase the acceptance.

Coupling with commercial simulator back-ends In the area of verification algorithms, the newly introduced verification methodologies based on complete state space-covering input stimuli generation are well suited for application to block level verification in today's non-formal verification flows without requiring the user to adapt to the up to now unfamiliar state space-based specification of analog properties. However, besides the already discussed modeling complexity, an one-click application of complete-coverage stimuli equivalence checking could be only made possible if the discrete modeling algorithms were closely coupled to commercial transient circuit simulators such as Cadence Spectre, requiring cooperation of the vendors to disclose interface internals.

Extension to mixed-signal verification While there are mature approaches to formal verification of digital systems, a challenge is posed by combining these approaches with the methodologies for the analog domain. Even on block level, there can be closely coupled feedback loops of digital and analog circuit parts, as found in delta-sigma converters or phase-locked loops. While the complexity of such blocks can still be handled by the analog discrete modeling, an abstraction of the digital circuit parts as proposed in [JH08] can further extend the applicability of the approaches presented in this thesis.

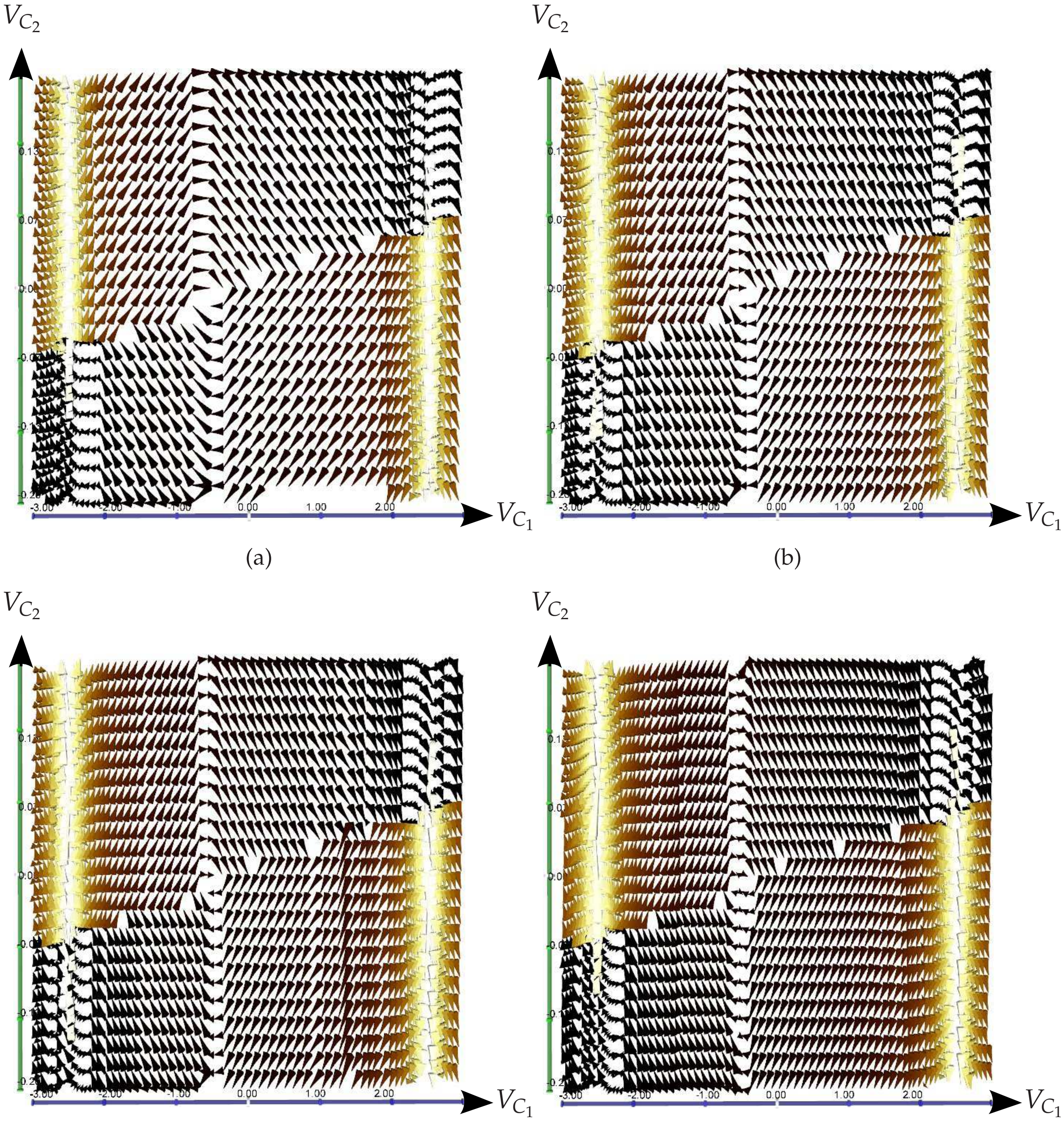
Considering parameter tolerances Although design verification primarily targets design flaws that can be identified by a discrepancy between a specified property and the implemented nominal functionality, the formal verification methodologies could be applied to verify implementations under consideration of parameter tolerances. On the one hand, this could directly be achieved by the straightforward approach of statistically varying the circuit parameters using the Monte Carlo method [MU49] in verification runs after the nominal circuit has been successfully verified. On the other hand, the parameter tolerances could be already introduced into the model to which the verification algorithms are applied using the concepts introduced in [GOB08].

Device macro-modeling using the trajectory-directed discretization Due to the increased accuracy of the discrete models generated with the trajectory-directed discretization approach, their application as device macro-models can be considered. As has been demonstrated in Section 6.3 by comparing transient analysis results to the corresponding expected waveforms calculated on the discrete model, the modeling error is very small. Hence, by implementing a wrapper that maps arbitrary input signals to a state sequence on paths and returns the sequence of state space variables of the visited states, a macro model for transient circuit analysis is given. The simulation times for piecewise linear input signals can be expected to be significantly lower than those of a transient analysis due to a simple directed graph traversal needed to determine the circuit response on the DATS model.

A

Appendix

A.1 VCO State Space Slices



(a) (b) (c) (d)

Figure A.1: *State space slices with all state transitions for input voltages* 0.66 *V (a),* 0.83 *V (b),* 1.00 *V (c) and* 1.17 *V (d) showing the adaption of the trajectory-directed discretization to the changed transition structure.*

Bibliography

[ADG07] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Inf.*, 43(7):451–476, 2007.

[Ake78] S.B. Akers. Binary decision diagrams. *IEEE Transactions on computers*, 27(6):509–516, 1978.

[And04] O. Andreassen. Visualization of vector fields using seed lic and volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):673–682, 2004. Member-Anders Helgeland.

[APT03] P.J. Ashenden, G.D. Peterson, and D.A. Teegarden. *The system designer's guide to VHDL-AMS*. Morgan Kaufmann, 2003.

[ASZT07] G. Al-Sammane, M. H. Zaki, and S. Tahar. A symbolic methodology for the verification of analog and mixed signal designs. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 249–254, San Jose, CA, USA, 2007. EDA Consortium.

[BBDE$^+$01] I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In *Computer Aided Verification*, pages 363–367. Springer, 2001.

[BCM$^+$90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, 1990.

[BCMP06] G. Bonfini, M. Chiavacci, R. Mariani, and E. Pescari. A mixed-signal verification kit for verification of analogue-digital circuits. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 88–93, 2006.

[Ben90] J. L. Bentley. K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 187–197, New York, NY, USA, 1990. ACM.

[BGG$^+$09] E. Barke, D. Grabowski, H. Graeb, L. Hedrich, S. Heinen, R. Popp, S. Steinhorst, and Y. Wang. Formal approaches to analog circuit verification. In *DATE '09: Proceedings of the conference on Design, automation and test in Europe*, pages 724–729, 2009.

[Bur01] B. Burdiek. Generation of optimum test stimuli for nonlinear analog circuits using nonlinear - programming and time-domain sensitivities. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 603–609, 2001.

[Cad07] Cadence. Incisive Enterprise Specman. 2007. URL: `http://www.cadence.com/rl/Resources/datasheets/specman_elite_ds.pdf`.

[CB95] Y.-A. Chen and R. E. Bryant. Verification of arithmetic circuits with binary moment diagrams. *Design Automation Conference*, 0:535–541, 1995.

[CE82] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, London, 1982.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model checking*. Springer, 1999.

[CHHK98] M. Chan, K.Y. Hui, C. Hu, and P.K. Ko. A robust and physical BSIM3 non-quasi-static transient and AC small-signal model for circuit simulation. *IEEE Transactions on Electron Devices*, 45(4):834–841, 1998.

[CJL$^+$97] Y. Cheng, M.C. Jeng, Z. Liu, J. Huang, M. Chan, K. Chen, P.K. Ko, and C. Hu. A physical and scalable I-V model in BSIM 3 v 3 for analog/digital circuit simulation. *IEEE Transactions on Electron Devices*, 44(2):277–287, 1997.

[CK07] H. Chang and K. Kundert. Verification of complex analog and rf ic designs. *Proceedings of the IEEE*, 95(3):622–639, 2007.

[Con04] Accellera Consortium. Property Specification Language Reference Manual. Version 1.1, 2004. URL: `http://www.eda.org/vfv/docs/PSL-v1.1.pdf`.

[CVK05] B. Cohen, S. Venkataramanan, and A. Kumari. *SystemVerilog Assertions Handbook:–for Formal and Dynamic Verification*. vhdlcohen publishing, 2005.

[DAC98] M. Dwyer, G. Avrunin, and J. Corbett. Property Specification Patterns for Finite-State Verification. In *Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP-98)*, pages 7–15, 1998.

[DC05] T. Dastidar and P. Chakrabarti. A Verification System for Transient Response of Analog Circuits Using Model Checking. In *VLSID'05: Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, pages 195–200, 2005.

[DC07] T. R. Dastidar and P. P. Chakrabarti. A verification system for transient response of analog circuits. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):1–39, 2007.

[DDM04] T. Dang, A. Donzé, and O. Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In A. J. Hu and A. K. Martin, editors, *FMCAD*, volume 3312 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2004.

[DH96] D. L. Darmofal and R. Haimes. An analysis of 3d particle path integration algorithms. *J. Comput. Phys.*, 123(1):182–195, 1996.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DL09] W. Dong and P. Li. Final-value odes: stable numerical integration and its application to parallel circuit analysis. In *ICCAD '09: Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 403–409, New York, NY, USA, 2009. ACM.

[DM07] A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. *Lecture Notes in Computer Science*, 4416:174–189, 2007.

[DN09] T. Dang and T. Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Form. Methods Syst. Des.*, 34(2):183–213, 2009.

[EF06] C. Eisner and D. Fisman. *A practical introduction to PSL*. Springer-Verlag New York Inc, 2006.

[EGG98] J. Eckmueller, M. Gropl, and H. Graeb. Hierarchical characterization of analog integrated cmos circuits. In *Proceedings of the conference on Design, automation and test in Europe*, pages 636–643. Citeseer, 1998.

[EMSS92] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Syst.*, 4(4):331–352, 1992.

[ES00] D. Estévez Schwarz. *Consistent initialization for index-2 differential algebraic equations and its application to circuit simulation*. PhD thesis, Humboldt-Univ. Berlin, 2000.

[FKR06] G. Frehse, B. Krogh, and R. Rutenbar. Verifying Analog Oscillator Circuits Using Forward/Backward Abstraction Refinement. In *DATE 2006: Design, Automation and Test in Europe*, 2006.

[FMW05] H. Foster, E. Marschner, and Y. Wolfsthal. IEEE 1850 PSL: The Next Generation. 2005. URL: `http://www.pslsugar.org/papers/ieee1850psl-the_next_generation.pdf`.

[FSS06] M. Freibothe, J. Schönherr, and B. Straube. Formal verification of the quasi-static behavior of mixed-signal circuits by property checking. *Electr. Notes Theor. Comput. Sci.*, 153(3):23–35, 2006.

[GDWL92] D. D. Gajski, N. D. Dutt, Allen C.-H. Wu, and S. Y.-L. Lin. *High-level synthesis: introduction to chip and system design*. Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, USA, 1979.

[GKR04] S. Gupta, B. H. Krogh, and R. A. Rutenbar. Towards formal verification of analog designs. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 210–217, Washington, DC, USA, 2004. IEEE Computer Society.

[GOB08] D. Grabowski, M. Olbrich, and E. Barke. Analog circuit simulation using range arithmetics. In *ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation*, pages 762–767, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

[GPHB05] D. Grabowski, D. Platte, L. Hedrich, and E. Barke. Time Constrained Verification of Analog Circuits using Model-Checking Algorithms. In *FAC 2005: Proceedings of the First Workshop on Formal Verification of Analog Circuits*, pages 37–52, 2005.

[GRW05] J. Goss, W. Roesner, and B. Wile. *Comprehensive Functional Verification*. Morgan Kaufmann Publishers, 2005.

[GV99] A. Ghosh and R. Vemuri. Formal verification of synthesized analog designs. In *ICCD '99: Proceedings of the 1999 IEEE International Conference on Computer Design*, page 40, Washington, DC, USA, 1999. IEEE Computer Society.

[GVL96] G.H. Golub and C.F. Van Loan. *Matrix computations*. Johns Hopkins Univ Pr, 1996.

[GY08] M. R. Greenstreet and S. Yang. Verifying start-up conditions for a ring oscillator. In *GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 201–206, New York, NY, USA, 2008. ACM.

[HA92] M. Godau H. Alt. Measuring the resemblace of polygonal curves. In *Proc. of the 8th Annual Symposium on Computational Geometry*, pages 102–109, 1992.

[Har03] W. Hartong. *Ansätze zum Model-Checking nichtlinearer analoger Systeme*. Fortschritt-Berichte VDI, Reihe 20 Nr. 364. VDI Verlag GmbH, Duesseldorf, 2003.

[HB95] L. Hedrich and E. Barke. A formal approach to nonlinear analog circuit verification. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 123–127, Washington, DC, USA, 1995. IEEE Computer Society.

[HB98] L. Hedrich and E. Barke. A formal approach to verification of linear analog circuits wth parameter tolerances. In *DATE '98: Proceedings of the conference on Design, automation and test in Europe*, pages 649–655, Washington, DC, USA, 1998. IEEE Computer Society.

[HBKK94] B.J. Hosticka, W. Brockherde, R. Klinke, and R. Kokozinski. Design methodology for analog monolithic circuits. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, pages 387–394, 1994.

[HGT91] S.A. Huss, M. Gerbershagen, and G. Traenkle. Automatic performance characterization of analog functional blocks. *Analog Integrated Circuits and Signal Processing*, 1(4):277–286, 1991.

[HH95] Henzinger, T. A. and Ho, P. -H. . Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939, pages 225–238, Liege, Belgium, 1995. Springer Verlag.

[HHB02a] W. Hartong, L. Hedrich, and E. Barke. Model checking algorithms for analog verification. In *Proceedings of the 39th conference on Design automation (DAC '02)*, pages 542–547, 2002.

[HHB02b] W. Hartong, L. Hedrich, and E. Barke. On discrete modeling and model checking for nonlinear analog systems. In *CAV '02: Proceedings of the 14th*

International Conference on Computer Aided Verification, pages 401–413, London, UK, 2002. Springer-Verlag.

[HKH04] W. Hartong, R. Klausen, and L. Hedrich. Formal verification for nonlinear analog systems: Approaches to model and equivalence checking. 2004.

[HLSS08] W. Hartong, N. Luetke-Steinhorst, and R. Schweiger. Coverage Driven Verification for Mixed Signal Systems. *GMM-Fachbericht-ANALOG'08*, 2008.

[Hol99] A. Holt. Formal verification with natural language specifications: guidelines, experiments and lessons so far. *South African Computer Journal*, 24:253–257, 1999.

[HRB75] C.W. Ho, A.E. Ruehli, and P.A. Brennan. The Modified Nodal Approach to Network Analysis. *IEEE Transactions on Circuits and Systems*, 22(6):504–509, 1975.

[JH08] A. Jesser and L. Hedrich. A Symbolic Approach for Mixed-Signal Model Checking. In *Proceedings of the 13th Asia and South Pacific Design Automation Conference (ASP-DAC'08)*, pages 404–409, COEX, Seoul, Korea, January 2008.

[JKK08] K. D. Jones, J. Kim, and V. Konrad. Some 'real world' problems in the analog and mixed signal domains. In Gordon J. Pace and Satnam Singh, editors, *Seventh International Workshop on Designing Correct Circuits: Budapest, 29–30 March 2008: Participants' Proceedings*, pages 15–29. ETAPS 2008, March 2008. A Satellite Event of the ETAPS 2008 group of conferences.

[JKN10] K. D. Jones, V. Konrad, and D. Ničković. Analog property checkers: a ddr2 case study. *Form. Methods Syst. Des.*, 36(2):114–130, 2010.

[KB05] R.H. Katz and G. Borriello. Contemporary logic design. 2005.

[Kir47] G. Kirchhoff. Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

[KM91] R. P. Kurshan and K. L. McMillan. Analysis of digital circuits through symbolic reduction. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 10(11):1356–1371, 1991.

[KZ04] K.S. Kundert and O. Zinke. *The designer's guide to Verilog-AMS*. Springer, 2004.

[Lam00] A. van Lamsweerde. Formal specification: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 147–159, New York, NY, USA, 2000. ACM.

[LSW$^+$06] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda. Verification of analog/mixed-signal circuits using labeled hybrid petri nets. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 275–282, New York, NY, USA, 2006. ACM.

[Mas83] W. S. Massey. Cross products of vectors in higher dimensional euclidean spaces. *The American Mathematical Monthly*, 90(10):697–701, 1983.

[McA93] D. F. McAllister, editor. *Stereo computer graphics: and other true 3D technologies*. Princeton University Press, Princeton, NJ, USA, 1993.

[McM92] K. L. McMillan. *Symbolic model checking: an approach to the state explosion problem*. PhD thesis, Pittsburgh, PA, USA, 1992.

[Men01] E. Mendelson. *Introduction to mathematical logic*. Chapman & Hall/CRC, 2001.

[MHW$^+$06] C. J. Myers, R. R. Harrison, D. Walter, N. Seegmiller, and S. Little. The case for analog circuit verification. *Electr. Notes Theor. Comput. Sci.*, 153(3):53–63, 2006.

[Mir95] R. Miranda. *Algebraic curves and Riemann surfaces*. Amer Mathematical Society, 1995.

[MM04] P. Molitor and J. Mohnke. *Equivalence checking of digital circuits: fundamentals, principles, methods*. Kluwer Academic Pub, 2004.

[MNP08] O. Maler, D. Nickovic, and A. Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In A. Avron, N. Dershowitz, and A. Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 475–505. Springer, 2008.

[MPDG09] R. Mukhopadhyay, S. K. Panda, P. Dasgupta, and J. Gough. Instrumenting ams assertion verification on commercial platforms. *ACM Trans. Des. Autom. Electron. Syst.*, 14(2):1–47, 2009.

[MU49] N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[MZXA08] Hong-Guang Ma, Xiao-Fei Zhu, Jian-Feng Xu, and Ming-Shun Ai. Circuit state analysis using chaotic signal excitation. *Journal of the Franklin Institute*, 345(1):75 – 86, 2008.

[Nag75] L. W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. PhD thesis, EECS Department, University of California, Berkeley, 1975.

[NM07] D. Nickovic and O. Maler. Amt: A property-based monitoring tool for analog systems. In J.-F. Raskin and P. S. Thiagarajan, editors, *FORMATS*, volume 4763 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 2007.

[PHHB98] R. Popp, W. Hartong, L. Hedrich, and E. Barke. Error estimation on symbolic behavioral models of nonlinear analog circuits. In *SMACD '98: Proceedings of the 5th International Conference on Symbolic Methods and Applications to Circuit Design*, pages 223–226, 1998.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57, 1977.

[Rap90] J. Raphson. Analysis Aequationum Universalis Seu Ad Aequationes Algebraicas Resolvendas Methodus Generalis, et Expedita, Ex nova Innitarum Serierum Doctrina, Deducta Ac Demonstrata, London. *Original in British Library, London*, 1690.

[SA01] S. Seshadri and J. A. Abraham. Frequency response verification of analog circuits using global optimization techniques. *J. Electron. Test.*, 17(5):395–408, 2001.

[SH08a] S. Steinhorst and L. Hedrich. A formal approach to complete state space-covering input stimuli generation for verification of analog systems. In *Analog 2008: 10. ITG/GMM-Fachtagung Entwicklung von Analogschaltungen mit CAE-Methoden*, 2008.

[SH08b] S. Steinhorst and L. Hedrich. Model Checking of Analog Systems using an Analog Specification Language. In *Proc. of the Conference on Design, Automation and Test in Europe 2008 (DATE'08)*, pages 3247–329, 2008.

[SH09] S. Steinhorst and L. Hedrich. Joint property specification for transient simulation and formal verification of analog circuits. In *Proc. of the edaWorkshop'09*, pages 13–18, Berlin, 2009. VDE Verlag.

[SH10a] S. Steinhorst and L. Hedrich. Advanced methods for equivalence checking of analog circuits with strong nonlinearities. *Formal Methods in System Design*, 36(2):131–147, 2010.

[SH10b] S. Steinhorst and L. Hedrich. Improving verification coverage of analog circuit blocks by state space-guided transient simulation. In *Circuits and Systems, 2010. ISCAS 2010. IEEE International Symposium on*, May 2010.

[SHZ$^+$01] M. Soma, S. Huynh, J. Zhang, S. Kim, and G. Devarayanadurg. Hierarchical ATPG for Analog Circuits and Systems. *IEEE Des. Test*, 18(1):72–81, 2001.

[SJH06] S. Steinhorst, A. Jesser, and L. Hedrich. Advanced Property Specification for Model Checking of Analog Systems. In *Analog 2006: 9. ITG/GMM-Fachtagung Entwicklung von Analogschaltungen mit CAE-Methoden*, pages 63–68, 2006.

[SPH09] S. Steinhorst, M. Peter, and L. Hedrich. State space exploration of analog circuits by visualized multi-parallel particle simulation. In *ICSPS '09: Proceedings of the 2009 International Conference on Signal Processing Systems*, pages 858–862, Washington, DC, USA, 2009. IEEE Computer Society.

[ST00] D.E. Schwarz and C. Tischendorf. Structural analysis of electric circuits and consequences for MNA. *International Journal of Circuit Theory and Applications*, 28(2):131–162, 2000.

[Syn03] I. Synopsys. OpenVera Language Reference Manual, 2003.

[SZDT07] G. Al Sammane, M. H. Zaki, Z. J. Dong, and S. Tahar. Towards assertion based verification of analog and mixed signal designs using psl. In *FDL*, pages 293–298. ECSI, 2007.

[TGP$^+$09] S. K. Tiwary, A. Gupta, J. R. Phillips, C. Pinello, and R. Zlatanovici. First steps towards sat-based formal analog verification. In *ICCAD '09: Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 1–8, New York, NY, USA, 2009. ACM.

[Tis96] C. Tischendorf. *Solution of index-2-DAEs and its application in circuit simulation*. PhD thesis, Humboldt-Univ. Berlin, 1996.

[Tsi03] Y. Tsividis. *Operation and modeling of the MOS transistor*. Oxford University Press, USA, 2003.

[VdPG97] W. Verhaegen, G. Van der Plas, and G. Gielen. Automated Test Pattern Generation for Analog Integrated Circuits. In *VTS '97: Proceedings of the 15th IEEE VLSI Test Symposium (VTS'97)*, pages 296–301, 1997.

[Vla94] A. Vladimirescu. *The SPICE book*. John Wiley & Sons, Inc. New York, NY, USA, 1994.

[Whi95] J.E. Whitesitt. *Boolean algebra and its applications*. Dover Pubns, 1995.

[WJWH09] Y. Wang, S. Joeres, R. Wunderlich, and S. Heinen. Modeling approaches for functional verification of rf-socs: limits and future requirements. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28(5):769–773, 2009.

[Ypm95] T.J. Ypma. Historical development of the Newton-Raphson method. *SIAM review*, 37(4):531–551, 1995.

Lebenslauf

Sebastian Steinhorst
geboren am 07. Februar 1980 in Mainz

Ausbildung

2006 - 2010	Promotionsstudium im Fachgebiet Informatik an der Goethe-Universität in Frankfurt am Main.
2000 - 2005	Studium der Informatik mit Nebenfach Betriebswirtschaftslehre an der Goethe-Universität in Frankfurt am Main. Abschluss als Diplom-Informatiker im Dezember 2005. Diplomarbeit an der Professur für Entwurfsmethodik: „Entwicklung einer Spezifikationssprache für das Model Checking von Mixed-Signal Systemen“. (Betreuer: Prof. Dr.-Ing. Lars Hedrich).
1990 - 1999	Besuch des Gymnasialen Zweiges der Kopernikusschule Freigericht, Abschluss Abitur (Allgemeine Hochschulreife) im Juni 1999.

Berufliche Tätigkeiten

seit März 2006	Wissenschaftlicher Mitarbeiter an der Professur für Entwurfsmethodik (Prof. Dr.-Ing. Lars Hedrich) an der Goethe-Universität in Frankfurt am Main.