TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Institut für Grundlagen der Elektrotechnik und Elektronik

Studienarbeit

Thema: Charakterisierung und Modellierung von Analogschaltungen

Vorgelegt von: Annegret Plänitz

Matrikelnummer: 2921798

Studiengang: Informationssystemtechnik

Betreuer: Dr.-Ing. Jan Müller,
Dipl.-Ing. Roland Jancke

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Wolfgang Schwarz

Tag der Einreichung: 16.03.2006

Fraunhofer Institut
Integrierte Schaltungen

Fraunhofer-Institut IIS Zeunerstraße 38 D-01069 Dresden

Außenstelle Entwurfsautomatisierung
Zeunerstraße 38
01069 Dresden

Leitung
Prof. Dr.-Ing. Günter Elst
Telefon +49 (0) 351 / 4640-701
Telefax +49 (0) 351 / 4640-703
www.eas.iis.fraunhofer.de

Roland Jancke
Telefon +49 (0) 351 / 4640-748
Roland.Jancke@eas.iis.fraunhofer.de

Dresden, 17. April 2006

Aufgabenstellung für die Studienarbeit von Frau Annegret Plänitz

Technischer Hintergrund:

Entwurf und Fertigung komplexer elektronischer Schaltungen sind sehr kostspielig. Daher wird nach Möglichkeiten gesucht, bereits vor der eigentlichen Fertigung die Richtigkeit des Entwurf sicherzustellen. Dazu bedient man sich häufig der Modellierung und Simulation am Rechner. Im Zuge der Entwurfsautomatisierung muss zunehmend auch der Nachweis eines korrekten Entwurfs mittels Modellierung und Simulation automatisiert werden.

Mit Hilfe von Simulatorskripten lassen sich wichtige Eigenschaften einer entworfenen Schaltung ermitteln: Ein- und Ausgangsimpedanzen, Übertragungsfunktion, Abhängigkeit von der Betriebsspannung, usw. Dieser Prozess wird Charakterisierung genannt. Eine skriptbasierte Charakterisierungsumgebung kann viele Eigenschaften einer ganzen Reihe von Entwurfsalternativen für eine Schaltung automatisch bestimmen.

Im Ergebnis der Charakterisierung entstehen Datenblätter zu den Schaltungen oder Parameter für ein mathematisches Modell der Schaltung. Mit einem solchen Modell kann das Zusammenwirken der Schaltung mit ihrer späteren Einsatzumgebung in der Simulation getestet werden.

Bei der durchzuführenden Arbeit sind die Eigenschaften von Transistorschaltungen per Simulation zu ermitteln und vorgegebene Modelle damit zu parametrisieren. Da es sich um eine Vielzahl von Schaltungen handelt, ist das Ziel, der Aufbau einer Charakterisierungsumgebung auf der Basis von Skripten zur Simulatorsteuerung.

Aufgabenstellung Praktikum / Studienarbeit

Charakterisierung und Modellierung von Analogschaltungen

1 Einarbeitung in die notwendige Softwareumgebung:

Analogsimulation mit SPICE, ADVance MS

Skripterstellung mit Tcl

2 Ermittlung charakteristischer Eigenschaften von Transistorschaltungen

3 Aufbau einer skriptbasierten Charakterisierungsumgebung

4 Parametrisierung und Generierung von Modellen der Schaltungen

5 Dokumentation der durchgeführten Arbeiten und erstellten Tools

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage dem Prüfungsausschuss der Fakultät Elektrotechnik eingereichte Studienarbeit zum Thema

„Charakterisierung und Modellierung von Analogschaltungen“

vollkommen selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den

Unterschrift

Kurzfassung

Elektronische Schaltungen werden durch die rasche Entwicklung immer kompakter, kleiner und komplizierter. Zur Bewältigung der hohen Anforderungen an heutige Schaltungsentwickler werden zunehmend Schaltungsmodelle eingesetzt. Mit Hilfe solcher Modelle kann zum Beispiel die Richtigkeit entworfener Schaltkreise einfach überprüft werden.

Die nachfolgende Arbeit beschäftigt sich nun mit der Erstellung von Modellen für bestimmte Schaltungsklassen. Dazu werden charakteristische Eigenschaften von Operationsverstärkern per Simulation bestimmt. Entsprechende Messschaltungen und Berechnungsvorschriften werden hergeleitet und vorgestellt. Aus den charakteristischen Eigenschaften wird das Modell der Schaltung erstellt.

Zur Unterstützung dieses Prozesses wird ein Programm entwickelt, welches automatisiert die Operationsverstärker charakterisiert und deren Modelle erstellt. Das Programm ist so ausgelegt, das neben dem Operationsverstärker weitere Schaltungsklassen ergänzt werden können. Die Anwendung selber und deren Entwicklung wird in dieser Arbeit vorgestellt.

Summary

Due to a rapid design and development electronic circuits are becoming more compact, smaller and more complicated. To manage the high requirements today’s developer of electronic circuits increase the usage of models. Via those models the correctness of the designed circuits can easily be verified.

The following essay discusses the generation of models for specific classes of circuits. For that purpose characteristical features of operational amplifier are determined by computer simulation. Adequate measuring circuits and calculation algorithms will be pointed out and affiliated. In connection with the characteristical features the model of the circuit is developed.

To assist this process a special software is designed to characterise automatically the operational amplifiers as well as to generate its models. Beside the operational amplifier the software also is able to add further classes of electronic circuits. The application itself as well as its development will be illustrated in this essay.

Inhaltsverzeichnis

1. Einleitung

1.1. Einführung

Die Aufgabenstellung „Charakterisierung und Modellierung analoger Schaltungen“ entstand am Fraunhofer Institut für Integrierte Schaltungen, Außenstelle Entwurfsautomatisierung. Werkzeuge und Methoden im analogen Schaltungsentwurf sind heutzutage im Vergleich zum digitalen Schaltungsentwurf noch unausgereift und fehleranfällig. Das Institut forscht im Rahmen des Projektes SAMS – Struktursynthese analoger Schaltungen – nach neuen Möglichkeiten zur Automatisierung und Vereinfachung des analogen Schaltungsentwurfs.
Dazu gehört unter anderem die Automatisierung der Modellierung analoger Schaltungen. Modelle werden zur Simulation und Verifikation von Schaltungen benötigt. Sie sollen die enorm hohen Simulationszeiten der analogen Originalschaltungen verkürzen und den Schaltungsentwurf somit beschleunigen.

Modellierungswerkzeuge für analoge Schaltungen existieren bereits. Es gibt verschiedene Verfahren mit unterschiedlichen Vor- und Nachteilen zur Modellgenerierung. Um mit der Entwicklung Schritt zu halten und unabhängig zu sein, soll ein eigenes Werkzeug entwickelt werden, welches als Basis für spätere Erweiterungen im Bereich Entwurfsautomatisierung dient.

Die Aufgabe besteht in der Entwicklung eines Werkzeuges zur Modellierung analoger Schaltungen. Dafür soll die Methode parametrisierbarer Modelle verwendet werden. Im Zuge des Projektes SAMS wurde bereits eine Vielzahl von Operationsverstärkerschaltungen automatisch generiert. Ziel ist es, diese dann mit dem Werkzeug zu modellieren.
Weiterhin sollen Möglichkeiten zur Anbindung anderer Modellierungsmethoden, anderer Simulatoren und Skriptsprachen etc. berücksichtigt werden.

1.2. Inhalt

Um Begriffe wie Charakterisierung, Modellierung etc. zu erklären und Ziele entsprechender Techniken zu erläutern, beschäftigt sich Kapitel *2* mit dem technischen Hintergrund der Schaltungsmodellierung. Dazu werden aktuelle Methoden, Werkzeuge und Probleme im Bereich analoger Schaltungsentwicklung behandelt. Zusätzlich soll dieses Kapitel aufzeigen, wie wichtig die Forschung im Bereich der Entwurfsautomatisierung ist.

Die Aufgabe lässt sich grob in zwei Wissensbereiche gliedern. Auf der einen Seite befindet sich die Elektrotechnik, im Besonderen Methoden des Schaltungsentwurfes sowie Grundlagen und Simulation analoger Schaltungen, speziell Operationsverstärker. Auf der anderen Seite steht die Informatik im Sinne eines übersichtlichen, durchdachten Softwareentwurfes zur Erstellung einer flexiblen, erweiterbaren Anwendung. Entsprechend zeigen Kapitel *3* und *4* für jeden Bereich verschiedene Lösungsansätze einschließlich der ausgewählten Lösungswege. Dabei behandelt Kapitel *3* die *Charakterisierung und Modellierung von Operationsverstärkern* und Kapitel *4* den *Softwareentwurf*.

Kapitel *5* enthält eine allgemeine *Zusammenfassung*. In Kapitel *6* befindet sich das *Literaturverzeichnis* und Kapitel *7* beinhaltet das *Symbolverzeichnis*. Kapitel *8* zeigt das *Bild- und Tabellenverzeichnis*.

1.3. Projektplan

Der nachfolgende Projektplan zeigt chronologisch das Vorgehen zur Lösung der Aufgabenstellung. Erkenntnisse und Ergebnisse der durchgeführten Einzelaufgaben sind in den Kapiteln *3* und *4* erläutert.

Charakterisierung und Modellierung von Analogschaltungen

I) Einarbeitung in die notwendige Softwareumgebung

- Literaturstudium
- Meilensteinberichte SAMS-Projekt
- externe Literatur zur Theorie
- Dokumentation Simulatoren
- Simulatoren: SPICE, Eldo, ADVance MS, SystemVision
- Postprocessing-Tools: EZWave
- Steuersprachen: Tcl/TK, Java

II) Simulationsexperimente mit Eldo

- SPICE-Simulator von Mentor Graphics
- Einrichtung der Umgebung
- Erstellung und Simulation einfacher Netzlisten
- Anzeigen und Auswerten der Ergebnisse mit EZWave

III) Simulationsexperimente mit ADVance MS

- Verhaltenssimulator von Mentor Graphics
- Einrichtung der Umgebung
- Simulation vorhandener Verhaltensmodelle
- Anzeigen und Auswerten der Ergebnisse mit EZWave

IV) Experimente zur Simulatorsteuerung mit Tcl

- vorhandene Skripte verstehen, nachvollziehen, abwandeln, ...
- Steuerung von Simulator und Postprocessing
- Variation der Messschaltungen
- Vergleich mit Möglichkeiten der Simulatorsteuerung durch Java
- Ermittlung charakteristischer Eigenschaften von Transistorschaltungen

V) Simulation einzelner Schaltungsstrukturen

- Auswahl von Schaltungen aus Menge der synthetisierten Strukturen
- Simulation im Zeit- und Frequenzbereich
- Zusammenstellung der notwendigen Messschaltungen

VI) Extraktion wichtiger Eigenschaften der Schaltung

- Berechnung der Schaltungseigenschaften im Zeit- und Frequenzbereich:
    - Portimpedanzen: Rin, Rout, Cin, Cout; allgemein: Zin, Zout
    - Verstärkung (DC und frequenzabhängig), Amplituden- und Phasengang
    - 3dB-Frequenz, Pole, Nullstellen höherer Ordnung
    - Ausgangsbegrenzung für Spannung und Strom
    - Eingangsoffsetspannung
    - Gleichtaktverstärkung
    - Betriebsspannungsunterdrückung
    - …
- Zusammenstellung der Berechnungsvorschriften

VII) Aufbau einer skriptbasierten Simulationsumgebung

- Festlegung des Softwarekonzepts
- Skriptbasierte Steuerung von Simulator und Postprocessing-Tool
- Programmierung einer gemeinsamen Oberfläche
- Java-basierte oder Tcl/TK basierte Softwareumgebung?
- Softwareerstellung mit Eclipse?
- Kriterien
    - Erweiterbarkeit für weitere Simulatoren, Verhaltensbeschreibungssprachen
    - Einbindung anderer Modellierungsmethoden (numerisch, symbolisch, ...)
    - Zukunftsträchtigkeit hinsichtlich Standardisierung, Verbreitung, Stand der Technik
    - verfügbare Entwicklungsumgebung

- Festlegung des inhaltlichen Umfangs
    - Schaltungsklassen
    - Messschaltungen
    - Automatisch charakterisierbare Eigenschaften
    - Vorbereitung der Modellvorlagen
- Implementierung der Software
    - Erstellung der Skripte
    - Programmierung einer Oberfläche
    - Test

VIII) Dokumentation der durchgeführten Arbeiten und erstellten Tools

- Sammlung von Erfahrungen, einschließlich nicht erfolgreicher Versuche
- Softwaredokumentation zur nachfolgenden Nutzung am Institut
- Darstellung der Arbeiten und Ergebnisse im Rahmen der Studienarbeit

2. Technischer Hintergrund

2.1. Aktuelle Entwicklung

2.1.1 Steigende Komplexität

Die Komplexität bei der Entwicklung von Mikroelektronik, zum Beispiel bei anwendungsspezifischen integrierten Schaltungen (ASICs – Application-specific ICs) oder anwendungsspezifischen Standardschaltkreisen (ASSPs – Application-specific Standard Parts), steigt stetig. Neue Technologien, wie Ultra-Deep-Submicron und Nanometer-CMOS, ermöglichen immer kleinere Strukturbreiten, höhere Integrationsdichten und damit komplexere Schaltungen.
Mittlerweile finden komplette Systeme auf einem einzigen Chip Platz. Dabei sind mehrere Prozessorenkerne, anwendungsspezifische Schaltungen, Speicher und Peripheriebausteine auf einem Chip untergebracht. Diese so genannten „Systems on a Chip“ (SoC) werden größtenteils in der Telekommunikation und im Multimediabereich verwendet.

Durch die rasante Entwicklung stehen die Hersteller enorm unter Zeitdruck. Sie müssen jederzeit Schritt halten mit der Entwicklung neuer Schaltungen. Zusätzlich treten durch die immer stärkere Integration neue störende Effekte auf, die beim Entwurfsprozess berücksichtigt werden müssen. Das widerspricht dem Wunsch nach einem schnellen und kostengünstigen Schaltungsentwurf. Es entsteht zunehmend eine Diskrepanz zwischen Schaltungskomplexität und Entwurfsproduktivität.
Aus diesen Gründen wird zunehmend nach Lösungen gesucht, den Entwurfsprozess zu verbessern. Man forscht nach neuen Vorgehensweisen, um effektive Schaltungen zu entwickeln. Gleichzeitig wird nach Werkzeugen für den rechnergestützten Entwurf und zur Schaltungsverifikation gesucht.

2.1.2 Mixed-Signal-Schaltkreise

Die meisten Funktionen in heutigen Systemen sind digital realisiert, zum Beispiel durch digitale Signal Prozessoren (DSP). Einige Funktionen allerdings müssen analog implementiert werden. Das sind die Schnittstellen zwischen dem elektronischen System und der wirklichen Welt. Folgende Beispiele zeigen typische Funktionen, die auch in Zukunft vorrausichtlich immer analog realisiert sein werden:

- Systemeingabe, zum Beispiel Signale von Sensoren, Mikrophonen und Antennen; realisiert durch zum Beispiel Mischer, Oszillatoren, Verstärker, Filter
- Systemausgabe, zum Beispiel Audio, Video; realisiert durch zum Beispiel Filter, Oszillatoren, Mischer
- Wandlerschaltkreise, zum Beispiel Sample-and-Hold-Schaltungen zur Abtastung, Analog/Digital Wandler
- Stark spezialisierte Schaltungen, d.h. Schaltungen mit speziellen Eigenschaften, wie zum Beispiel sehr schnelle Schaltungen mit geringem Energieverbrauch

Analoge Schaltungen sind somit in einem Großteil aller elektronischen Anwendungen enthalten. Um Kosten zu sparen und Performanz zu steigern, werden analoge Teilschaltungen heute meist in digitale Schaltungen integriert. Das heißt, ein Chip enthält sowohl analoge als auch digitale Komponenten. Diese Schaltungen werden Mixed-Signal-Schaltkreise genannt. Mixed-Signal-Schaltkreise finden einen immer größer werdenden Markt, zum Beispiel in der Telekommunikation oder in der Biomedizin. Seit 1990 beträgt die mittlere Wachstumsrate für Mixed-Signal-Schaltkreise jährlich 15-20% [1].

2.1.3 CAD-Werkzeuge

CAD-Werkzeuge (Computer Aided Design) unterstützen den Entwurfsprozess. Sie bieten Strukturierungsmöglichkeiten zur Reduktion der Komplexität und Automatisierungsfunktionen bei Routine- oder wiederholbaren Aufgaben. CAD-Werkzeuge erhöhen somit die Produktivität von Schaltungsentwicklern und reduzieren anfallende Kosten.

Für das digitale Einsatzgebiet existieren bereits viele dieser CAD-Werkzeuge. Das liegt daran, dass der Markt für digitale integrierte Schaltungen sehr groß ist. Außerdem können digitale Systeme – im Gegensatz zu analogen Systemen – einfach durch boolesche Ausdrücke und Programmiersprachen repräsentiert werden. Viele Prozesse in den unteren Ebenen des Entwurfs sind dadurch automatisiert. So gibt es Hardwarebeschreibungssprachen, wie VHDL und Verilog, Synthesewerkzeuge, welche eine strukturelle Repräsentation aus der Verhaltensbeschreibung erstellen, und Logiksynthesewerkzeuge, welche die strukturelle Spezifikation in eine Gate-Level-Netzliste übersetzen. Schließlich unterstützen Layoutwerkzeuge das Platzieren und Routen der Netzliste, basierend auf einer ausgewählten Technologie und einer

speziellen Zellbibliothek. Große Teile des Entwurfsprozesses digitaler Schaltungen werden dadurch bereits gut unterstützt.

Im Gegensatz zum digitalen Schaltungsentwurf ist der Entwurf analoger Schaltkreise deutlich komplizierter. Analoge Schaltungen haben sehr viele Abhängigkeiten und viele Nebeneffekte. Darum werden auch heute noch Spezialwissen und Schaltungsentwurfsfähigkeiten aus jahrelanger Erfahrung benötigt. Trotzdem gibt es bereits einige gute Werkzeuge, mit denen das Verhalten analoger Schaltungen simuliert werden kann. So wird seit Jahrzehnten bereits der analoge Schaltungssimulator SPICE eingesetzt, der mit Hilfe einer Komponentenbibliothek aus Widerständen, Quellen, Kondensatoren etc. Spannungen und Ströme analoger Schaltungen numerisch berechnet.
Doch aufgrund der Komplexität analoger Schaltungen stoßen diese Werkzeuge bei sehr großen Schaltungen an ihre Grenzen. Insbesondere beim Entwurf von Mixed-Signal-Schaltkreisen können SPICE-ähnliche analoge Simulatoren kaum noch verwendet werden, da sie teilweise Tage bis Wochen zur Simulation benötigen. Das führt dazu, dass der Entwurf der analogen Bauteile in Mixed-Signal-Schaltkreisen einen Engpass im kompletten Entwurfsprozess bildet, obwohl diese nur einen kleinen Teil aller Bauteile ausmachen.

2.1.4 Entwurf von Mixed-Signal-Schaltkreisen

Die Mixed-Signal-Schaltkreise werden im Idealfall nach einem Top-Down-Prinzip entworfen, dargestellt in *Abbildung 1*.
Am Anfang steht die Idee. Danach erfolgt ein grober Systementwurf. Dieser wird in einzelne Funktionsblöcke unterteilt und zunehmend konkretisiert, bis ein Schaltkreisentwurf auf Transistorebene entstanden ist. Zusätzlich zu dem Top-Down-Prinzip erfolgt eine Bottom-Up-orientierte Verifikationsphase.
Die Verifikation erfolgt beginnend mit der Simulation der Teilsysteme auf unterster Ebene, aufsteigend durch Zusammensetzen der Teilsysteme, bis hin zur Simulation des Gesamtsystems. Hierbei wird der Entwurf simuliert, getestet und korrigiert. Die Verifikation ist dadurch eng an den Entwurfsprozess gekoppelt.

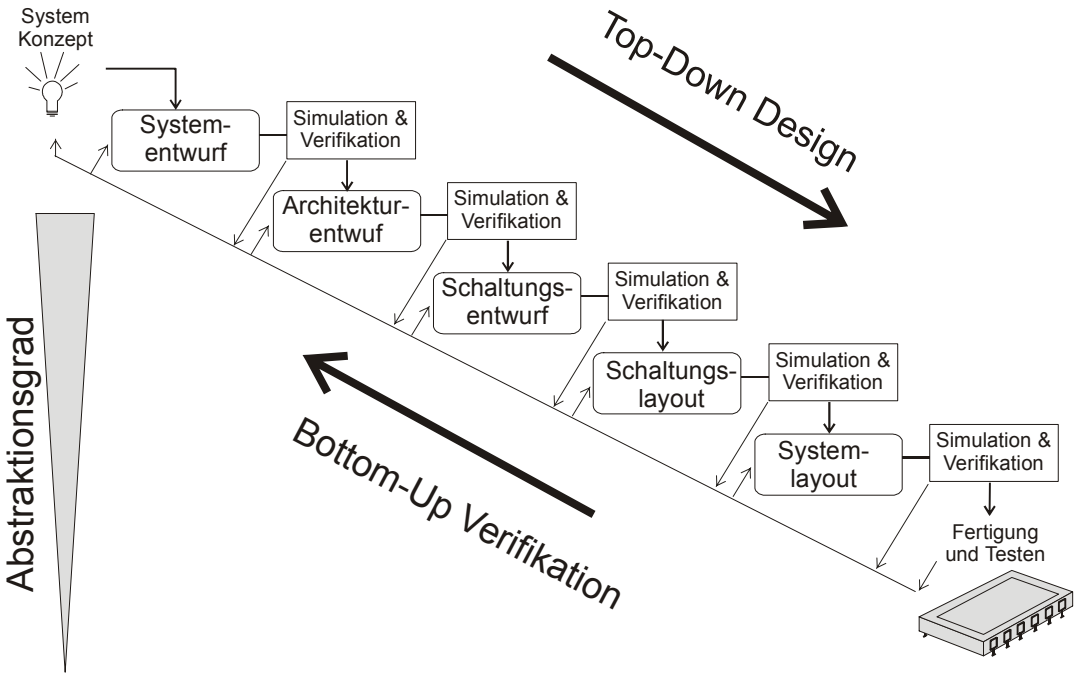

Abbildung 1 – Entwurfsprozess von Mixed-Signal-Schaltkreisen [1]

Verifikation ist ein sehr wichtiger Prozess im Schaltungsentwurf. Die exakte Verifikation ist deshalb so notwendig, weil die Chip-Produktion sehr teuer ist. Es muss sichergestellt sein, dass ein Design komplett funktionstüchtig ist und allen Anforderungen entspricht, bevor es in die Produktion geht. Die frühen Testphasen im Entwurfsprozess ermöglichen es, Fehler rechtzeitig zu entdecken und zu korrigieren. So wird der Zeitaufwand im Schaltkreisentwurf minimiert.

Doch Simulation und Verifikation der Gesamtsysteme sind teilweise nur schwer umsetzbar. So können Schaltungen mit analogen Anteilen nicht mittels digitaler Werkzeuge simuliert werden. Analogsimulatoren sind zu langsam, wenn diese die digitalen Anteile auf Transistorebene simulieren.

Zur Behebung des vorstehend genannten Problems werden so genannte Mixed-Mode-Simulatoren entwickelt, welche Schaltungen mit digitalem und analogem Anteil simulieren können. Zur Einsparung von Simulationszeit werden komplexe Teilkomponenten des Systems durch abstrakte Verhaltensmodelle ersetzt. Mit den Modellen wird das Verhalten dieser Teilkomponenten im Gesamtsystem verifiziert.

2.2. Schaltungsmodellierung

2.2.1 Abstraktion durch Modelle

Die in dieser Arbeit betrachteten Modelle sind keine Großsignal- oder Kleinsignalmodelle, wie sie zur Berechnung analoger Schaltungen verwendet werden. Es werden nicht nur einzelne Bauteile einer Schaltung durch Modelle ersetzt, sondern es soll ein neues Modell eines komplexen Schaltungsteils erstellt werden. Dabei bilden die erstellten Modelle nur das Ein- und Ausgabeverhalten einer Schaltung nach. Der innere Schaltungsaufbau wird nicht berücksichtigt. Solche Modelle bieten folgende Vorteile:

- Durch die Verwendung einfacher Modelle kann Simulationszeit verkürzt werden, da komplizierte, aber irrelevante Schaltungseigenschaften vernachlässigt werden können.
- Ein konsequenter Top-Down-Entwurfsprozess kann leichter durchgeführt werden, da Modelle bereits auf hohen Abstraktionsebenen das gewünschte Verhalten eines Entwurfs simulieren können.
- Modelle haben eine große Bedeutung bei der Wiederverwendung und beim Verkauf von Schaltungskomponenten. So können Designer bereits mit den Modellen arbeiten, ohne die eigentliche Implementierung der fremden Schaltung zu kennen. Sie können so später erst entscheiden, ob sie die fremde Schaltung dazukaufen.

2.2.2 Einteilung von Modellen

Modelle werden in der Literatur unterschiedlich kategorisiert. Die Modelle werden unter anderem in vier Beschreibungsebenen eingeteilt: Transistorebene, Makroebene, Verhaltensebene und funktionale Ebene, dargestellt in *Tabelle 1*.

In der Transistorebene wird die Schaltung durch Basiselemente, wie Transistoren, Dioden, Kondensatoren, Widerstände, Spulen etc., beschrieben. Dieses Modell ist ein Strukturmodell bzw. Netzlistenmodell. Strukturmodelle bestehen aus mehreren miteinander verknüpften Teilmodellen. Die Untergliederung ist hierarchisch und kann bis auf die genannten Basiselemente zurückgeführt werden. In SPICE werden zum Beispiel die analogen Schaltungen als Strukturmodelle dargestellt.
Das Makromodell ist aus dem Transistormodell abgeleitet. Es ist somit ebenfalls ein Strukturmodell. Es enthält zusätzlich zu den Grundelementen abhängige und unabhängige Quellen. Mit diesen idealisierten Quellen können Bauelemente mit komplexerem Schaltungsverhalten vereinfacht nachgebildet werden.

In einem Verhaltensmodell oder einem funktionalen Modell beschreiben mathematische Gleichungen das Verhalten der Schaltung. Es gibt verschiedene Sprachen zur Beschreibung des Schaltungsverhaltens: abstrakte, sequentielle, aber auch Sprachen für die direkte Eingabe der Differentialgleichungssysteme. Ein Beispiel ist die VHDL-Erweiterung VHDL-AMS.
Die Abstraktionsebenen können auch gemischt auftreten, indem zum Beispiel Verhaltensmodelle in Makromodelle eingebunden werden.

Die Tabelle zeigt den Aufbau der analogen Modelle in den verschiedenen Ebenen jeweils in kontinuierlicher und in diskreter Zeit. Im Gegensatz zu Modellen in zeitkontinuierlicher arbeiten Modelle in zeitdiskreter Darstellung mit Werten zu abgetasteten Zeitpunkten. Durch die Abarbeitung auf einem digitalen Rechner haben Analogsimulatoren immer eine minimale Schrittweite auf der Zeitachse. In dieser Arbeit werden somit ausschließlich Modelle in quasi zeitkontinuierlicher Darstellung betrachtet.

	Kontinuierliche Zeit	**Diskrete Zeit**
Funktionale Ebene	Signalflussdiagramm mit Blöcken von mathematischen Gleichungen	Signalflussdiagramm mit Blöcken von mathematischen Gleichungen
Verhaltensebene	Blockdiagramm aus Einzelblöcken, die durch Algebro-Differentialgleichungen und/oder Laplace-Transferfunktionen beschrieben sind	Blockdiagramm mit Datenflussblöcken, die durch Algebro-Differenzengleichungen und/oder Z-Transferfunktionen beschrieben sind
Makroebene	Schaltung aus Grundelementen und gesteuerten Quellen	Schaltung aus Schaltern, Kondensatoren und gesteuerten Quellen
Schaltungsebene/ Transistorebene	Schaltung aus Grundelementen (Transistoren, Dioden, Kondensatoren, Widerständen etc.)	

Tabelle 1 – Abstraktionsebenen [1]

2.2.3 Modellierungsmethode parametrisierbarer Modelle

Die Modellierung mittels parametrisierbarer Modelle ist eine Methode zur Erstellung von Modellen für analoge Schaltungen. Hierbei werden Modelle für ganze Schaltungsklassen entwickelt und für eine gegebene Schaltung parametrisiert. Eine Schaltungsklasse ist eine Menge von pinkompatiblen Schaltungen mit ähnlichem Verhalten, die sich nur durch die Werte Ihrer Kenngrößen unterscheiden. Schaltungsklassen sind zum Beispiel Oszillatoren, Operationsverstärker, Stromspiegel etc. Die Kennwerte, wie zum Beispiel Eingangs-,

Ausgangswiderstand, Verstärkung etc. der konkreten Schaltung, sind die Parameter des Modells.

Modelle besitzen häufig die in *Abbildung 2* dargestellte Struktur. Die Modelle werden in Eingangs-, Übertragungs- und Ausgangsstufe unterteilt. Die Übertragungsstufe, auch Transfer- oder Funktionsstufe genannt, realisiert die allgemeine Übertragungsfunktion der Schaltungsklasse. Eingangs- und Ausgangsstufe bilden im Wesentlichen das analoge Anschlussverhalten nach.

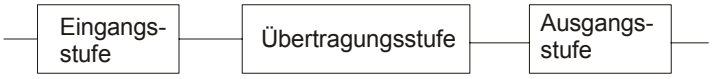

Abbildung 2 – Struktur eines parametrisierbaren Modells

Einfache Kenngrößen, wie zum Beispiel Eingangs- und Ausgangsimpedanzen, können meist als Makromodelle durch Widerstände, Kondensatoren etc. nachgebildet werden. Dabei sind die Werte der Schaltungselemente die Parameter des Modells.
Komplexere Eigenschaften der Schaltung werden oft durch eine funktionale Beschreibung realisiert. So kann zum Beispiel das Übertragungsverhalten durch Polynome, Exponentialfunktionen, Wurzelfunktionen etc. nachgebildet werden, wobei die Modellparameter direkte Parameter der Funktionen sind. Komplexe Eigenschaften können und werden teilweise auch auf Makroebene realisiert, doch werden die Modelle dann wiederum größer und komplizierter.

Die Genauigkeit und der dadurch entstehende Gültigkeitsbereich eines Modells können beeinflusst werden. Zum Beispiel können Kennwerte einer Schaltung idealisiert und im Modell vernachlässigt werden. So kann zum Beispiel der Eingangswiderstand eines Operationsverstärkers als unendlich angenommen und vernachlässigt werden. Dadurch kann die Simulationsgeschwindigkeit des Modells erhöht werden. Gleichzeitig leidet jedoch die Genauigkeit des Modells darunter.
Die mathematischen Zusammenhänge komplexer Eigenschaften können auch unterschiedlich formuliert werden. Dabei gilt meist die Regel: Je präziser ein Zusammenhang realisiert wird, desto umfangreicher wird die Funktion und deren Berechnungsaufwand.

So muss beim Modellentwurf abgewogen werden, wie genau das Modell der Originalschaltung entsprechen soll, und inwiefern dabei Geschwindigkeitseinbußen in der Simulation in Kauf genommen werden können.

Der Vorteil dieser Methode ist, dass bei Bedarf ein sehr präzises Modell der Originalschaltung erstellt werden kann, bzw., dass das Modell direkt den Anforderungen seiner Einsatzumgebung angepasst werden kann. Der Nachteil der Methode liegt im Aufwand zum Erstellen der Modelle. So existieren zwar allgemeine Modelle für Schaltungsklassen, allerdings müssen gegebene Schaltungen charakterisiert und zugehörige Modelle parametrisiert werden.

2.2.4 Charakterisierung einer Schaltung

Der Begriff Charakterisierung beschreibt die Ermittlung von Kennwerten einer Schaltung. Kenngrößen sind meist bekannte Eigenschaften, die eine Schaltungsklasse gemeinsam hat. So sind zum Beispiel die Kennwerte beim Kauf eines Schaltkreises in dem dazugehörigen Datenblatt beschrieben. Diese Kennwerte charakterisieren den gegebenen Schaltkreis.

Zur Ermittlung der Kennwerte wird die gegebene Schaltung in eine Messschaltung eingesetzt, vgl. *Abbildung 3*. Die gegebene Schaltung wird DUT (Device under Test) genannt. Die Messschaltungen bestehen aus Quellen, Widerständen etc., die an die Eingangs-, Ausgangs- und Versorgungsklemmen der gegebenen Schaltung angeschlossen werden. Dabei wird meist für jeden Kennwert ein unterschiedlicher Aufbau der Messschaltung benötigt.

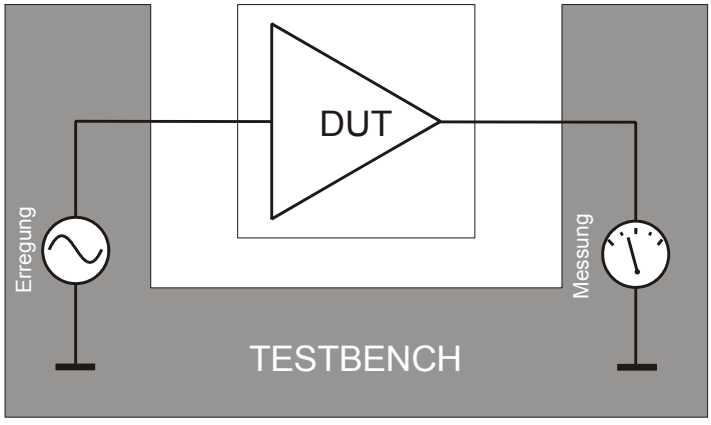

Abbildung 3 – DUT mit Testbench

Die Messschaltungen sind so aufgebaut, dass sie durch entsprechende Erregung ein spezielles Verhalten der Schaltung bewirken. Dabei wird das Verhalten der Schaltung möglichst nur durch den gesuchten und bekannte Kennwerte bestimmt. Aus den Simulationsergebnissen von der Messschaltung mit der gegebenen Schaltung kann der gesuchte Kennwert berechnet werden. Diese Berechnung wird Extraktion genannt, da sie oftmals aus der Bestimmung spezieller Einzelwerte aus den Simulationsergebniskennlinien besteht. Die Verbindung von Messschaltung und den Extraktionsgleichungen wird in dieser Arbeit mit Testbench (Messanlage) bezeichnet.

Testbenches können meist für alle Schaltungen einer Schaltungsklasse verwendet werden, da sie ähnlich einem Black-Box-Verfahren nur das Eingangs-, Ausgangs- und Übertragungsverhalten der Schaltung betrachten. So ist nur relevant, ob der entsprechende Kennwert in der Schaltung existiert, und ob die Klemmenbelegung des DUT mit der der Messschaltung übereinstimmt.

2.2.5 Weitere Modellierungsmethoden

Neben der Modellierung mittels parametrisierbarer Modelle existieren noch weitere Modellierungsmethoden, die in diesem Abschnitt kurz vorgestellt werden sollen.
Bekannte Modellierungsmethoden sind:

- Methode parametrisierbarer Modelle
- Numerische Modellierung
- Symbolische Modellierung
- Ordnungsreduktion

Mit der numerischen Modellierung werden Tabellenmodelle der Schaltung erstellt. Die Schaltung wird unter Änderung mehrerer Parameter, wie zum Beispiel Temperatur, Versorgungsspannung etc., simuliert und die Ergebnisse werden tabellarisch gespeichert. Bei Verwendung des Modells wird dann auf diese Ergebnisse zurückgegriffen.
Der Vorteil der Methode liegt in der guten Übereinstimmung mit der Originalschaltung. Nachteilig ist der Aufwand bei der Erstellung der Modelle. Zudem muss zwischen den Tabellenwerten interpoliert werden. Das führt bei mehrdimensionalen Abhängigkeiten zu einem erheblichen Rechenaufwand.

Bei der symbolischen Modellierung werden die Gleichungen, die die Schaltung beschreiben, mathematisch vereinfacht. Nachteil ist aber, dass die Vereinfachung der Gleichungen sehr schwierig ist, und die erstellten Modelle entweder immer noch zu langsam oder zu ungenau sind. Außerdem muss auch hier für jede Schaltung eine neue Vereinfachung der Gleichungen durchgeführt werden.

Bei der Methode der Ordnungsreduktion werden mit Hilfe mathematischer Techniken Modelle durch Reduktion der Ordnung der gegebenen Schaltung generiert. Bei der Ordnung einer Schaltung handelt es sich im Wesentlichen um die Größe der beschreibenden Matrizen in der Netzwerkgleichung. Entsprechende mathematische Werkzeuge funktionieren derzeit nur für lineare Netzwerke, so dass sie kaum Anwendung für elektronische Schaltungen bieten.

2.3. Zusammenfassung

In diesem Kapitel wurde auf den aktuellen Entwicklungsstand, die Techniken und Methoden im analogen und Mixed-Signal-Schaltungsentwurf ausführlich eingegangen. Die steigende Komplexität bei elektronischen Schaltkreisen erfordert ständig neue und verbesserte Methoden und Werkzeuge. Insbesondere die Diskrepanz bei der Verwendung von Digital- und Analogtechnik bei Mixed-Signal-Schaltungen führt zu Problemen im Schaltungsentwurf.
Mit Hilfe abstrakter Verhaltensmodelle können auch Gesamtsysteme mit digitalem und analogem Anteil in akzeptabler Zeit verifiziert werden.

Um der engen Kopplung von Verifikation und Entwurf von Mixed-Signal-Schaltkreisen gerecht zu werden, erfolgen ständig Entwurfs- und Modellierungsprozesse. Zur Zeiteinsparung sollen die Modelle automatisch aus den neuen oder korrigierten Schaltungsentwürfen generiert werden.
Diese Arbeit beschäftigt sich demzufolge mit der Entwicklung eines Werkzeuges zur Charakterisierung und Generierung von Modellen, konkret am Beispiel von Operationsverstärkerschaltungen.

3. Charakterisierung und Modellierung von Operationsverstärkern

3.1. Einleitung

In diesem Kapitel wird ein Modell des Operationsverstärkers vorgestellt. Der Operationsverstärker ist ein wichtiger Baustein im gesamten Elektronikbereich. Er findet zum Beispiel in analogen Rechenschaltungen, bei der Entkopplung von Signalen oder als Verstärker Anwendung. Das hier hergeleitete Modell soll in dem zu erstellenden Werkzeug Verwendung finden.

Zum Einsatz kommt die Modellierungsmethode parametrisierbarer Modelle, vgl. Kapitel *2.2.3* und *2.2.4*. Zunächst wird der Operationsverstärker anhand typischer Eigenschaften charakterisiert. Dazu werden Messschaltungen und Extraktionsgleichungen vorgestellt.

In Kapitel *3.3* werden die typischen Eigenschaften als Modell des Operationsverstärkers zusammengefasst und erläutert.

3.2. Charakterisierung

3.2.1 Einleitung

Jede Eigenschaft des Operationsverstärkers wird in diesem Kapitel nach folgendem Schema beschrieben: Anfänglich werden tabellarisch die zu ermittelnden Kenngrößen dargestellt. Danach folgt eine kurze Erläuterung zum Verständnis der Kenngrößen. Als Nächstes wird der Aufbau der Messschaltung beschrieben und schließlich werden die Gleichungen zur Extraktion der Kenngrößen aus den Simulationsergebnissen hergeleitet.

Zu jeder Messschaltung sind Standardwerte für die Betriebsparameter, wie zum Beispiel Versorgungsspannung, angegeben. Der Nutzer kann die Standardwerte entsprechend verändern.

Erläuterungen zu den verwendeten Formel- und Funktionsbezeichnungen befinden sich in Kapitel *7*.

3.2.2 Offsetspannung

Kenngrößen

Eingangs-Offsetspannung U_{os} in V

Erläuterung

Abbildung 4 zeigt den prinzipiellen Verlauf der Ausgangsspannung in Abhängigkeit von der Eingangsspannung. Die Offsetspannung U_{os} ist die Differenz-Eingangsspannung, bei der die Ausgangsspannung geradezu Null wird. Sie ist eine Fehlerdifferenzspannung zwischen den Eingängen, die bewirkt, dass übliche Operationsverstärker trotz Anlegen einer Differenzspannung von Null am Eingang eine Ausgangsspannung aufweisen.

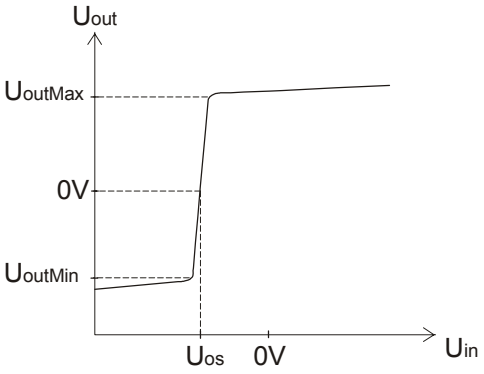

Abbildung 4 – Transfer-Kennlinie des Operationsverstärkers

Messschaltung

Abbildung 5 zeigt die verwendete Messschaltung. Dabei wird an den Eingang eine Quelle gelegt, die linear ihre Spannung von U_{inMin} auf U_{inMax} erhöht. Es wird eine DC-Transfer-Simulation durchgeführt und die Ausgangsspannung $U_{out}(U_{in})$ aufgezeichnet.

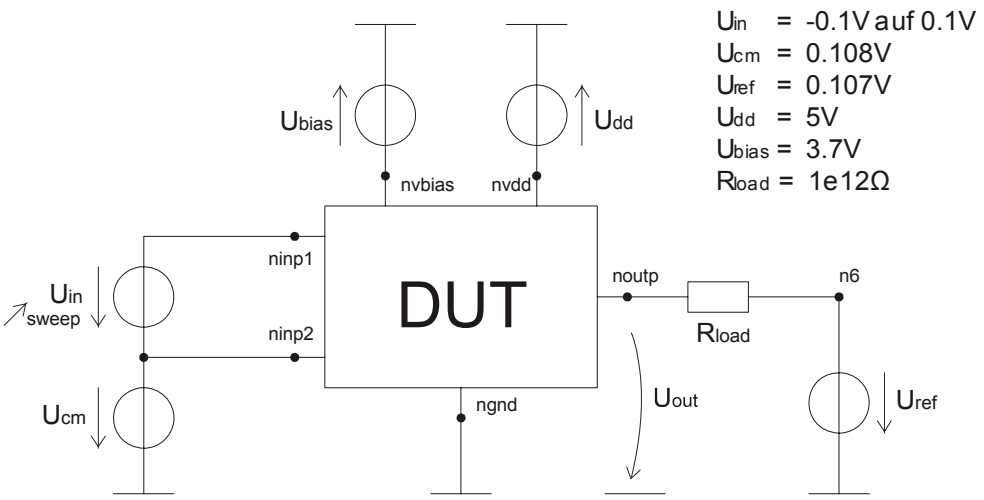

Abbildung 5 – Messschaltung Eingangs-Offsetspannung und Ausgangsspannungsbegrenzung

Extraktion

An der Stelle des Nulldurchgangs der Ausgangsspannung U_{out} wird die dazugehörige Eingangsspannung U_{in} als Offsetspannung U_{os} bestimmt. Dabei wird zwischen den beiden nächstgelegenen Werten interpoliert. Das Verfahren ist durch Nutzung vorgefertigter Extraktionsbefehle festgelegt. Vorraussetzung ist, dass U_{os} innerhalb der Grenzen U_{inMin} und U_{inMax} liegt.

$$U_{os}: \quad U_{out}(U_{os}) = 0,\ U_{inMin} < U_{os} < U_{inMax} \qquad \textbf{(1)}$$

3.2.3 Ausgangsaussteuerbereich

Kenngrößen

Obere Grenze des Aussteuerbereiches U_{outMax} in V

Untere Grenze des Aussteuerbereiches U_{outMin} in V

Erläuterung

Der Ausgangsaussteuerbereich des Operationsverstärkers ist der annähernd lineare Teil der Transfer-Kennlinie, vgl. *Abbildung 4*. Die daraus resultierenden Grenzen sind die obere Grenze U_{outMax} und die untere Grenze U_{outMin} des Aussteuerbereiches.

Messschaltung

Zur Bestimmung der Ausgangsspannungsbegrenzung wird die gleiche Messschaltung wie zur Bestimmung der Offsetspannung verwendet, vgl. *Abbildung 5*.

Extraktion

Durch Ableitung der Ausgangsspannung $U_{out}(U_{in})$ nach der Differenzeingangsspannung U_{in} wird die Verstärkung $V(U_{in})$ bestimmt. Da die Transfer-Kennlinie im Aussteuerbereich nur annähernd linear ist, besitzt die Verstärkungskennlinie keinen konstanten Bereich, sondern zeigt immer einen gewissen Anstieg. *Abbildung 4*, zeigt wie solch eine Verstärkungskennlinie aussehen könnte.

Zur Bestimmung der Grenzen werden die Punkte der Verstärkungskennlinie verwendet, an denen die maximale Verstärkung um einen bestimmten Faktor gesunken ist. Dieser Faktor, in den folgenden Gleichungen mit rate bezeichnet, ist ein vom Anwender vorzugebender Extraktionsparameter, der die Genauigkeit der Modellierung beeinflusst.

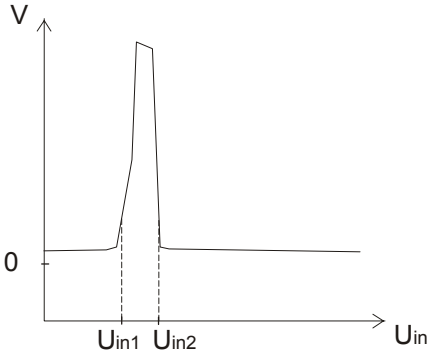

Abbildung 6 – Ableitung der Transfer-Kennlinie des Operationsverstärkers

Die Ausgangsspannungsgrenzen werden bei den ermittelten Punkten U_{in1} und U_{in2} abgelesen.

$$V(U_{in}) = \frac{dU_{out}(U_{in})}{dU_{in}} \qquad \textbf{(2)}$$

$$U_{outMin,Max} = U_{out}(U_{in1,2}) \text{ mit } U_{in1,2}: \quad V(U_{in1,2}) = \frac{\max\left(V(U_{in})\right)}{\text{rate}} \qquad \textbf{(3)}$$

3.2.4 Eingangsimpedanz

Abbildung 7 zeigt das verwendete Ersatzschaltbild der Eingangsimpedanz des Operationsverstärkers.

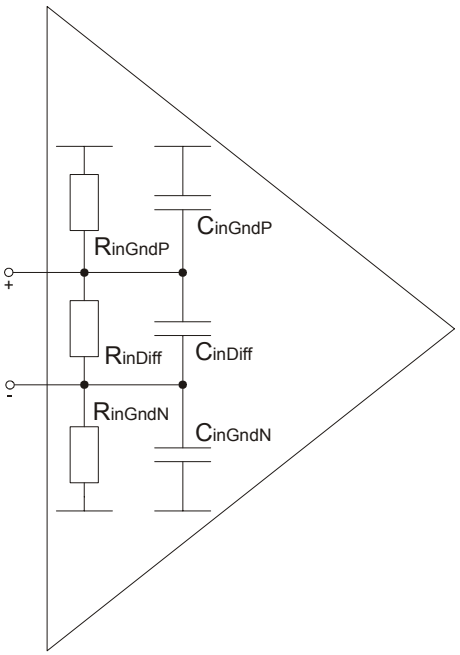

Abbildung 7 – Ersatzschaltung Eingangsimpedanz des OPV

Man unterscheidet zwischen den Gleichtaktwiderständen R_{inGndP}, R_{inGndN} und dem Differenz-Eingangswiderstand R_{inDiff} sowie deren Parallelkapazitäten C_{inGndP}, C_{inGndN} und C_{inDiff}. Da diese Widerstände bei den betrachteten CMOS-Operationsverstärkern sehr hohe Werte haben (zum Beispiel $R_{inDiff} = 10^9..10^{25}\Omega$, $R_{inGndP} \approx R_{inGndN} \approx 10 \cdot R_{inDiff}$), werden sie bei der Modellierung des Operationsverstärkers vernachlässigt.

3.2.4.1. Gleichtakt-Eingangskapazität

Kenngrößen

Gleichtakt-Eingangskapazitäten am P- und N-Eingang C_{inGndP} und C_{inGndN} in F

Erläuterung

Für den positiven und den negativen Eingang des Operationsverstärkers wird eine Parallelersatzschaltung für die Kapazitäten C_{inGndP} und C_{inGndN} angenommen, vgl. *Abbildung 8*. Für den komplexen Eingangsleitwert $\underline{Y}_{in}$ der beiden Eingänge gelten folgende Gleichungen:

$$\underline{Y}_{inP} = \frac{\underline{I}_{inP}}{\underline{U}_{in}} = \mathrm{Re}\{\underline{Y}_{inP}\} + j \cdot \mathrm{Im}\{\underline{Y}_{inP}\} = j \cdot 2\pi f \cdot C_{inGndP} \quad \textbf{(4)}$$

$$\underline{Y}_{inN} = \frac{\underline{I}_{inN}}{\underline{U}_{in}} = \mathrm{Re}\{\underline{Y}_{inN}\} + j \cdot \mathrm{Im}\{\underline{Y}_{inN}\} = j \cdot 2\pi f \cdot C_{inGndN} \quad \textbf{(5)}$$

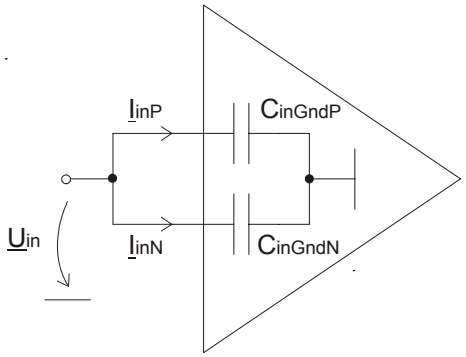

Abbildung 8 – Ersatzschaltung Gleichtakt-Eingangsimpedanz

Messschaltung

Der Operationsverstärker wird im Gleichtaktbetrieb (common mode) betrieben. Am Eingang liegt eine AC-Spannungsquelle, vgl. *Abbildung 9*. In einer AC-Simulation werden die Kennlinien $\underline{I}_{inP}(f)$, $\underline{I}_{inN}(f)$ und $\underline{U}_{in}(f)$ aufgenommen.

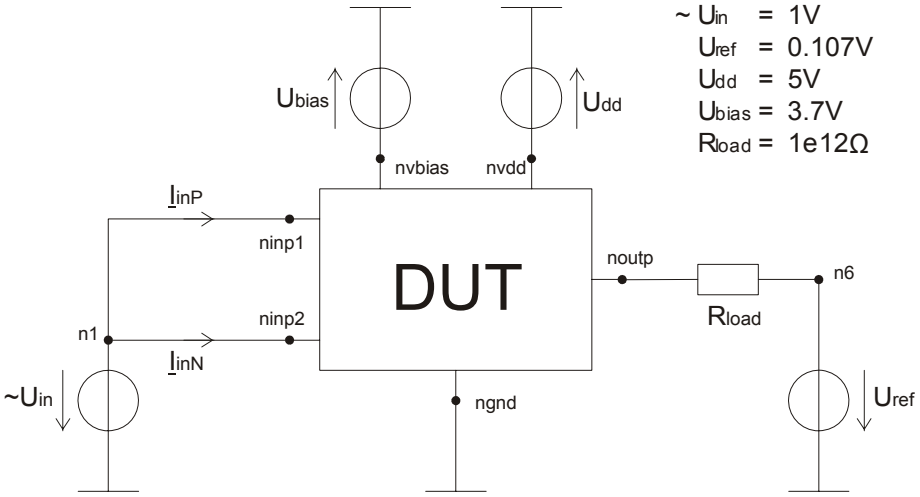

Abbildung 9 – Messschaltung Gleichtakt-Eingangsimpedanz

Extraktion

Aus *(4)* und *(5)* lassen sich die Gleichungen zur Bestimmung der Kapazitäten bei der Messfrequenz f_m ableiten. Die Messfrequenz ist ein Extraktionsparameter, der durch den Anwender festgelegt werden muss.

$$C_{inGndP} = \frac{1}{2\pi f_m} \cdot \mathrm{Im}\left\{\frac{\underline{I}_{inP}(f_m)}{\underline{U}_{in}(f_m)}\right\} \qquad \textbf{(6)}$$

$$C_{inGndN} = \frac{1}{2\pi f_m} \cdot \mathrm{Im}\left\{\frac{\underline{I}_{inN}(f_m)}{\underline{U}_{in}(f_m)}\right\} \qquad \textbf{(7)}$$

3.2.4.2. Differenz-Eingangskapazität

Kenngrößen

Differenz-Eingangskapazität C_{inDiff} in F

Erläuterung

Abbildung 10 zeigt das Ersatzschaltbild zur Ermittlung der Differenz-Eingangskapazität des Operationsverstärkers. Es gilt folgende Gleichung für den Eingangsleitwert $\underline{Y}_{in}$:

$$\underline{Y}_{in} = \frac{\underline{I}_{in}}{\underline{U}_{in}} = \underline{Y}_{inDiff} + \underline{Y}_{inGnd} = j \cdot 2\pi f_m \cdot C_{inDiff} + \frac{1}{\frac{1}{j \cdot 2\pi f_m \cdot C_{inGndP}} + \frac{1}{j \cdot 2\pi f_m \cdot C_{inGndN}}} \qquad \textbf{(8)}$$

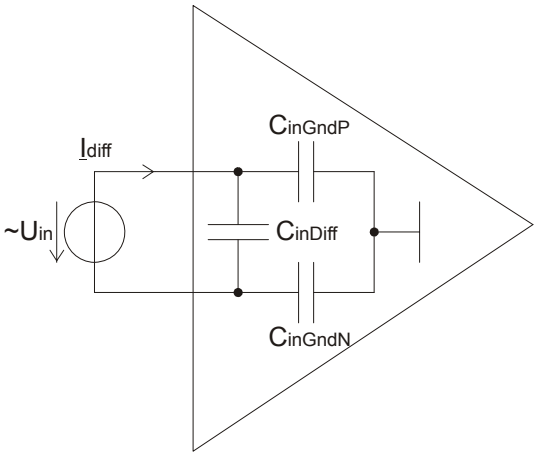

Abbildung 10 – Ersatzschaltung Differenz-Eingangsimpedanz

Messschaltung

Die beiden Eingänge des Operationsverstärkers werden an eine AC-Spannungsquelle angeschlossen. *Abbildung 11* zeigt die entsprechende Messschaltung. Eine AC-Simulation wird durchgeführt und die Kennlinien $\underline{I}_{in}(f)$ und $\underline{U}_{in}(f)$ werden aufgezeichnet.

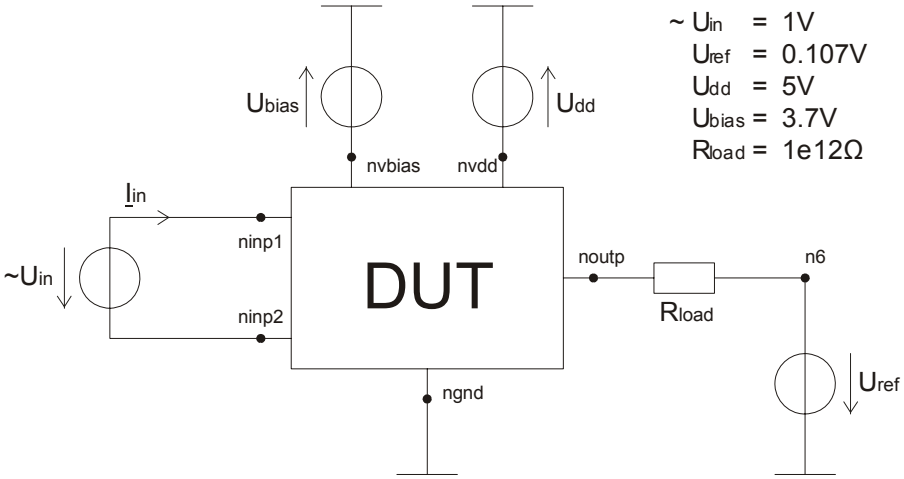

Abbildung 11 – Messschaltung Differenz-Eingangsimpedanz

Extraktion

Aus *(8)* lässt sich folgende Gleichung zur Berechnung der Differenzeingangskapazität C_{inDiff} ableiten. Die Gleichtakteingangsimpedanzen C_{inGndP} und C_{inGndN} müssen bekannt sein.

$$C_{inDiff} = \frac{1}{2\pi f_m} \cdot \mathrm{Im}\left\{\frac{\underline{I}_{in}(f_m)}{\underline{U}_{in}(f_m)}\right\} - \frac{1}{\frac{1}{C_{inGndP}} + \frac{1}{C_{inGndN}}} \qquad \textbf{(9)}$$

3.2.5 Ausgangsimpedanz

Kenngrößen

Ausgangskapazität C_{out} in F

Ausgangswiderstand R_{out} in Ω

Erläuterung

Als Ersatzschaltung für die Ausgangsimpedanz wird eine Parallelersatzschaltung angenommen, vgl. *Abbildung 12*. Es gilt folgende Gleichung für den Ausgangsleitwert $\underline{Y}_{out}$:

$$\underline{Y}_{out} = \frac{\underline{I}_{out}}{\underline{U}_{out}} = \mathrm{Re}\{\underline{Y}_{out}\} + j \cdot \mathrm{Im}\{\underline{Y}_{out}\} = \frac{1}{R_{out}} + j \cdot 2\pi f \cdot C_{out} \qquad \textbf{(10)}$$

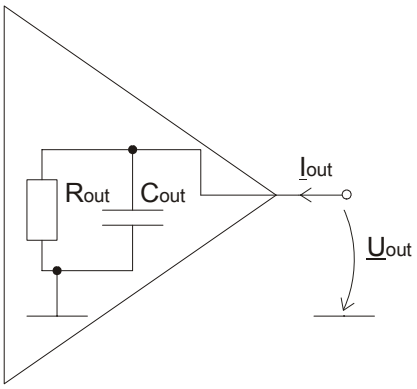

Abbildung 12 – Ersatzschaltung Ausgangsimpedanz

Messschaltung

Zur Bestimmung der komplexen Kennlinien $\underline{I}_{out}(f)$ und $\underline{U}_{out}(f)$ wird eine AC-Simulation durchgeführt.

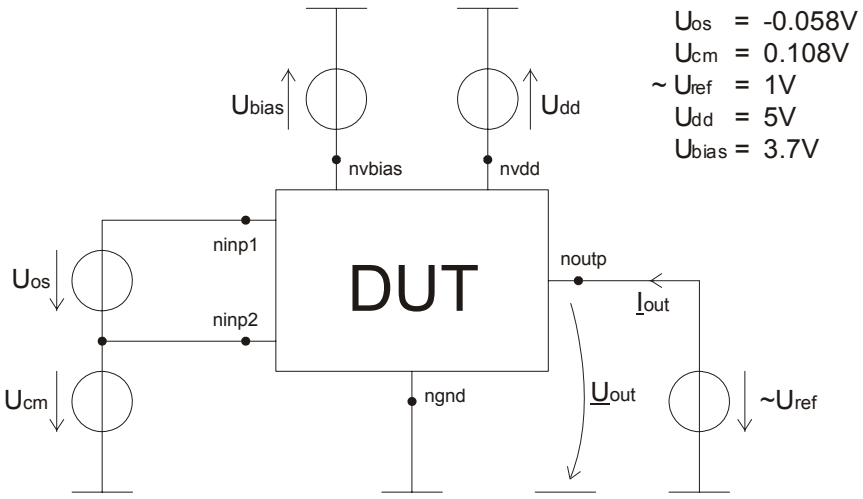

Abbildung 13 – Messschaltung Ausgangsimpedanz

Dabei wird an den Ausgang des Operationsverstärkers eine AC-Spannungsquelle gelegt. *Abbildung 13* zeigt die gesamte Messschaltung.

Extraktion

Aus *(10)* lassen sich folgende Gleichungen zur Berechnung der Ausgangkapazität C_{out} und des Ausgangswiderstandes R_{out} ableiten:

$$C_{out} = \frac{1}{2\pi f_m} \cdot \mathrm{Im}\left\{\frac{\underline{I}_{out}(f_m)}{\underline{U}_{out}(f_m)}\right\} \quad \textbf{(11)}$$

$$R_{out} = \frac{1}{\mathrm{Re}\left\{\frac{\underline{I}_{out}(f_m)}{\underline{U}_{out}(f_m)}\right\}} \quad \textbf{(12)}$$

3.2.6 Leerlaufverstärkung

Kenngrößen

Leerlaufverstärkung V_{d0} in dB

Erläuterung

Die Differenzverstärkung V_d des Operationsverstärkers ist frequenzabhängig. Da die Verstärkung im Allgemeinen logarithmisch aufgetragen wird, erfolgt die Bestimmung der Leerlaufverstärkung V_{d0} bei sehr tiefer Frequenz (zum Beispiel 10Hz).

Messschaltung

Am Operationsverstärker wird ein großer Lastwiderstand angeschlossen und an den Eingang ein komplexes Eingangssignal gelegt, vgl. *Abbildung 14*. Aufgezeichnet wird die Verstärkungskennlinie $V_d(f) = \mathrm{dB}\left(\left|\frac{\underline{U}_{out}(f)}{\underline{U}_{diff}(f)}\right|\right)$. Zur Bestimmung der Leerlaufverstärkung würde ein einzelner Messwert ausreichen. Zur weiteren Verwendung der Messschaltung zur Bestimmung des Frequenzganges der Verstärkung, vgl. *3.2.7*, wird die gesamte Verstärkungskennlinie aufgezeichnet und daraus der DC-Wert extrahiert.

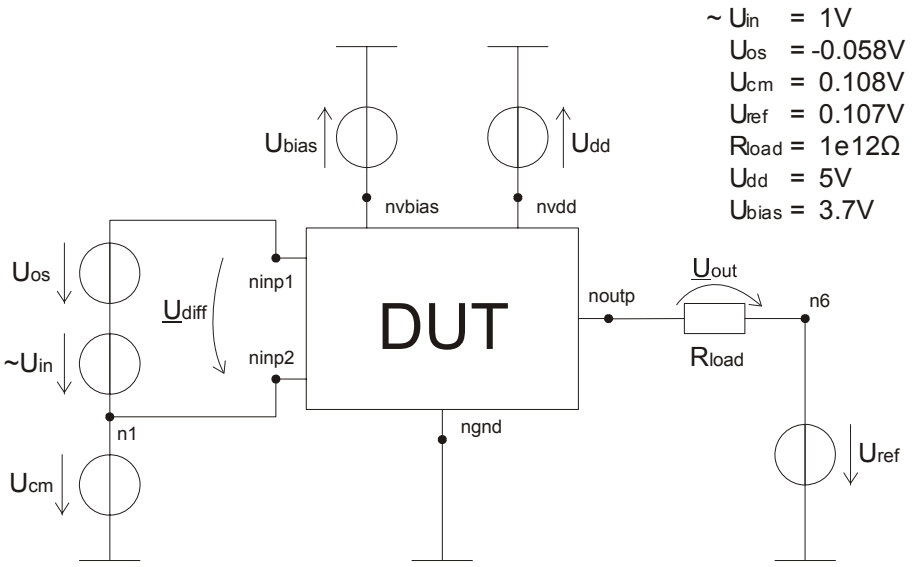

Abbildung 14 – Messschaltung Frequenzgang der Leerlaufverstärkung

Extraktion

$$V_{d0} = V_d(10\text{Hz}) \qquad \textbf{(13)}$$

3.2.7 Frequenzgang der Verstärkung

Kenngrößen

Obere Grenzfrequenz f_o in Hz

Erläuterung

f_o ist die obere Grenzfrequenz des Operationsverstärkers. Bei dieser Frequenz ist die Differenzverstärkung V_d vom Wert der Leerlaufverstärkung V_{d0} auf den Wert $V = \frac{1}{\sqrt{2}} \cdot V_{d0}$ gesunken (3-dB-Grenzfrequenz).

Bei der Transitfrequenz f_t hat die Verstärkung V_d den Wert $V_d(f_t) = 0\text{dB}$.

Messschaltung

Zur Bestimmung der oberen Grenzfrequenz f_o und der Transitfrequenz f_t wird die gleiche Messschaltung wie zur Bestimmung der Leerlaufverstärkung verwendet, vgl. *Abbildung 14*.

Extraktion

$$f_o: \quad V_d(f_o) = V_{d0} - 3\text{dB} \qquad \textbf{(14)}$$

$$f_t: \quad V_d(f_t) = 0\text{dB} \qquad \textbf{(15)}$$

Probleme

Es kann sein, dass sich im Modell die obere Grenzfrequenz f_o bereits durch die Ausgangsimpedanz ergibt. Wenn nun im Modell zusätzlich die obere Grenzfrequenz f_o nachgebildet wird, entsteht ein unerwünschter doppelter Pol bei f_o. Durch Berechnung der Polstelle der Ausgangsimpedanz, abgeleitet aus *(10)*, kann der Operationsverstärker daraufhin überprüft werden:

$$f_o = \frac{1}{2\pi \cdot R_{out} \cdot C_{out}} \qquad \textbf{(16)}$$

3.2.8 Gleichtaktunterdrückung

Kenngrößen

Gleichtaktunterdrückungsverhältnis CMRR (common mode rejection ratio) in dB

Erläuterung

Die Gleichtaktunterdrückung CMRR ergibt sich aus der Differenz von Differenzverstärkung V_d in dB und Gleichtaktverstärkung V_{gl} in dB: $CMRR(f) = V_d(f) - V_{gl}(f)$. Die Gleichtakt-unterdrückung ist frequenzabhängig. Zur Bestimmung der Kenngröße wird ein Wert bei sehr tiefer Frequenz (zum Beispiel 10Hz) genommen, analog der Extraktion der Leerlaufverstärkung.

Messschaltung

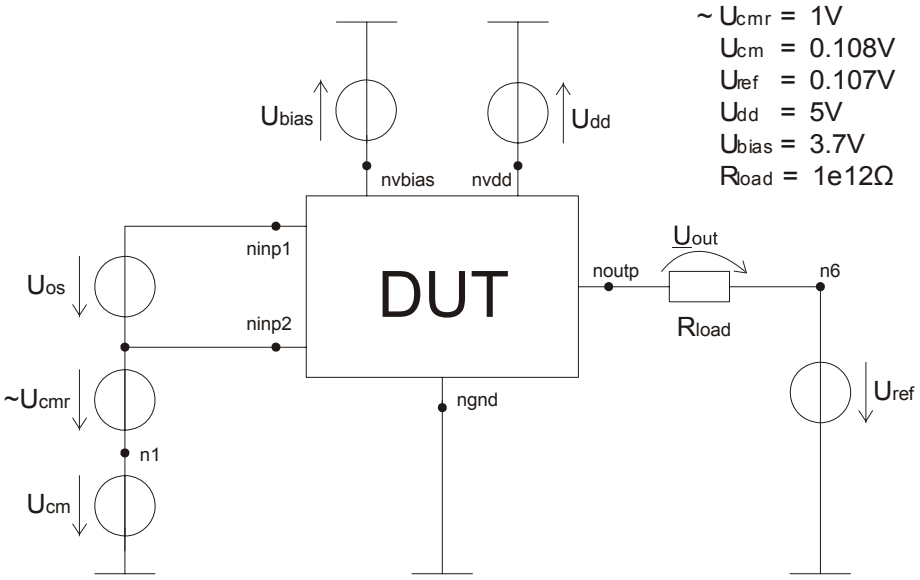

Abbildung 15 – Messschaltung CMRR

Abbildung 15 zeigt die Messschaltung zur Bestimmung des Gleichtaktunterdrückungsverhältnisses. Es wird eine Störspannungsquelle an den Eingang gelegt und eine AC-Simulation durchgeführt. Dabei werden die Kennlinien $\underline{U}_{cmr}(f)$ und $\underline{U}_{out}(f)$ aufgezeichnet.

Extraktion

Zur Berechnung des Gleichtaktunterdrückungsverhältnisses muss die Leerlaufverstärkung V_{d0} bekannt sein:

$$V_{gl}(f) = dB\left(\left|\frac{\underline{U}_{out}(f)}{\underline{U}_{cmr}(f)}\right|\right) \quad \textbf{(17)}$$

$$CMRR = \left|V_{d0} - V_{gl}(10Hz)\right| \quad \textbf{(18)}$$

3.2.9 Betriebsspannungsunterdrückung

Kenngrößen

Betriebsspannungsunterdrückungsverhältnis PSRR (power supply rejection ratio) in dB

Erläuterung

Die Betriebsspannungsunterdrückung PSRR gibt den Einfluss der Änderung der Betriebspannung in Abhängigkeit zur Differenzverstärkung an. Er lässt sich aus der Differenz der Differenzverstärkung V_d in dB und der Betriebsspannungsverstärkung V_{psr} in dB berechnen: $PSRR(f) = V_d(f) - V_{psr}(f)$. Die Betriebsspannungsunterdrückung ist frequenzabhängig. Zur Bestimmung der Kenngröße wird ein Wert bei sehr tiefer Frequenz (zum Beispiel 10Hz) genommen, analog der Extraktion der Leerlaufverstärkung.

Messschaltung

Abbildung 16 zeigt die Messschaltung zur Bestimmung des Betriebsspannungsunterdrückungsverhältnisses. Es wird eine Störspannungsquelle an den Versorgungseingang gelegt und eine AC-Simulation durchgeführt. Dabei werden die Kennlinien $\underline{U}_{psr}(f)$ und $\underline{U}_{out}(f)$ aufgezeichnet.

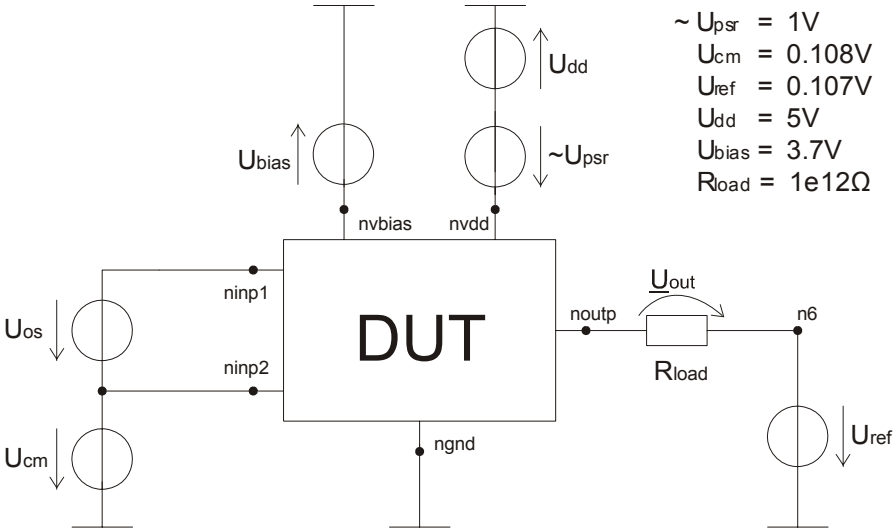

Abbildung 16 – Messschaltung PSRR

Extraktion

Zur Berechnung des Betriebsspannungsunterdrückungsverhältnisses muss die Leerlaufverstärkung V_{d0} bekannt sein:

$$V_{psr}(f) = dB\left(\left|\frac{\underline{U}_{out}(f)}{\underline{U}_{psr}(f)}\right|\right) \quad \textbf{(19)}$$

$$PSRR = \left|V_{d0} - V_{psr}(10Hz)\right| \quad \textbf{(20)}$$

3.2.10 Slew Rate

Kenngrößen

Positive Slew Rate S_{RP} in $\frac{V}{s}$

Negative Slew Rate S_{RN} in $\frac{V}{s}$

Erläuterung

Die Slew Rate bezeichnet die Anstiegsgeschwindigkeit der Ausgangsspannung bei impulsförmiger Ansteuerung des Eingangs. Dabei ist die positive Slew Rate S_{RP} durch einen positiven Sprung und die negative Slew Rate S_{RN} durch einen negativen Sprung am Eingang gekennzeichnet.

Messschaltung

Abbildung 17 zeigt die Messschaltung zur Bestimmung der Slew Rate. An den positiven Eingang des Operationsverstärkers wird ein positiver bzw. negativer Spannungssprung angelegt. Es wird eine Transienten-Simulation durchgeführt und die Kennlinie $U_{out}(t)$ aufgenommen.

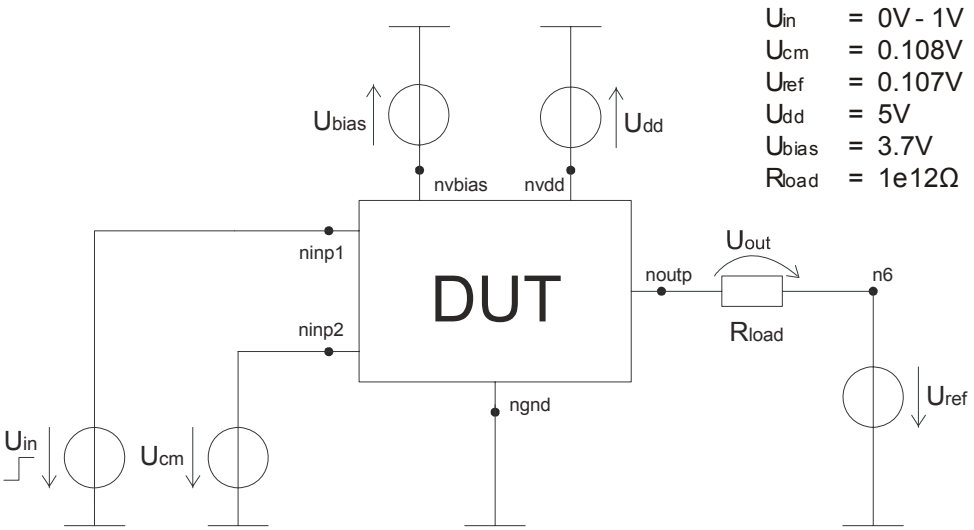

Abbildung 17 – Messschaltung Slew Rate

Extraktion

$$S_{RP} = \max\left(\frac{dU_{out}(t)}{dt}\right) \qquad \textbf{(21)}$$

$$S_{RN} = \min\left(\frac{dU_{out}(t)}{dt}\right) \qquad \textbf{(22)}$$

3.3. Modell

3.3.1 Einleitung

In diesem Kapitel wird der Aufbau eines Operationsverstärkermodells vorgestellt. Dabei werden ausschließlich die in Kapitel *3.2* vorgestellten Charakterisierungsergebnisse des Operationsverstärkers berücksichtigt.

Die gezeigten Kenngrößen des Operationsverstärkers, deren Betrag mit Hilfe der Messschaltungen bestimmt werden kann, dienen als Parameter des Modells. Die Kenngrößen können dann wiederum mit den Messschaltungen am Modell gemessen und mit der Originalschaltung verglichen werden. *Abbildung 18* zeigt den Aufbau des Modells. Es ist in

Eingangs-, Übertragungs- und Ausgangsstufe unterteilt. Die nächsten beiden Kapitel beschreiben detailliert den Aufbau der Eingangs- und Ausgangsstufe, sowie den Aufbau der Übertragungsstufe.

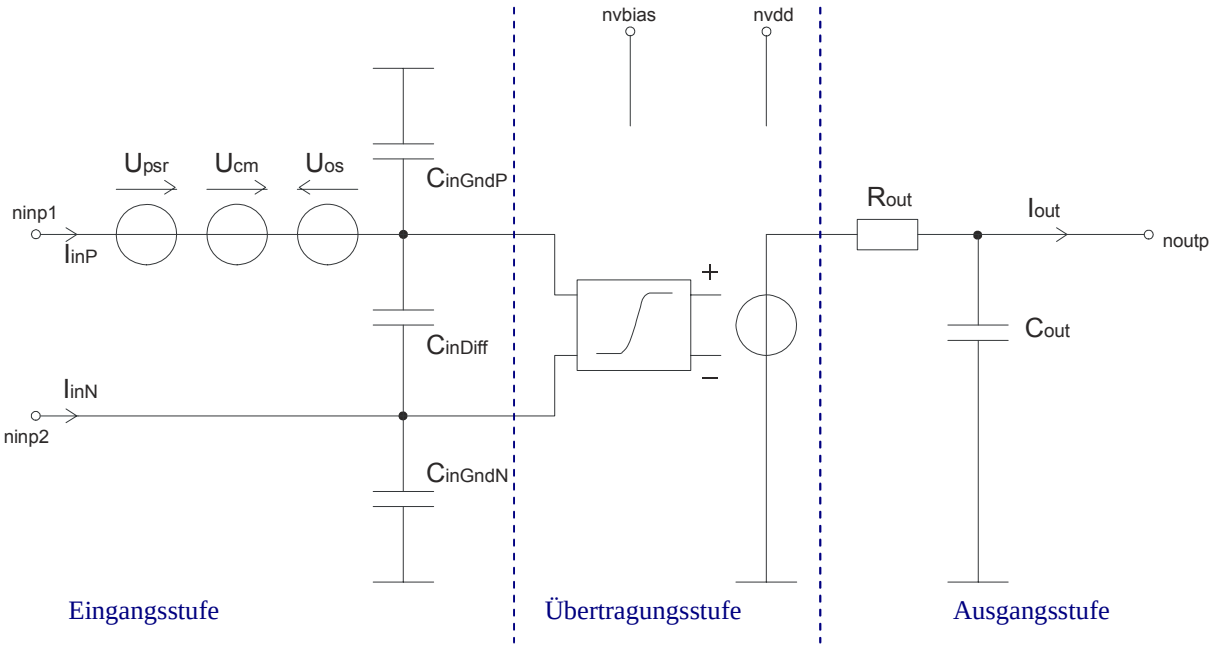

Abbildung 18 – Modell des Operationsverstärkers

3.3.2 Ein- und Ausgangsstufe

Die Eingangsstufe ist durch die Eingangskapazitäten, die Offsetspannung und die Störspannungsquellen gekennzeichnet, vgl. *Abbildung* 18. Diese Eigenschaften können mit den Grundelementen Kondensator und Spannungsquelle nachgebildet werden. Dadurch ist zur Nachbildung des Eingangsverhaltens ein Modell auf Schaltungsebene ausreichend.

Die Eingangskapazitäten C_{inGndP}, C_{inGndN}, C_{inDiff} werden in Kapitel *3.2.4* hergeleitet. Die Werte der Störspannungsquellen lassen sich aus Gleichtaktunterdrückung CMRR, vgl. Kapitel *3.2.8*, und Betriebsspannungsunterdrückung PSRR, vgl. Kapitel *3.2.9*, nach *(23)* und *(24)* berechnen.

$$U_{psr} = U_{vdd} \cdot \frac{1}{10^{\frac{PSRR}{20}}} \qquad \textbf{(23)}$$

$$U_{cm} = \frac{U_{ninp1} + U_{ninp2}}{2} \cdot \frac{1}{10^{\frac{CMRR}{20}}} \qquad \textbf{(24)}$$

Die Berechnung der Offsetspannung U_{os} wird in Kapitel *3.2.2* hergeleitet.

Die Ausgangsstufe wird nur durch die Ausgangsimpedanz C_{out} und R_{out} bestimmt, vgl. Kapitel *3.2.5*. Dadurch ist auch hier ein Modell auf Schaltungsebene zur Beschreibung der Ausgangsstufe ausreichend.

3.3.3 Übertragungsstufe

In der Übertragungsstufe werden die Eigenschaften Frequenzgang, Verstärkung, Slew Rate und Ausgangsspannungsbegrenzung nachgebildet, vgl. Kapitel *3.2.3*, *3.2.6*, *3.2.7* und *3.2.10*. Zur Beschreibung dieser Eigenschaften soll ein funktionales Modell verwendet werden.

Der Frequenzgang der Übertragungsfunktion ist durch die obere Grenzfrequenz gekennzeichnet. Mit Hilfe einer vorgefertigten Funktion der Verhaltensbeschreibungssprache, welche die Laplace-Übertragungsfunktion aus Null- und Polstellen berechnet, kann der Frequenzgang nachgebildet werden.
Der Verstärkungsfaktor wird als gesteuerte Quelle durch Multiplikation mit der Eingangsspannung nachgebildet:

$$k = 10^{\frac{V_{d0}}{20}} \qquad \textbf{(25)}$$

$$U_{out}(U_{in}) = k \cdot U_{in} \qquad \textbf{(26)}$$

Die Anstiegsgeschwindigkeit der Ausgangsspannung, gekennzeichnet durch die Slew Rate, kann ebenfalls in VHDL-AMS mit einer vorgefertigten Funktion nachgebildet werden.
Je nach Anforderungen an das Modell können zur Nachbildung der Ausgangsspannungsbegrenzung verschiedene Ansätze verwendet werden. Ziel ist es, die in *Abbildung 4* dargestellte Übertragungsfunktion durch die gegebenen Grenzen nachzubilden. Die einfachste Art einer Begrenzung ist die stückweise lineare Beschreibung der Kennlinie nach *(27)*. *Abbildung 19* stellt die Funktion graphisch dar.

$$U_{out}(U_{in}) = \begin{cases} U_{outMin} & , \text{ für } U_{in} < \frac{U_{outMin}}{k} \\ k \cdot U_{in} & , \text{ für } \frac{U_{outMin}}{k} \le U_{in} < \frac{U_{outMax}}{k} \\ U_{outMax} & , \text{ für } U_{in} \le \frac{U_{outMax}}{k} \end{cases} \qquad \textbf{(27)}$$

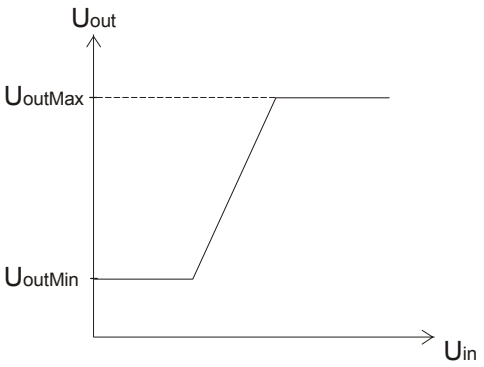

Abbildung 19 – Stückweise lineare Begrenzungskennlinie

Vorteil dieser Funktion ist, dass sie sehr einfach zu parametrisieren und während der Simulation schnell zu berechnen ist. Der entscheidende Nachteil liegt in den beiden Knickstellen. Hier treten für die Lösungsalgorithmen von Schaltungssimulatoren häufig Probleme mit der Konvergenz auf.

Eine andere Möglichkeit ist die Darstellung mit Hilfe einer Tangens-hyperbolicus-Funktion nach *(30)*. *Abbildung 20* zeigt den prinzipiellen Verlauf dieser Funktion.

$$\text{scale} = \frac{U_{\text{outMax}} - U_{\text{outMin}}}{2} \quad \textbf{(28)}$$

$$\text{offset} = \frac{U_{\text{outMax}} + U_{\text{outMin}}}{2} \quad \textbf{(29)}$$

$$U_{\text{out}}(U_{\text{in}}) = \text{scale} \cdot \tanh(k \cdot U_{\text{in}}) + \text{offset} \quad \textbf{(30)}$$

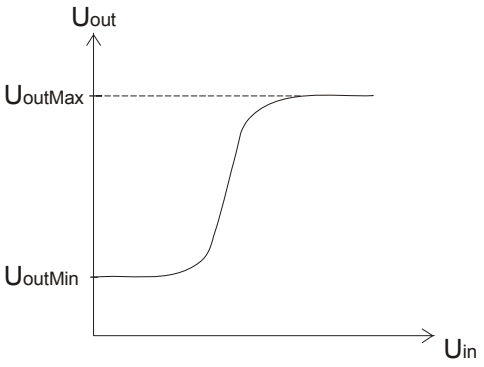

Abbildung 20 – Tangens hyberbolicus Ansatz

Diese Funktion ist numerisch sehr robust, da weder Überlauf noch Knickstellen auftreten. Von Nachteil ist, dass die Funktion nicht ausreichend parametrisierbar ist. So kann die Größe des annähernd linearen Bereiches bis zum Knick nicht explizit eingestellt werden.

Eine dritte Möglichkeit wäre die Darstellung der Begrenzungskennlinie mit Hilfe einer Potenz-Wurzel-Funktion nach *(31)*. *Abbildung 21* zeigt den prinzipiellen Verlauf dieser Funktion.

$$U_{out}(U_{in}) = k \cdot U_{in} \cdot \left(1 + \left(\frac{k \cdot U_{in}}{scale}\right)^{n}\right)^{-\frac{1}{n}} + offset \qquad \textbf{(31)}$$

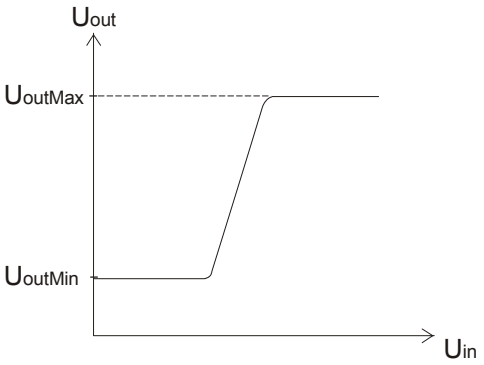

Abbildung 21 – Ansatz mit einer Potenz-Wurzel-Funktion

Vorteil neben dem robusten Verlauf der Funktion ohne Knickstellen oder Überlauf ist, dass mit Hilfe des Parameters n die Güte der Annäherung der Kennlinie an die stückweise lineare Funktion eingestellt werden kann. Je größer der Wert für n ist, desto schärfer wird die Knickstelle. Damit kann die Größe des linearen Bereiches eingestellt werden. Nachteil dieser Funktion liegt in ihrer Komplexität. So ist es fraglich, ob der notwendige Aufwand zur Berechnung der Funktion für den Simulator im Verhältnis zur gewonnenen Genauigkeit lohnenswert ist.

4. Softwareentwurf

4.1. Einleitung

Die folgenden Seiten beschreiben den Softwareentwurf des Werkzeuges „Chameleon“ (**CHA**racterization and **M**od**EL**ing **E**nvir**ON**ment). Der klassische Softwareentwurf besteht aus vier Phasen: Analysephase, Entwurfsphase, Implementierungsphase sowie Test- und Wartungsphase.

In Abschnitt *4.2 Analyse* werden die Anforderungen an die Software konkretisiert. Es erfolgt eine detaillierte Aufgabenbeschreibung. Probleme und Ziele anhand unterschiedlicher Betrachtungsweisen werden erläutert. Als Resultat dieser Phase entsteht eine Anforderungsspezifikation. Diese bietet eine Basis für Designentscheidungen in der Entwurfs- und Realisierungsphase und für Akzeptanztestrichtlinien in der Test- und Wartungsphase.

In Abschnitt *4.3 Entwurf* werden verschiedene Lösungsansätze zur Erfüllung der Anforderungsspezifikation beschrieben und getroffene Entscheidungen begründet. Resultat dieser Phase ist der theoretische Systementwurf des Programms.

In Abschnitt *4.4 Implementierung* wird das fertige Programm anhand von Beispielen vorgestellt.

In Abschnitt *4.5 Test und Wartung* werden verwendete Testmethoden beschrieben.

4.2. Analyse

4.2.1 Einleitung

Die Analyse im Softwareentwurf dient dazu, das Ziel der Aufgabenstellung zu verdeutlichen. Dazu werden gängige Analysemethoden des Softwaredesigns verwendet. Nach der Aufgabenanalyse werden kurz existierende Werkzeuge mit ähnlichem Funktionsumfang vorgestellt. Danach erfolgen die Analyse mittels sog. Stakeholder, die Problemanalyse und die Zielanalyse. Da Kapitel *2 – Technischer Hintergrund* – bereits ausführlich die Problematik im analogen Schaltungsentwurf beschreibt, wird die Problemanalyse unter Punkt *4.2.4* nur kurz angesprochen.

In der Systemanalyse zeigen verschiedene Diagramme die gewünschten Anforderungen und Ziele.

4.2.2 Aufgabenanalyse

Primäraufgabe ist die Entwicklung eines Programms zur Schaltungsmodellierung anhand der Methode parametrisierbarer Modelle. Die Erstellung geeigneter Modelle ist ein sehr komplexer Vorgang. Aus diesem Grund soll das Programm Möglichkeiten zur Eingabe und Verwaltung von Modellvorlagen bieten.
Für die Schaltungscharakterisierung sollen Bibliotheken für Testbenches erstellt und verwaltet werden können. Zur Simulation der Schaltungen sollen vorhandene Simulatoren verwendet werden.

4.2.3 Analyse ähnlicher Werkzeuge

Ein Modellierungswerkzeug ist zum Beispiel das Werkzeug ADTB (ADVanced Design ToolBox) von der Firma MENTOR GRAPHICS. Das Werkzeug beinhaltet laut Dokumentation folgende Funktionen:

- Schaltungscharakterisierung und Datenblattgenerierung
- Robustheit- und Effizienzanalyse
- Anpassung von Verhaltensmodellen

Das Werkzeug bietet eine umfangreiche Oberfläche zur Schaltungscharakterisierung. Aus den Ergebnissen können Datenblätter generiert, Verhaltensmodelle erstellt, Analysen über ungünstige Fälle (worst case) durchgeführt und kritische Schaltungsparameter bestimmt werden. Das Werkzeug ist aber schlecht erweiterbar. So ist das Hinzufügen weiterer Messschaltungen und Modelle eingeschränkt und die Erweiterung für andere Modellierungsmethoden nicht möglich.

Ein anderes Modellierungswerkzeug dieser Art ist das Werkzeug VSdE/VCME von Cadence. *Cadence Virtuoso Specification-driven Environment (VSdE),* früher Aptivia, ist eine Entwurfs- und Simulationsumgebung. Mit VSdE können eine Vielzahl von Verifikationsaufgaben automatisiert werden. Zur Erstellung von Modellen dient das zugehörige Werkzeug *Virtuoso Characterization & Modeling Environment (VCME). VCME* automatisiert die Charakterisierung von Schaltungen auf Transistorebene und erstellt aus den extrahierten Daten äquivalente Tabellenmodelle in Verilog-A.

Neben den hohen Anschaffungskosten solcher Werkzeuge ist vor allem die Abhängigkeit von den Herstellern ein großer Nachteil. So basieren diese Werkzeuge meist auf anderen

Werkzeugen des gleichen Herstellers. Die Bindung an die Hersteller ist demzufolge sehr stark. Ziel bei der Entwicklung eines eigenen Werkzeuges soll sein, sich an Funktionen vorhandener Werkzeuge zu orientieren, dabei aber flexibel und beliebig erweiterbar zu bleiben.

4.2.4 Problem- und Zielanalyse

Stakeholder

Die Definition so genannter „Stakeholder“ im Softwareentwurf dient dazu, verschiedene Anforderungen der Anwendung zu verdeutlichen. Dazu werden unterschiedliche Benutzergruppen, die Stakeholder (Interessenvertreter), definiert. Jeder Stakeholder hat unterschiedliche Aufgaben, Vorraussetzungen, Probleme und Ziele. Für dieses Projekt werden die drei folgenden Stakeholder definiert:

- Modellierungsexperte
- Schaltungsentwickler
- Softwareentwickler

Aufgaben

Der Modellierungsexperte entwirft Modellvorlagen für eine Klasse von analogen Schaltungen. Zur Parametrisierung der Modelle erstellt er zusätzlich zugehörige Testbenches, bestehend aus Messschaltungen und Extraktionsgleichungen.
Der Schaltungsentwickler hat die Aufgabe, die entworfene Schaltung mittels abstrakter Modelle zu verifizieren.
Der Softwareentwickler erweitert im Nachhinein das Werkzeug, zum Beispiel durch weitere Modellierungsmethoden, andere Simulatoren, Skriptsprachen etc.

Voraussetzungen

Der Modellierungsexperte ist Experte in der Modellierung analoger Schaltungen. Er kennt sich mit Hardwarebeschreibungssprachen, Skriptsprachen, Simulatoren, analogen Schaltungen etc. aus. Er hat jedoch erfahrungsgemäß in der Verwaltung größerer Softwareprojekte wenig Kenntnis.
Der Schaltungsentwickler dagegen hat zwar Expertenwissen über das Verhalten analoger Schaltungen, kennt sich jedoch kaum mit Skript- oder Hardwaresprachen und entsprechenden Simulatoren aus.
Der Softwareentwickler hat in vielen Fällen unzureichendes Wissen über den Schaltungsentwurf, kennt sich dafür aber in Methoden der Softwareentwicklung aus.

Probleme und Ziele

Ein Entwurf allgemeingültiger Modelle ist sehr schwierig. Der Modellierungsexperte muss Schaltungseigenschaften kennen und sich passende Modelle vorstellen können. Der Entwurf zugehöriger Messanlagen ist ebenfalls aufwändig, langwierig, demzufolge insgesamt sehr langsam und teilweise unübersichtlich. Die Modell- und Messanlagenentwürfe sowie das Anlegen entsprechender Bibliotheken sollen durch einfache und übersichtliche Methoden unterstützt werden.

Der Schaltungsentwickler ist hauptsächlich mit seinem Projekt beschäftigt. Die Modellierung von Teilschaltungen ist eine Nebenaufgabe, die möglichst schnell und fehlerfrei gelöst werden muss. Mit wenigen Arbeitsschritten möchte der Schaltungsentwickler ein Modell erstellen, welches die wichtigsten Eigenschaften seiner Schaltung nachbildet und eine geringe Simulationszeit hat.

Der Softwareentwickler entwickelt und erweitert konsistente, fehlerfreie, einfache und übersichtliche Software.

4.2.5 Systemanalyse

Programmfunktionen

Mit Hilfe von Anwendungsfalldiagrammen werden die aus der *Problem- und Zielanalyse* abgeleiteten Programmfunktionen gezeigt. Anwendungsfalldiagramme (Use-Case Diagramme) werden häufig in der Softwareentwicklung verwendet. Sie sind durch die Modellierungssprache UML (Unified Modeling Language) standardisiert. Die Diagramme zeigen die definierten Stakeholder und deren Aufgaben, die durch das Programm unterstützt werden sollen.

Das erste Anwendungsfalldiagramm, vgl. *Abbildung 22,* stellt einen allgemeinen Funktionsüberblick dar. *Abbildung 23* zeigt notwendige Funktionen im Bereich der Schaltungsmodellierung und *Abbildung 24* zeigt Funktionen zur Verwaltung von Modellen und Messanlagen.

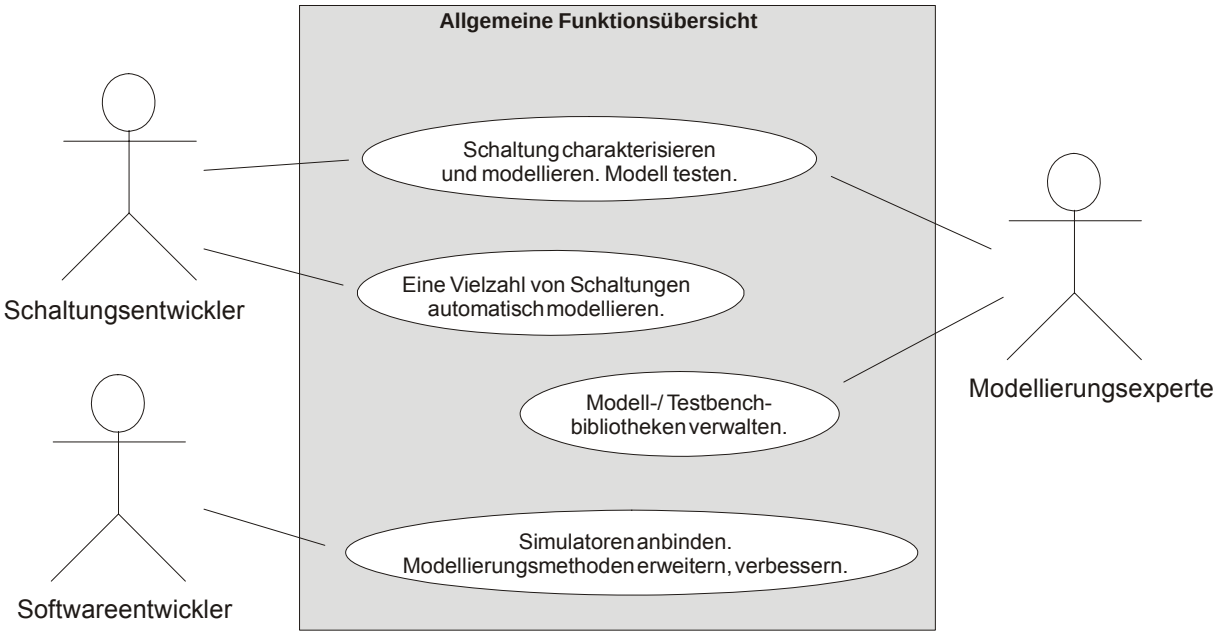

Abbildung 22 – Anwendungsfalldiagramm zur allgemeinen Funktionsübersicht

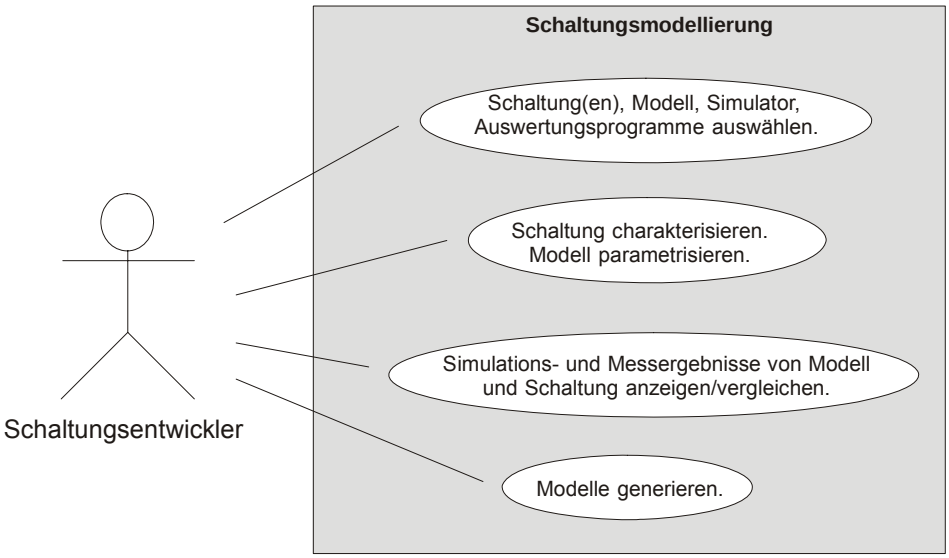

Abbildung 23 – Anwendungsfalldiagramm zur Schaltungsmodellierung

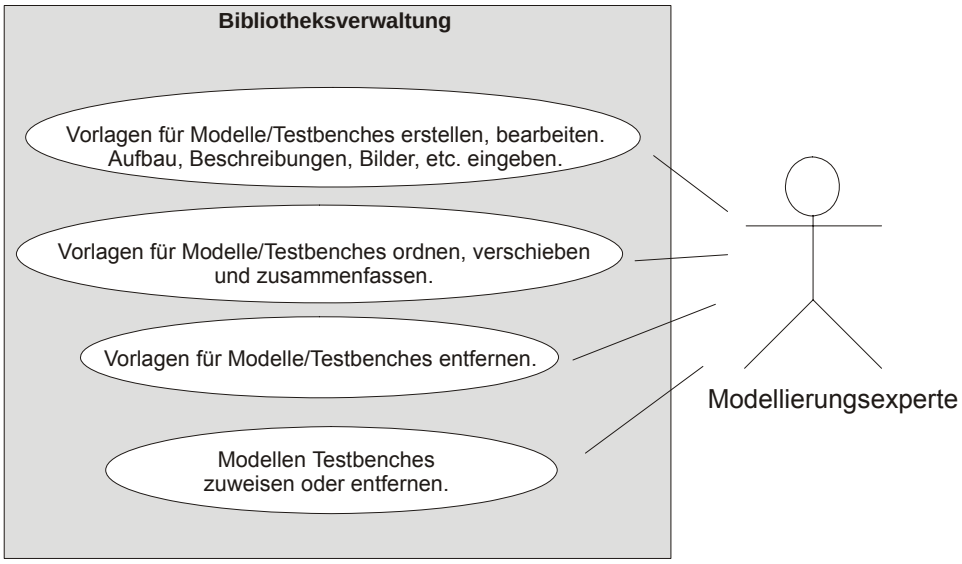

Abbildung 24 – Anwendungsfalldiagramm zur Bibliotheksverwaltung

Datenverarbeitungsprozess

Abbildung 25 zeigt den Datenverarbeitungsstrom des Programms. Aus einer Schaltung wird durch Hinzufügen einer Modellvorlage und Parametrisierung das Modell der Originalschaltung generiert. Die Parametrisierung erfolgt über Messschaltungen und Skripte zur Extraktion bestimmter Eigenschaften aus der Originalschaltung.

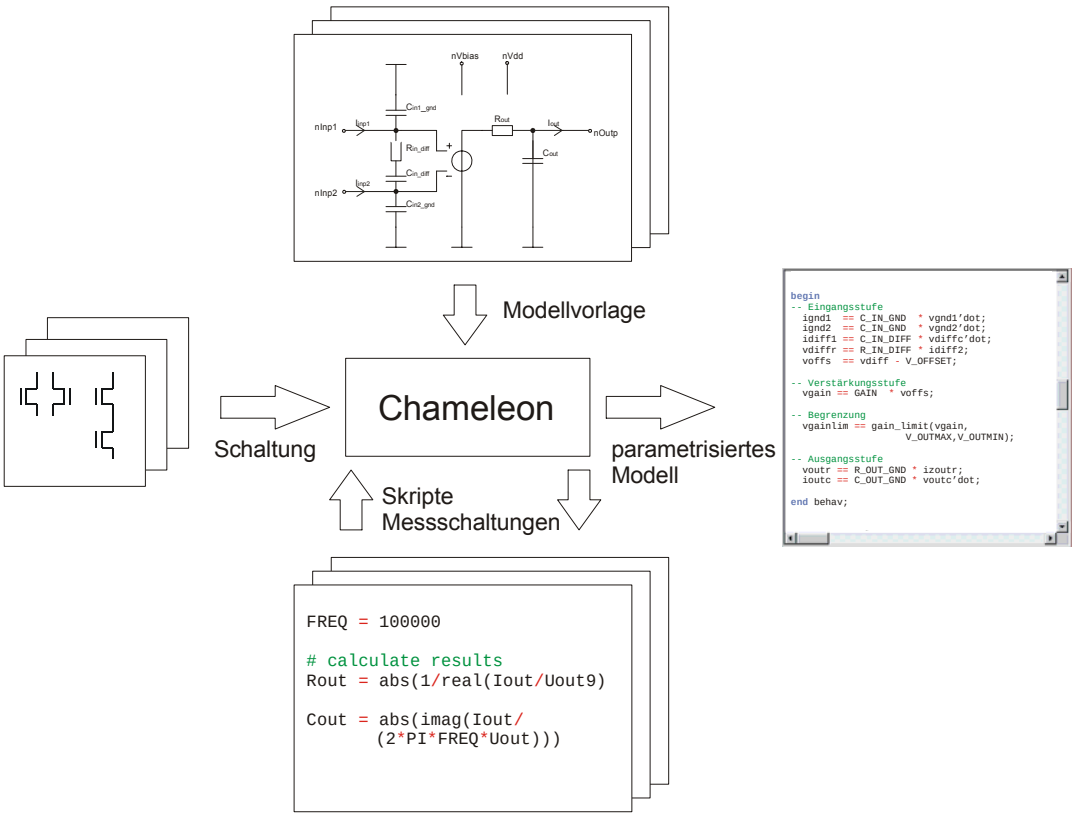

Abbildung 25 – Chameleon Datenverarbeitungsprozess

4.2.6 Anforderungsspezifikation

Eine Anforderungsspezifikation bildet im Softwareprozess oft die Vertragsgrundlage zwischen Kunde und Softwareentwickler. In dieser Arbeit dient sie als Grundlage sämtlicher Entwurfsentscheidungen. Es erfolgt üblicherweise eine Einteilung in funktionale und nichtfunktionale Anforderungen. Dabei sind die funktionalen Anforderungen konkrete Funktionen, die das Programm unterstützen. Nichtfunktionale Anforderungen sind eher Gütekriterien des Programms.

Aus der Aufgabenstellung, der Problem- und Zielanalyse sowie den Anwendungsfalldiagrammen lässt sich folgende Anforderungsspezifikation ableiten. Die einzelnen Punkte sind mit *(A1), (A2), (A3),* … nummeriert. In den nächstfolgenden Kapiteln wird darauf zurückgegriffen.

Funktionale Anforderungen

(A1) Automatische Ermittlung und Darstellung charakteristischer Kennwerte von Operationsverstärkern; Darstellung durch Ausgabe einzelner Wertepaare und graphischer Anzeige von Wertefolgen

(A2) Parametrisierung einer Modellschaltung aus den Kennwerten

(A3) Ermittlung von Abweichungen der Kennwerte von Modell und Originalschaltung und Vergleich der Simulationszeit

(A4) Anbindung eines Simulators, Möglichkeit der Anbindung weiterer Programme und Hardwarebeschreibungssprachen

(A5) Bereitstellung einer Modell- und Messanlagenbibliothek

(A6) Möglichkeit der Erweiterung und Änderung der Modellbibliothek und der zugehörigen Testbenchbibliothek durch den Anwender

(A7) Anpassung der Anzeigeoptionen von Modellen und Testbenches

(A8) Modellgenerierung nach bestimmten Eingaben

Nichtfunktionale Anforderungen

Tabelle 2 zeigt die Gewichtung der nichtfunktionalen Anforderungen.

Für die Portabilität ist in erster Linie die Unterstützung verschiedener Betriebssysteme und Betriebssystemversionen ausschlaggebend. Dabei sollen verschiedenste UNIX-Systeme standardmäßig und unterschiedliche Kernelversionen unterstützt werden, da gängige Software

für den Schaltungsentwurf meist für SOLARIS konstruiert ist. Wenn möglich soll das Programm zusätzlich für Windows Systeme portierbar sein. Ein weiteres Portabilitäts-kriterium ist die Anpassung der Oberfläche an vorhandene Fenstersysteme, wie zum Beispiel CDE (Common Desktop Environment), Gnome (GNU Network Object Model Environment), KDE (K-Desktop-Environment) etc. So kann durch eine Unterstützung bekannter Fenstersysteme die Nutzerakzeptanz erhöht werden.

Die Geschwindigkeit des Programms spielt eine untergeordnete Rolle. Es sollte natürlich auf gängigen Rechnern in akzeptabler Geschwindigkeit laufen, muss aber keine Echtzeitaufgaben leisten, wie zum Beispiel Reaktionen auf kritische Situationen.
Die Bedienbarkeit stellt oft ein Akzeptanzkriterium für den Anwender dar. Sie sollte möglichst einfach, gut strukturiert, informativ und anpassbar sein. Daraus resultiert, dass eine übersichtliche Datenstrukturierung ebenfalls einen wichtigen Faktor darstellt, besonders für das Verwalten der Bibliotheken.
Eine sehr gute Erweiterbarkeit wurde bereits in der Aufgabenstellung gefordert, insbesondere die Möglichkeit der Erweiterung der Modellierungsverfahren.

	sehr relevant	**relevant**	**wenig relevant**
(A9) Portabilität		x	
(A10) Geschwindigkeit			x
(A11) Bedienbarkeit		x	
(A12) Strukturierungsmöglichkeiten	x		
(A13) Erweiterbarkeit	x		

Tabelle 2 – nichtfunktionale Anforderungen

4.3. Entwurf

4.3.1 Einleitung

Es gibt mehrere Möglichkeiten zur Realisierung der Anforderungen. In der Entwurfsphase werden einige unterschiedliche Lösungsvarianten vorgestellt und getroffene Design-entscheidungen begründet. In den ersten Unterkapiteln werden verwendete Sprachen und Software im Prototyp aufgezeigt. Als Nächstes erfolgt die Erläuterung von Daten- und Systemstruktur. Zur Visualisierung werden UML-Klassendiagramme verwendet.

Ein vollständiges Gesamtklassendiagramm schränkt durch seine Komplexität das Verständnis der Systemstruktur ein. Aus diesem Grund werden nur die wichtigsten Methoden und Eigenschaften der Klassen gezeigt. Der Systemaufbau ist zusätzlich in einzelne Diagramme (Pakete) unterteilt, die jeweils eigene Aufgabengebiete abdecken.

In den folgenden Kapiteln befinden sich kurze Theorieteile gängiger Entwurfsmethoden. Solche Entwurfsmuster (Design Patterns) sind bewährte Lösungsansätze für verallgemeinerte Problemstellungen im Softwareentwurf. Diese Entwurfsmuster sind insbesondere auf Flexibilität und Erweiterbarkeit der Software ausgelegt. Zusätzlich dienen sie zur Verallgemeinerung der Softwarestruktur durch Zusammenfassung gleicher Systemstrukturen und behalten dadurch die Übersichtlichkeit. Es heißt, dass ein System, welches aus zehntausend Klassen besteht, selbst für erfahrene Entwickler schwer zu beherrschen ist. Besteht dessen Klassenstruktur jedoch aus zweihundert bekannten Entwurfsmustern, so bleibt die Software überschaubar.

4.3.2 Untersuchung analoger Hardwarebeschreibungssprachen

Da das Programm später mehrere Hardwarebeschreibungssprachen unterstützen soll, vgl. *(A4)*, werden nur zwei verschiedene Sprachen zur Erstimplementierung verglichen: VHDL-AMS und SPICE. Die laut Aufgabenstellung zu modellierenden Operationsverstärker-schaltungen sind in VHDL-AMS gegeben. Aus diesem Grund wurde diese Hardware-beschreibungssprache ausgewählt. Zusätzlich soll VHDL-AMS noch mit SPICE verglichen werden, da SPICE relativ leicht verständlich ist, und dadurch eine Verbesserung der *Bedienbarkeit (A11)* und *Strukturierungsmöglichkeiten (A12)* vorstellbar ist.

Tabelle 3 zeigt die Vor- und Nachteile der beiden Sprachen. Die meisten Vor- und Nachteile sind durch die Klassifizierung der Sprachen mit SPICE als Netzlistenbeschreibungssprache und VHDL-AMS als Verhaltensbeschreibungssprache begründet. In einer Netzlisten-beschreibungssprache werden die Schaltungen durch Zusammenfügen einzelner Grund-elemente, wie zum Beispiel Widerstände, Quellen, Kondensatoren etc., beschrieben. In einer Verhaltensbeschreibungssprache werden die Zusammenhänge von Spannungen und Strömen in den Schaltungen mit Hilfe mathematischer Gleichungen beschrieben.

Der große Vorteil von SPICE besteht in seiner Einfachheit, insbesondere durch die einfache, kompakte Syntax der Netzlistendarstellung. Für den Modellierungsexperten würden sich bei

Verwendung von SPICE der Aufwand bei der Erstellung neuer Schaltungsvorlagen verringern und die Verständlichkeit vorhandener Schaltungsvorlagen erhöhen. Ein weiterer kleinerer Vorteil ist, dass SPICE-Dateien sofort von einem Simulator simuliert werden können, während VHDL-AMS Dateien vorher erst kompiliert werden müssen. Durch die vielen Kompilationsdateien ist bei VHDL-AMS der Dateiaufwand größer und unübersichtlicher.
Ein weiterer Nachteil von VHDL-AMS ist, dass noch keine standardisierte Bibliothek existiert. Einerseits kann man jedoch seine eigene Bibliothek erstellen, und andererseits wird es in der Zukunft sicher größere Bibliotheken geben.

Modelle sind meist sehr abstrakt und haben oft nichts mehr mit dem eigentlichen Aufbau des Originals gemein. Darum müssen teilweise komplizierte, in der Natur unübliche Zusammenhänge realisiert werden. Als Netzlistenbeschreibungssprache ist SPICE wegen seiner festen Komponentenbibliothek in der Beschreibung solcher Zusammenhänge eingeschränkt. Dagegen bietet VHDL-AMS als Verhaltensbeschreibungssprache durch die beliebige Formulierung von Gleichungen wesentlich mehr Möglichkeiten in diesem Bereich. Diese Eigenschaft ist ein weiterer großer Vorteil von VHDL-AMS gegenüber SPICE. Mit Hilfe der Verhaltensbeschreibung ist es sehr einfach, komplizierte Zusammenhänge, wie zum Beispiel das Übertragungsverhalten, in die Modellschaltung zu integrieren. Entsprechende Beschreibungen in SPICE sind wesentlich umfangreicher und komplizierter.

	SPICE	**VHDL-AMS**
Mächtigkeit	+ feste Komponenten-bibliothek	++ Komponentenbibliothek leicht erweiterbar – keine Standardbibliotheken vorhanden
Codeaufwand	++ gering, einfach	– sehr groß, umständlich
Dateiaufwand	+ gering	– hoch, unübersichtlich
Übersetzungsaufwand	++ nicht vorhanden	– vorhanden
Verwendung Modellparameter	– schwierig	++ einfach

Tabelle 3 – Vergleich analoger Hardwarebeschreibungssprachen

Fazit: Für die Modellbeschreibung ist VHDL-AMS geeigneter als SPICE. Falls der verwendete Simulator die Anbindung verschiedener Verhaltensbeschreibungssprachen unterstützt, bietet sich zusätzlich die Verwendung von SPICE für die Messschaltungen an.

4.3.3 Untersuchung der Simulations- und Auswertungsvarianten

Am Fraunhofer Institut Integrierte Schaltungen werden Simulationswerkzeuge aus der Produktfamilie von MENTOR GRAPHICS verwendet. Die Werkzeuge aus dieser Produktfamilie sollen für die Anbindung an das Werkzeug genutzt werden. Die Simulatoren von MENTOR GRAPHICS erstellen binäre Datendateien der Kennlinien aus der Simulation. Das Format dieser Dateien ist unbekannt. Demzufolge muss zur Auswertung, d.h. zur Extraktion einzelner Messkennwerte, ebenfalls die Software von MENTOR GRAPHICS verwendet werden.
Im folgenden Abschnitt werden die verschiedenen Möglichkeiten zur Simulation und Auswertung aus der MENTOR GRAPHICS Produktfamilie verglichen.

Entwicklungsbedingt hat MENTOR GRAPHICS zwei Analogsimulatoren herausgebracht. Einen älteren Simulator, ELDO, und eine neuere Version, ADVanceMS.
Tabelle 4 zeigt Vor- und Nachteile der beiden Simulatoren. Der große Vorteil des neueren Simulators ADVanceMS ist, dass er mehrere Schaltungsbeschreibungssprachen und deren gemischte Nutzung unterstützt. Verbunden mit den Erkenntnissen aus *4.3.2* sind mit diesem Simulator die Anforderungen *(A11) Bedienbarkeit* und *(A12) Strukturierungsmöglichkeiten* wesentlich besser erfüllt. Dieser Vorteil gleicht den Nachteil der längeren Simulationszeiten von ADVanceMS aus, vgl. Wichtung der Anforderungen *(A10)* mit *(A11)* und *(A12)*.
Ein weiterer Nachteil ist, das ADVanceMS noch nicht alle Simulatorsteuerbefehle des Vorgängers ELDO übernommen hat. Das betrifft jedoch nur wenige Befehle, die auch mit Hilfe anderer Befehle nachgebildet werden können. Ein problematischer Befehl war der Parameter SWEEP Befehl. Mit diesem kann man während der Simulation einen Parameterwert durchlaufen lassen (zum Beispiel Eingangsspannung von 0V auf 1V). Das Problem kann aber genauso mit einer PWL Quelle (Quelle mit linearer Funktion) in SPICE oder einer eigenen entsprechenden Quelle in VHDL-AMS gelöst werden.

Es gibt einen kostenlosen Simulator für Windows namens SystemVision von MENTOR GRAPHICS, der Ähnlichkeiten zu ADVanceMS aufweist (kann von der Internetseite des Herstellers herunter geladen werden). Es ist eine reine Oberflächenanwendung und muss darum erst auf Tauglichkeit zur Programmanbindung untersucht werden. Darum soll dieser Punkt nur bedingt betrachtet werden.

	ELDO	**ADVanceMS**
unterstützte Schaltungs-sprachen	+ SPICE	+ SPICE + VHDL-AMS + Verilog-AMS + Sprachen gemischt
Simulations-eigenschaften	+ Viele, mächtige Befehle zur Simulationssteuerung + geringe Simulationszeit	– Befehlssatz zur Simulationssteuerung ist eingeschränkt/unausgereift – Teilweise längere Simulationszeiten
Ergebnis-auswertung	+ ".EXTRACT" Befehl im Simulator + TCL-Postprocessor im Simulator + TCL-Postprocessor im Auswertungsprogramm EZWave	+ „.EXTRACT" Befehl im Simulator + TCL-Postprocessor im Auswertungsprogramm EZWave
Portabilität	+ Linux	+ Linux, evtl. Windows

Tabelle 4 – Vergleich analoger Simulatoren

Vor der Entscheidung für einen Simulator werden die verschiedenen Auswertungsvarianten zur Extraktion der Kennwerte verglichen, vgl. *Tabelle 5*. Dadurch wird festgestellt, ob es wirklich von Nachteil ist, dass der ADVanceMS Simulator weniger Varianten in diesem Bereich unterstützt.

Der „.EXTRACT" Befehl ist ein Simulatorsteuerungsbefehl, ähnlich .DC, .TRAN etc. Mit diesem und mit Hilfe einer weiteren Funktionsbibliothek von MENTOR GRAPHICS können bestimmte Werte aus den Kennlinien einer Schaltung extrahiert und in eine Datei geschrieben werden. Diese Funktionsbibliothek stellt zum Beispiel Funktionen zum Ermitteln von Maxima, Minima, Ableitungen, Integralen etc. aus den Kennlinien bereit. So sieht zum Beispiel die Ermittlung des komplexen Stromes über einer Spannungsquelle bei 100kHz folgendermaßen aus:

```
.EXTRACT AC label=I_100kHz file=result.txt yval(Ii(V1),100kHZ)
```

Der Vorteil dieser Variante liegt in der einfachen und schnellen Realisierung der Extraktionsgleichungen. Nachteile sind unter anderem die begrenzte Erweiterbarkeit der mathematischen Funktionsbibliothek sowie das starre Ausgabeformat.

Mit Hilfe des TCL-Postprocessors im ELDO-Simulator können Dateien der Skriptsprache TCL in die Schaltungsbeschreibung eingebunden, und mit der gleichen mathematischen Funktionsbibliothek wie beim „.EXTRACT“ Befehl kann auf die Kennlinien zugegriffen werden. Das Beispiel von oben würde dann folgendermaßen aussehen:

```
set I_100kHz [evalExpr [yval $wf_I,100000]]
```

Ein Vorteil dabei ist, dass zusätzlich TCL-Bibliotheken verwendet werden können, und so zum Beispiel das Schreiben in eine Datei selbst übernommen werden kann. Das bedeutet zwar mehr Codeaufwand, die Formatierungsmöglichkeiten sind aber beliebig. Die Erweiterung der mathematischen Bibliothek hingegen ist leider nur sehr eingeschränkt möglich, da die Skriptsprache TCL nicht für mathematische Berechnungen gedacht ist. Zahlen werden zum Beispiel nur als Strings gehandhabt. Daraus folgt, dass TCL nur wenige mathematische Basisfunktionen bereitstellt und auch in seiner Genauigkeit stark eingeschränkt ist.

TCL lässt sich zudem schwer lesen, da es sich in seiner Syntax von anderen Programmiersprachen unterscheidet. Ein Beispiel ist die Syntax der Zuweisung von Variablen: „x = 2“ heißt in TCL:

```
Set x 2
```

Zusätzlich sind Syntax-Fehlermeldungen in TCL nicht sehr präzise, was die Fehlerfindung im eigenen Skript sehr erschwert.

Die dritte Variante ist die Verwendung des TCL-Postprocessors von EZWave. EZWave ist ein weiteres Programm von MENTOR GRAPHICS. Es ist ein grafisches Programm zur Darstellung und Auswertung von Simulationsergebnissen. Man kann es auch aus der Konsole heraus mit einem TCL-Skript starten. Das oben genannte Beispiel sieht dann folgendermaßen aus:

```
set I_100kHz evalExpression "yval(wf(\"<circuit/AC>Ii(V1)\"),100000)"
```

Der Codeaufwand ist wieder etwas gestiegen, demzufolge leidet die Übersichtlichkeit. Doch ein großer Vorteil ist, dass EZWave eine größere Funktionsbibliothek bereitstellt. So gibt es zum Beispiel noch zusätzliche Funktionen für das Rechnen mit komplexen Zahlen. Außerdem unterstützt EZWave die Fehlerermittlung durch eine bessere Fehlerausgabe in der Konsole.
Ein Nachteil bei EZWave besteht darin, dass es ein zusätzliches Programm ist. Das bedeutet mehr Implementierungsaufwand und eine Reduzierung der Geschwindigkeit.

	„.EXTRACT“ Befehl im Simulator	**TCL-Postprocessor im Simulator**	**TCL-Postprocessor im Auswertungsprogramm EZWave**
Ausgabeformat	– fest	+ beliebig	+ beliebig
Genauigkeit	+ exakt (immer auf 5 Stellen)	+ sehr exakt, aber umständlich	++ sehr exakt(ca. 17 Stellen)
Funktionsumfang	+ gut	+ gut	++ sehr gut
Erweiterbarkeit	– mäßig durch Macros	+ gut	+ gut
Fehlerermittlung	+ gut	– schlecht	+ gut
Codeaufwand	++ gering	– groß	– mittelmäßig
Geschwindigkeit	++ schnell	++ schnell	+ langsam
Sonstiges			+ extra Programm, muss extra nach Simulation aufgerufen werden

Tabelle 5 – Vergleich der Auswertungsvarianten

Fazit: Zur Erfüllung des funktionalen Anforderungspunktes *(A1),* der Ermittlung charakteristischer Kennwerte, ist das Vorhandensein einer umfangreichen mathematischen Bibliothek notwendig. Aus diesem Grund ist die Erweiterungsmöglichkeit der Bibliothek zur Auswertung der Kennlinien von großer Bedeutung. Darum sollte ein TCL-Postprocessor verwendet werden. Die Einschränkung in der *(A11) Bedienbarkeit* muss dabei in Kauf genommen werden.

Bei den Postprocessoren ist der von EZWave aufgrund seiner größeren Funktionsbibliothek und der besseren Fehlerermittlung der geeignetere.

Zusammenfassung

Der Simulator ADVanceMS wird zur Schaltungssimulation und das Programm EZWave zur Extraktion der Kennwerte verwendet. Die Modelle werden mit VHDL-AMS und die Messschaltungen mit SPICE beschrieben.

4.3.4 Untersuchung von Programmiersprachen

Zur Implementierung des Prototyps eignen sich auf den ersten Blick die beiden Sprachen TCL und JAVA. TCL würde sich eignen, da viele Simulatoren bereits TCL unterstützten. JAVA bietet sich an, da einige Simulatorsoftware bzw. im Fall von MENTOR GRAPHICS die

Auswertungssoftware EZWave in JAVA geschrieben ist, und somit vorhandene Bibliotheken verwendet werden könnten. *Tabelle 6* zeigt Vor- und Nachteile der beiden Sprachen.

TCL	**JAVA**
+ schnelle und einfache Erstellung von Oberflächen mit TCL/TK möglich	– Oberflächenerstellung teilweise umständlich, da Komponenten sehr abstrakt sind
+ Viele Simulatoren und Auswertungsprogramme unterstützen bereits TCL	+ Gute Anbindung an EZWave, da EZWave auch in Java geschrieben ist
– TCL ist im Kern prozedural und nur durch Erweiterungen objektorientiert, es hat nur eine schwache Typisierung, dadurch ist es teilweise unübersichtlich und fehleranfällig	+ Gute Strukturierung durch Objektorientierung und strenge Typisierung möglich
– Es gibt keine ausgereiften Entwicklungsumgebungen mit Funktionen zur Fehlerfindung	+ Sehr gute freie Entwicklungssoftware (Eclipse) vorhanden
– Der Code ist offen, da TCL eine Interpretersprache ist, und so leicht durch den Anwender lesbar	+ Anwendungen können teilweise dekompiliert werden, dies ist aber aufwändiger als bei einer reinen Interpretersprache
+ plattformunabhängig durch entsprechenden Interpreter	+ plattformunabhängig

Tabelle 6 – Vergleich Programmiersprachen

Zur Entwicklung der Hauptanwendung soll JAVA Verwendung finden, da die Anforderung der *Erweiterbarkeit (A13)* durch die Objektorientierung und die ausgereiften Entwicklungsumgebungen für JAVA sehr gut unterstützt wird.

TCL-Skripte können mit speziellen Bibliotheken in JAVA eingebunden werden. So käme evtl. die Verwendung von TCL zur Anbindung bzw. zur Kommunikation mit den Simulatoren in Frage. Kapitel *4.3.8 Entwurf zur Simulatoranbindung* zeigt *w*eitere Untersuchungen zu diesem Thema.

4.3.5 Entwurf zur externen Datenverwaltung

Varianten der Datenspeicherung

Zur Speicherung von Daten gibt es zwei Möglichkeiten: Erstens die Verwendung einer Datenbank oder zweitens die Speicherung der Daten in Dateien auf der Festplatte. Entscheidet man sich für die zweite Variante, muss zusätzlich entschieden werden, ob die Daten für den

Benutzer transparent sind oder nicht. Transparenz bedeutet, dass die Daten für den Anwender auch ohne das Programm lesbar und editierbar sind.

Eine Datenbank vereinfacht zwar die Datenhaltung für den Programmierer, würde aber vom Anwender verlangen, die entsprechende Software zusätzlich zu installieren. Das führt zu mehr Arbeit für den Anwender und evtl. zu Lizenzproblemen. Aus diesen Gründen ist die Speicherung der Daten auf Festplatte unerlässlich.
Für die Anwender ist es oft angenehmer, wenn sie ihre Daten, in diesem Fall zum Beispiel Simulationsergebnisse oder Simulatoreinstellungen, auch ohne Hilfe des Programms lesen und ändern können. Das bedeutet in diesem Fall offene Datenhaltung.

Datenstruktur auf der Festplatte

Gespeichert werden sollten:

- Modelle, Messschaltungen und Extraktionsbedingungen, die der Modellierungsexperte entwirft
- Zusatzdaten zu den Schaltungen, wie unterstützte Simulatoren, Parameternamen und Standardwerte für Parameter, evtl. verschiedene Varianten der Modelle, damit der Anwender später zum Beispiel die Genauigkeit seines Modells ändern kann
- das konkrete, parametrisierte Modell und evtl. die Ergebnisse aus den Simulationen der Messschaltungen
- Einstellungsdaten des Anwenders, wie zum Beispiel Genauigkeit, Parameterwahl, gewähltes Modell und gewählter Simulator

Prinzipiell unterscheidet man zwischen Daten, die für alle Programmanwender gelten und sichtbar sind, und zwischen den spezifischen Daten eines Anwenders. Aus diesem Grund werden in Zukunft die beiden Speicherorte Projektordner und Programmordner verwendet.
Der Programmordner ist der Ort auf der Festplatte, an dem das Programm installiert ist. Dort werden die Modelle, die Testbenches, deren Eigenschaften und globale Einstellungen gespeichert.

Den Projektordner wählt der Schaltungsentwickler aus, wenn er eine neue Schaltung modellieren will. In dem Ordner befinden sich Originalschaltungen, das fertige parametrisierte Modell, Simulationsergebnisse und persönliche Einstellungen. Simulatoren erstellen teilweise eine Vielzahl von Dateien bei einer Schaltungssimulation. Um

Speicherplatzproblemen vorzubeugen und Schreibrechte für die Simulatorsoftware zu sichern, sollen auch die Simulationen in diesem Verzeichnis stattfinden.

4.3.6 Oberflächen- und Dialogentwurf

MVC-Entwurfsmuster

Zur Oberflächendarstellung und zur Steuerung des Dialogablaufes soll das Entwurfsmuster Model-View-Controller, kurz MVC, verwendet werden. Vorteil des MVC-Musters ist die strikte Trennung von Daten (Model), deren Darstellung (View) sowie der Steuerung von Daten und Darstellung (Controller). Übersichtlichkeit und Wartung der Software werden damit gefördert und die Anforderung de *(A13) Erweiterbarkeit* erfüllt. Zusätzlich wird die Wiederverwendung einzelner Komponenten ermöglicht und der Programmieraufwand gemindert.

Ursprünglich kommt das MVC-Muster aus der Entwicklung von Web-Anwendungen, findet aber wegen seiner guten Eigenschaften auch auf anderen Gebieten Einsatz.

Abbildung 26 zeigt die Struktur des Musters, in das der Prototyp implementiert werden soll. Die Anzeige stellt den Zustand des Modells dar und informiert seinen Controller bei Benutzereingaben, wie zum Beispiel Mausklicks oder Tastatureingaben. Der Controller reagiert auf diese Ereignisse, ändert das entsprechende Modell oder zeigt gewünschte Dialoge an. Bei einer Änderung des Modells informiert der Controller seine zugehörigen Anzeigen zur Aktualisierung der Datenanzeige.

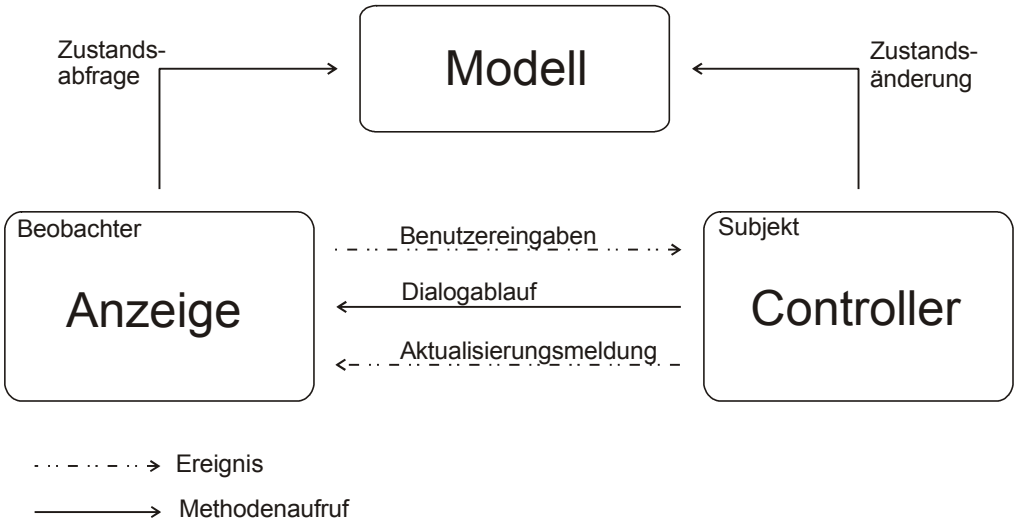

Abbildung 26 – Model-View-Controller-Architektur [6]

Beobachter-Entwurfsmuster

Zur Realisierung der Aktualisierungsfunktionen der grafischen Anzeige soll das Beobachter-Entwurfsmuster (Observer) verwendet werden, vgl. *Abbildung 27*. Dabei ist die Anzeige der Beobachter und der Controller das Subjekt. Konkrete Beobachter sind dann zum Beispiel Fenster und Dialoge. Dem Controller können mehrere Anzeigen angemeldet werden. Die registrierten Anzeigen werden dann durch Aufruf der Methode „Aktualisiere()“ durch den Controller angepasst.

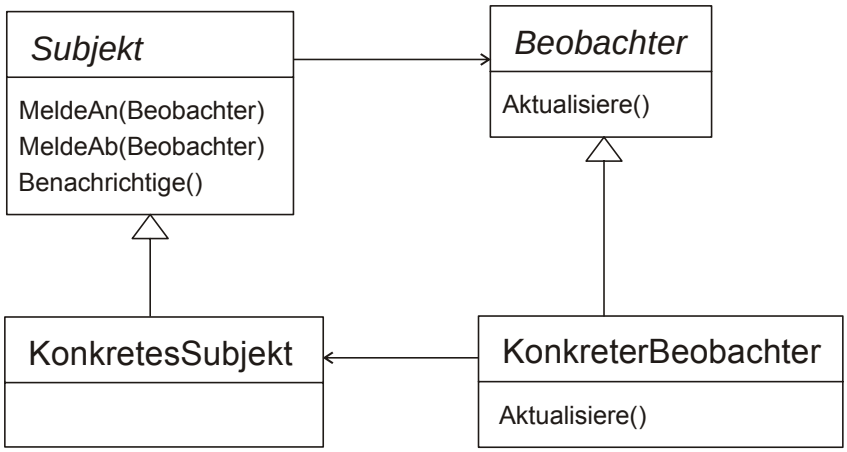

Abbildung 27 – Beobachter Entwurfsmuster [7]

Klassendiagramm zur Oberflächen- und Dialogstruktur

Abbildung 28 zeigt das Klassendiagramm der Schnittstellen zur Oberflächen- und Dialogsteuerung des Werkzeuges Chameleon. Auf der Basis dieses Klassendiagramms sind sämtliche Fenster und Dialoge des Programms implementiert. Da der Begriff Modell aus dem MVC-Muster zu möglichen Verwechslungen führt (vgl. Begriff Modell aus *2.2 Schaltungsmodellierung*), wird stattdessen der Begriff Datenobjekt (DataObject) verwendet. Wie im Klassendiagramm gekennzeichnet, besteht die Funktionsweise des Diagramms aus dem MVC- und dem Beobachter-Entwurfsmuster. Zur Realisierung des Beobachter-Entwurfsmusters werden die bereits in Java vorhandenen Klassen Observer (Beobachter) und Observable (Subjekt) verwendet.

Zusätzlich zur Oberflächenkomponentenverwaltung gibt es vier konkrete Ansichten. Dabei besteht die Hauptanwendung (FrameView) aus Dialogen (DialogView) und Hauptansichten (PageView). Zur besseren Wiederverwendbarkeit grafischer Komponenten bestehen die Dialoge und Seiten aus einzelnen Elementen (PanelView), wie zum Beispiel Bilddarstellung, strukturierte Textanzeige oder Darstellung von Graphen.

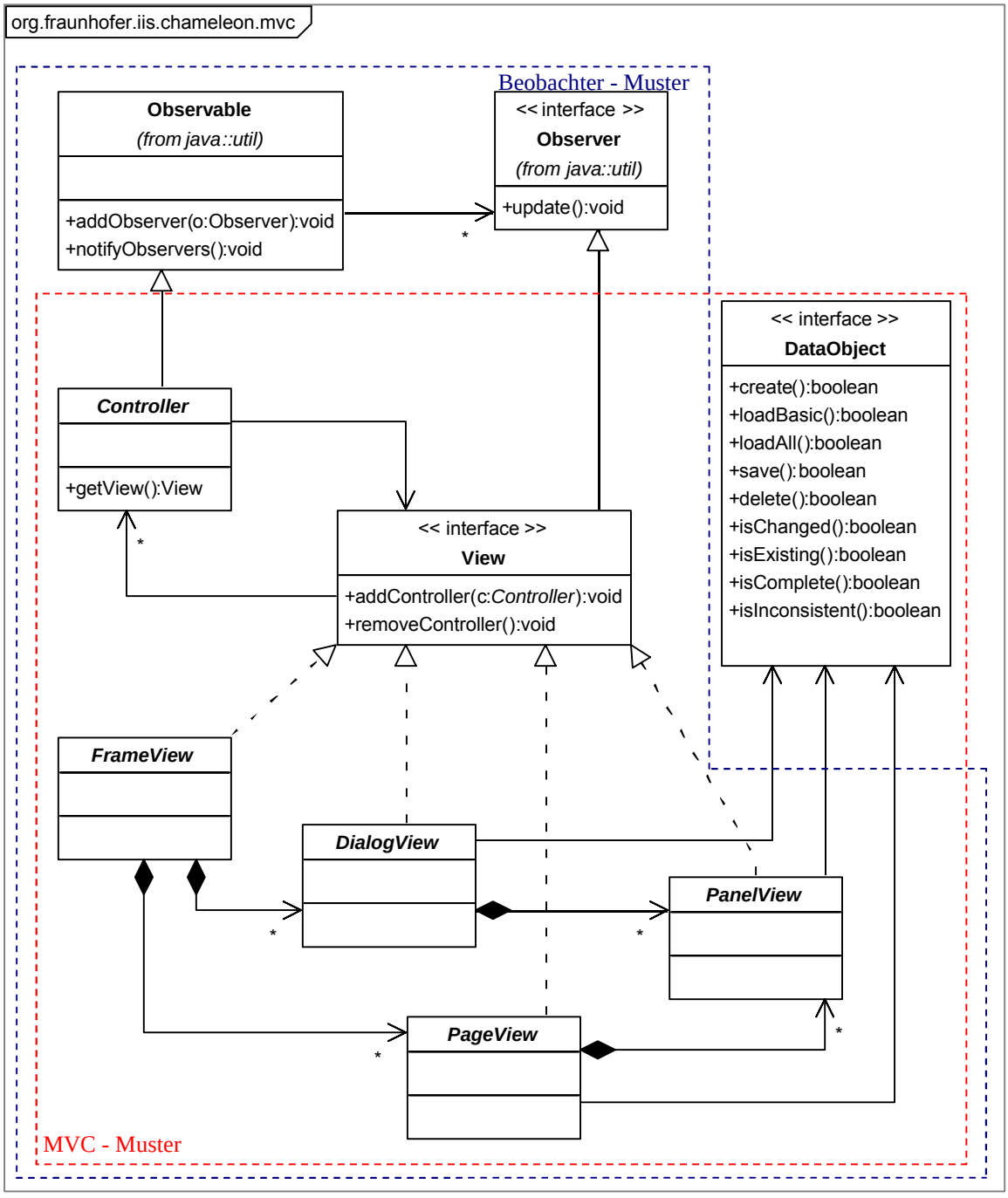

Abbildung 28 – Klassendiagramm zur Dialogsteuerung

Das Klassendiagramm zur Darstellung der konkreten Oberflächen- und Dialogsteuerung des Programms wird hier nicht angegeben, da es im Prinzip keine neuen Strukturen aufweist. Es ist konsequent nach oben beschriebenem MVC-Modell implementiert.

Grafischer Entwurf der Oberfläche

Für die einzelnen Fenster und Dialoge wurden vor der Implementierung Skizzen angefertigt. Nach den Skizzen wurde die Oberfläche erstellt. In *4.4 Implementierung* werden Bildschirmfotos des fertigen Programms, entsprechend den angefertigten Skizzen, gezeigt.
Für die Oberflächenkomponenten gibt es unterschiedliche Frameworks in JAVA. Das sind AWT, SWING und SWT. AWT von der Firma SUN ist das älteste dieser Frameworks und findet heute kaum mehr Verwendung.
SWT von der Firma IBM ist recht jung und modern. Vorteil von SWT ist, dass es sich in seinem Aussehen automatisch der gegebenen Fensterumgebung anpasst und damit die Anforderung *(A9) Portabilität* gut erfüllt. Die Bibliothek muss aber zusätzlich mit installiert werden.
SWING von der Firma SUN ist dagegen bereits in der Standard-Java-Bibliothek enthalten, und es kann ein beliebiges Aussehen gewählt werden, so dass man zum Beispiel das CDE-Aussehen auch auf einem Windowssystem verwenden kann. Aus diesen Gründen soll SWING Anwendung finden.

In *(A1)* wurde eine graphische Anzeige der Messkennlinien gefordert. Zur Darstellung von Graphen gibt es eine Vielzahl von freien und kostenpflichtigen JAVA-Bibliotheken. Praktischerweise soll das MScope Framework vom Fraunhofer Institut dafür verwendet werden.

4.3.7 Entwurf zur programminternen Datenverwaltung

Wie unter *4.3.6* bereits erläutert, soll eine strikte Trennung von Datenstruktur, Dialog- und Anzeigeebene mit Hilfe des MVC-Entwurfsmusters realisiert werden. In diesem Kapitel werden dazu alle notwendigen Datenobjekte der zukünftigen Anwendung betrachtet.

Entwurfsmuster Schablonenmethode

Zur Erfüllung der Anforderungen *(A4) Anbindung weiterer Simulatoren und Hardware-beschreibungssprachen* und *(A13) Erweiterbarkeit* wird das Entwurfsmuster Schablonen-methode (TemplateMethod) verwendet, vgl. *Abbildung 29*. Es existiert eine Schnittstelle, die Abstrakte Klasse, die bestimmte Methoden definiert. Konkrete Objekte, die von der Schnittstelle erben, müssen diese Funktionen implementieren. Die Anwendung verwendet nun die definierten Funktionen zur Lösung einer Aufgabe. Dabei kann die Anwendung arbeiten,

ohne die konkreten Klassen zu kennen. Eine Erweiterung durch Erstellung neuer Varianten der Aufgabenlösung ist somit ohne viel Aufwand möglich.

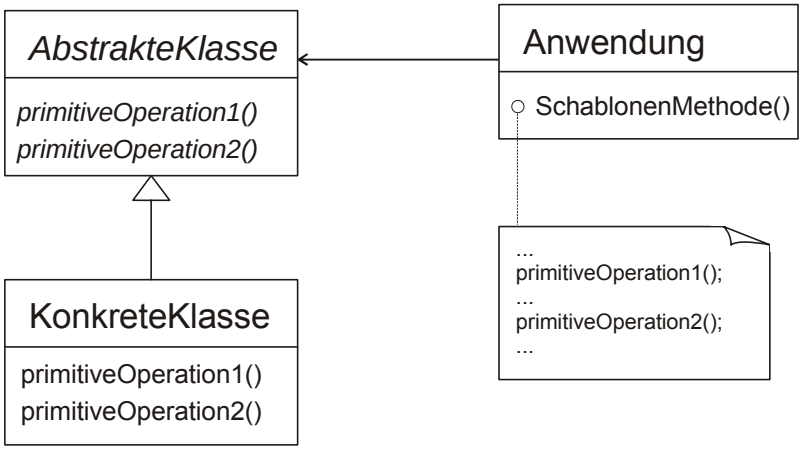

Abbildung 29 – Entwurfsmuster Schablonenmethode [7]

Das allgemeine Datenobjekt (vgl. DataObject aus *Abbildung 28*) ist bereits Teil dieses Entwurfsmusters. Die abstrakten Methoden speichern, laden, löschen etc. werden von der Anwendung benutzt, ohne dass diese die konkreten Daten kennt. Dabei können verschiedenste Objekte hinter dem Datenobjekt stecken, wie zum Beispiel eine Textdatei oder ein komplettes Projekt.

Basisobjekte zur Ein- und Ausgabe

Das Klassendiagramm in *Abbildung 30* stellt alle Objekte dar, die zur Ein- und Ausgabesteuerung des Programms dienen.

Der Dateimanager (FileManager) bietet Hilfsfunktionen wie zum Beispiel Kopieren oder Verschieben zur Dateiverwaltung.

Das Objekt „Einfache Datei“ (SimpleFile) bietet grundlegende Funktionen zum Lesen und Schreiben von Textdaten in eine Datei. Davon sind dann alle weiteren Dateitypen abgeleitet. Die Konfigurationsdatei (ConfigFile) bietet Funktionen zum Setzen und Laden von Einstellungsdaten. Die Ergebnisdatei (ResultFile) und die Funktionsdatei (WaveFile) stellen Funktionen zur Verwaltung von Simulationsergebnisdaten bereit.

Die Kodedatei (CodeFile) mit ihren Kindern implementiert erneut das Entwurfsmuster Schablonenmethode. Sie stellt somit eine Schnittstelle zu Dateitypen von verschiedenen Skriptsprachen, wie zum Beispiel VHDL, SPICE, Python dar. Sollen später weitere Hardwarebeschreibungssprachen unterstützt werden, muss nur eine entsprechende Klasse angelegt werden, die von der Schnittstelle Kodedatei abgeleitet ist und die geforderten

Funktionen implementiert. Dabei soll die Methode „getParameter()“ alle Parameternamen und -werte, und die Methode „getVariableNames()“ alle Namen der Variablen aus dem Skript ermitteln. Parameter sind Betriebs- und Extraktionsparameter, die vor einer Simulation geändert werden können. Variablen sind Messwerte, die nach einer Simulation abgefragt werden können. Die Methode „includeFile()“ soll das Skript so erweitern, dass ein anderes Skript in die Datei eingebunden wird. Das wird u. a. bei der Einbindung von beliebigen Schaltungen in eine Messschaltung benötigt.

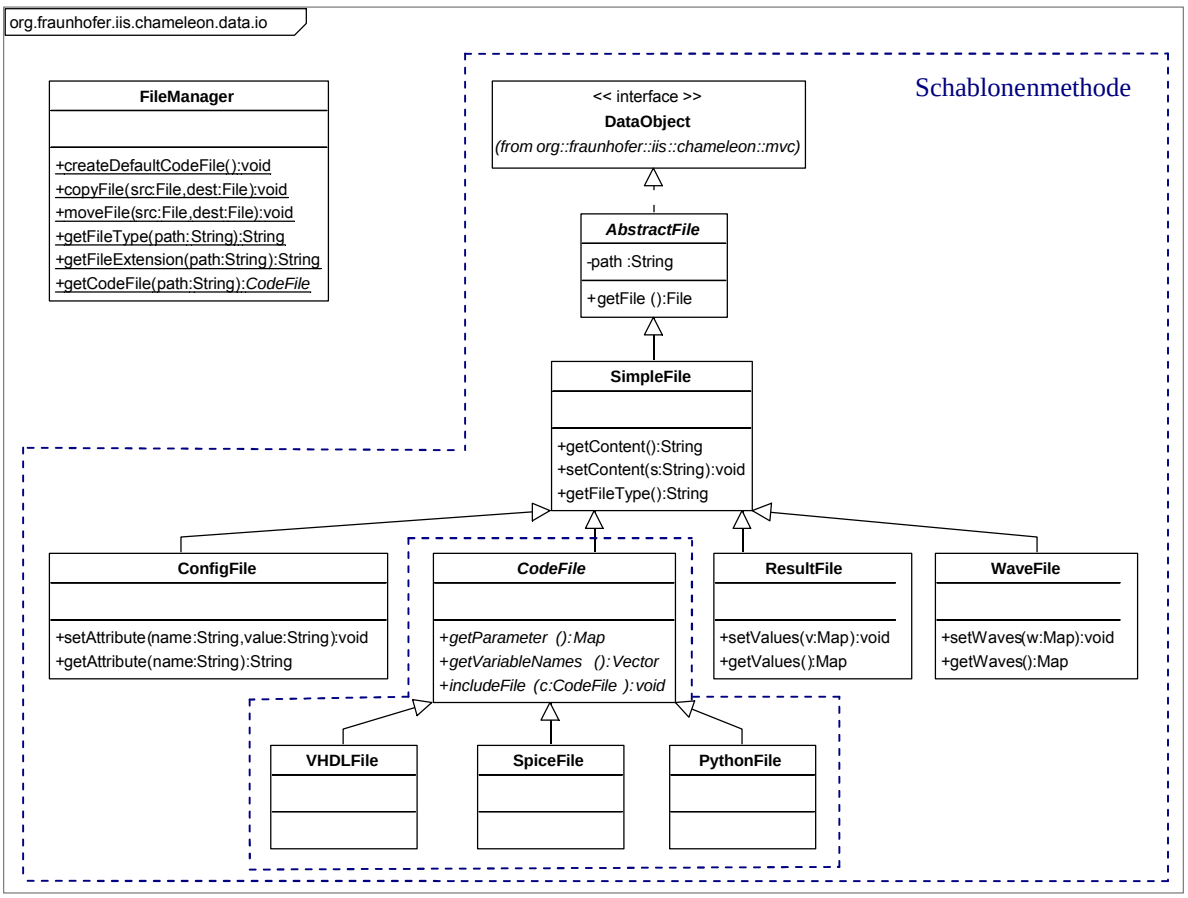

Abbildung 30 – Klassendiagramm zur Ein- und Ausgabe

Entwurfsmuster Kompositum

Zur Erfüllung der Anforderung *(A5)* und *(A6) Bereitstellen einer erweiterbaren Testbench- und Modellbibliothek*, bzw. zum Anbieten von *(A12) Strukturierungsmöglichkeiten* dieser Bibliothek, findet das Entwurfsmuster Kompositum Verwendung. Mit diesem Muster können relativ einfach rekursive Datenstrukturen realisiert werden. Dadurch kann die Bibliothek als Baumstruktur mit beliebigen Ordnern und Unterordnern dargestellt werden. *Abbildung 31* zeigt das Klassendiagramm des Entwurfsmusters. Das Kompositum, der Knoten in der

Baumstruktur, besteht aus mehreren Komponenten. Dabei können Komponenten wieder ein Kompositum darstellen oder ein Blatt am Ende des Zweiges.

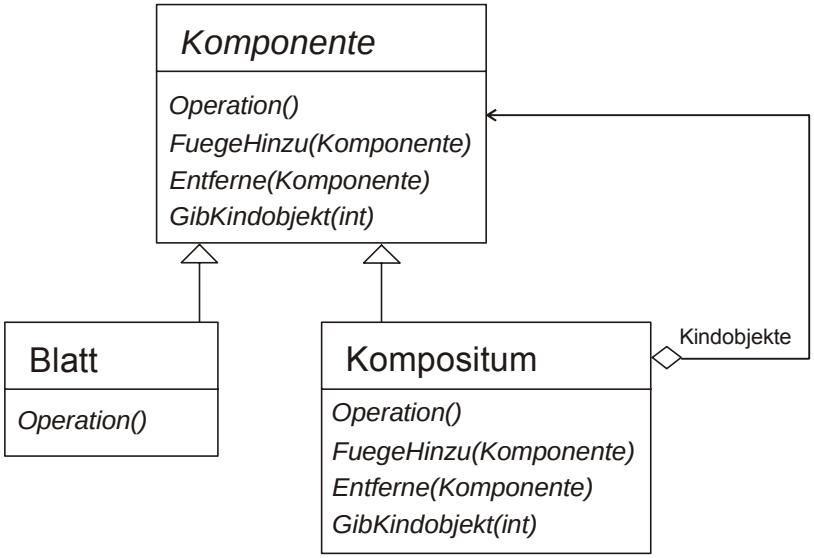

Abbildung 31 – Entwurfsmuster Kompositum [7]

Struktur der Hauptdaten

Abbildung 32 zeigt sämtliche Datenobjekte, die zur Automatisierung der parametrisierten Modellierung von Schaltungen benötigt werden.

Wie bereits erwähnt, dienen einfache Ordner (SimpleFolder) mit beliebigen Unterordnern zur Datenstrukturierung. Alle anderen interessanten Datenobjekte, wie Schaltungen und Schaltungsvorlagen, können Blätter dieses Baumes sein. Diese Datenobjekte sind ebenfalls eine Art Ordner (AbstractFolder), nur dass sie keine weiteren Strukturierungsordner beinhalten können. Die einfache Schaltung (SimpleCircuit) ist ein solcher abstrakter Ordner und kann mehrere Schaltungsdaten, zum Beispiel SPICE Dateien, Einstellungsdateien zur Speicherung von unterstützten Simulatortypen, den Namen der Top-Level-Schaltungsdatei und Ähnliches beinhalten.

Von der einfachen Schaltung sind alle weiteren speziellen Datenobjekte abgeleitet. Das ist zum Beispiel die abstrakte Vorlage (AbstractTemplate), die als Schnittstelle für die Vorlagen der Modelle (ModelTemplate) und der Testbenches (TestbenchTemplate) dient, wobei zu einem Modell mehrere Testbenches gehören können. Das Modellierungsprojekt (ModellingProject) ist der Ordner, in dem sich die Originalschaltung befindet. Zusätzlich werden noch das zugehörige parametrisierte Modell (ModelTemplateInstance) und erweiterte Testbenches (TestbenchTemplateInstance) in dem Verzeichnis gespeichert. Diese erweiterten Testbenches besitzen alle notwendigen Informationen für den Simulator und speichern die

Simulationsergebnisse. So werden alle Daten zur Simulation und Auswertung der Schaltungen in dem Projektordner gehalten.

Wie bei den Basisdatenobjekten gibt es hier einen Manager (DataManager), der Basisfunktionen zur Verwaltung der Daten bereithält.

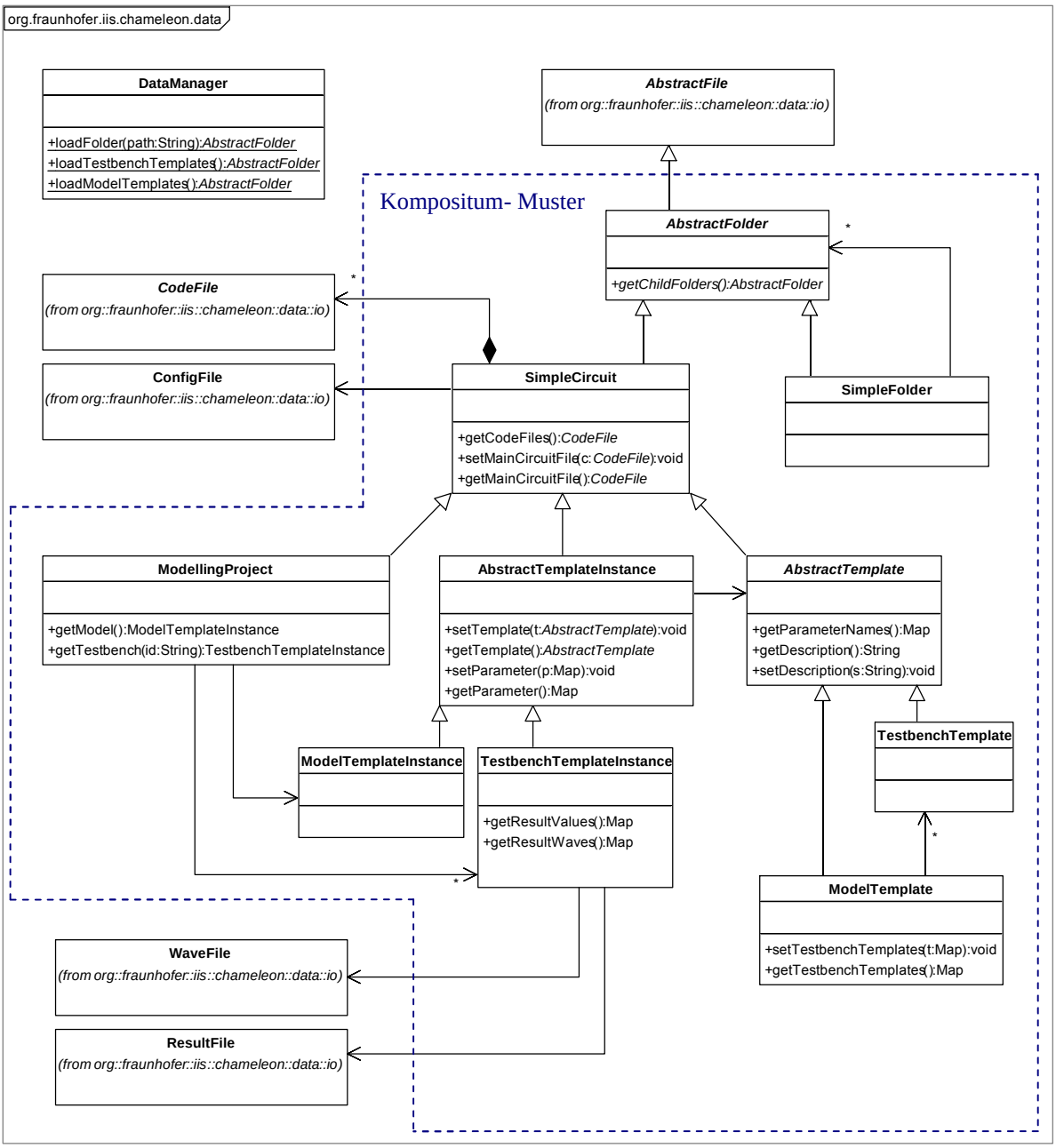

Abbildung 32 – Klassendiagramm der Hauptdaten

4.3.8 Entwurf zur Simulatoranbindung

Allgemeine Simulatoranbindung

Zur Simulatoranbindung soll wiederum das Entwurfsmuster Schablonenmethode verwendet werden, vgl. *Abbildung 29*. *Abbildung 33* zeigt das entsprechende Klassendiagramm. Dabei sollen sämtliche externen Programme, egal, ob es sich um Simulatoren oder Auswertungssoftware handelt, gleich behandelt werden. Soll ein neues Programm angebunden werden, muss von der Klasse abstrakter Ausführer (AbstractExecuter) abgeleitet und die entsprechenden Methoden implementiert werden. Die „Anwendung“ aus der Schablonenmethode befindet sich in der Klasse Executer. Sie steuert Simulation und Auswertung einer Schaltung in einer Testbench und speichert die Ergebnisse.

Zur Sicherung vor Zugriffen auf die Simulatoren wird das Singleton Muster verwendet. Dieses Entwurfsmuster sichert durch spezielle Recht-Vergabe der Methoden die Existenz von jeweils nur einer Instanz der Simulatoren. So wird erreicht, dass die externen Programme nicht beliebig ausgeführt werden können.

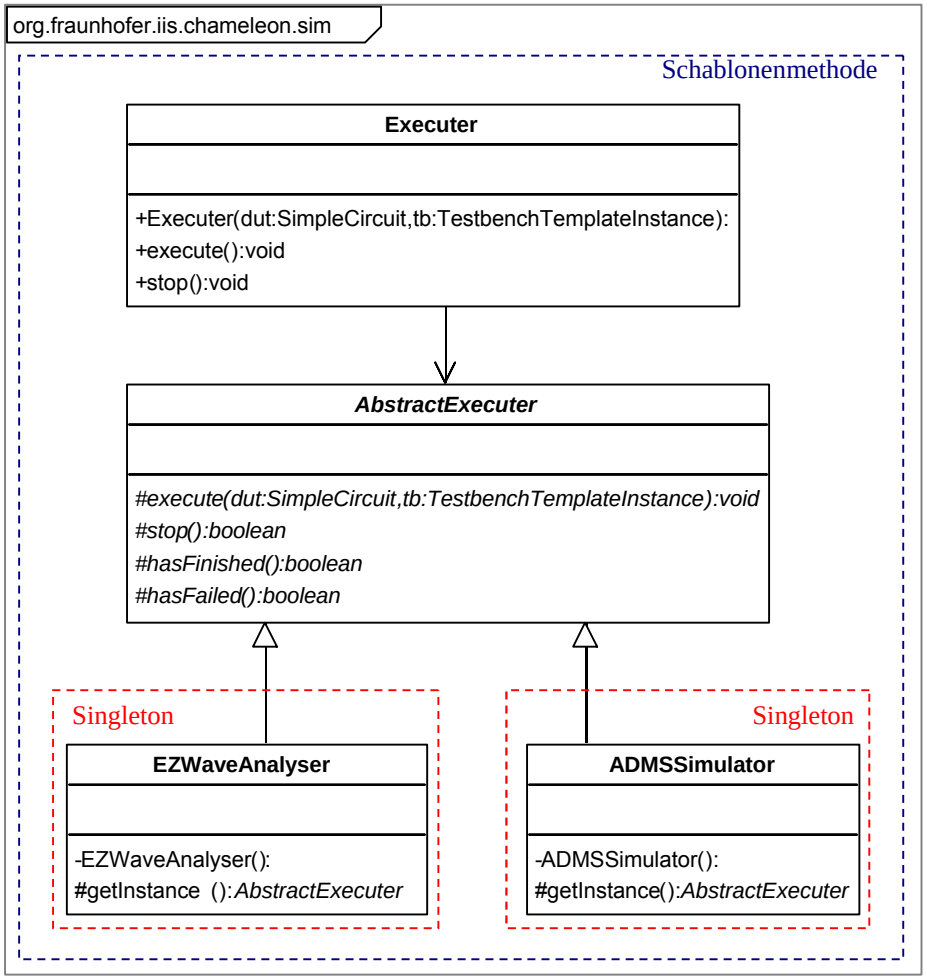

Abbildung 33 – Klassendiagramm zur Simulatoranbindung

Detaillierte Anbindung des ADVanceMS Simulators und der EZWave Auswertungssoftware

Wie unter *4.3.3 Untersuchung der Simulations- und Auswertungsvarianten* erläutert, werden die beiden Programme ADVanceMS und EZWave von MENTOR GRAPHICS in Chameleon eingebunden.

Die Anbindung des ADVanceMS Simulators gestaltet sich relativ einfach. Man kann den Simulator aus der Konsole heraus mit einem Befehl starten und ihm gleichzeitig die zu simulierende Schaltungsdatei übergeben. In JAVA kann dies durch das Starten eines Prozesses umgesetzt werden. Dabei werden Fehler- und Statusausgaben für die grafische Anzeige entsprechend umgeleitet.

Im Prinzip kann die gleiche Lösung für die Auswertungssoftware EZWave verwendet werden. Mit Angabe eines TCL-Skripts und den Ergebnisdaten des Simulators kann der TCL-Postprocessor des Programms aus der Konsole heraus gestartet werden, ohne die grafische Oberfläche von EZWave starten zu müssen. Mit Hilfe des TCL-Skripts werden die Messwerte ermittelt und diese Daten zur Weiterverarbeitung gespeichert.

Unter Punkt *4.3.3 Untersuchung der Simulations- und Auswertungsvarianten* wurde bereits die Einschränkung der *(A11) – Bedienbarkeit*, die diese Methode mit sich bringt, erläutert. EZWave soll darum genauer untersucht werden. So könnte dieser Nachteil durch eine direkte Anbindung des Programms vermieden werden.

Als erstes soll der Aufbau der besagten TCL-Skripte genauer betrachtet werden. Zur Wiederholung der Syntax soll noch einmal das Beispiel aus *4.3.3* gezeigt werden: Es wird aus der Kennlinie des imaginären Stromes über einer Spannungsquelle der Wert bei 100000 Hz extrahiert.

```
set I_100kHz evalExpression "yval(wf(\"<circuit/AC>Ii(V1)\"),100000)"
```

Zur Extraktion von Messwerten aus einer Simulationskennlinie muss das EZWave-eigene TCL-Kommando „evalExpression“ aufgerufen werden. Als Parameter muss ein String angegeben werden, der das Extraktionskommando beinhaltet. Dieses Extraktionskommando besitzt eine TCL-fremde Syntax. Es besteht wiederum aus einem Befehl und entsprechenden Parametern. Im obigen Beispiel ist das der Befehl „yval“ mit zwei Parametern. Dabei steht „yval“ für die Bestimmung des Y-Wertes aus einer Kennlinie. Der erste Parameter gibt den Namen der Kennlinie an und der zweite Parameter die X-Koordinate.

Hier könnte man eine Steigerung der *(A11) Bedienbarkeit* erreichen, indem man die gezeigten Extraktionskommandos direkt an EZWave übergibt. So kann die Verwirrung, die durch die beiden unterschiedlichen Syntaxdarstellungen entsteht, beseitigt, und die Lesbarkeit der Skripte erhöht werden.
Um Möglichkeiten in dieser Richtung genauer zu untersuchen, soll die interne Struktur von EZWave betrachtet werden. Dazu wurden mit Hilfe des Programms JAD (JAva Decompiler) die Bibliotheken des Programms zurück übersetzt und ausgewertet. Es entstanden folgende Erkenntnisse: EZWave verarbeitet die TCL-Skripte durch einen JAVA-internen TCL-Interpreter. Trifft dieser spezielle TCL-Interpreter auf das Kommando „evalExpression", wird das oben gezeigte Extraktionskommando durch einen JAVA-internen Python Interpreter (Jython) ausgewertet. Die erwähnte Funktionsbibliothek zur Verarbeitung der Kennlinien ist dabei vollständig in Python realisiert.
EZWave kombiniert damit drei verschiedene Programmiersprachen, um bei jeder Funktionalität nur die Vorteile der jeweiligen Sprache zu nutzen. Dabei dient TCL als anerkannte Skriptsprache zum Erstellen von Programmschnittstellen, zur möglichen Programmeingabe. Java ist geeignet zur Erstellung komplexer Anwendungen. Darum ist das Hauptprogramm in dieser Sprache realisiert. Doch sowohl Java als auch TCL sind für mathematische Berechnungen ungeeignet. Python hingegen hat eine unbegrenzte Genauigkeit in der Zahlenverarbeitung und bereits komplexe Zahlen als primitiven Datentyp. Es ist dadurch bestens geeignet zur Auswertung der Kennlinien.

Es stellte sich heraus, dass der JAVA-interne Pythoninterpreter von EZWave für sich allein aufgerufen werden kann. So kann man wie erhofft, die Extraktionskommandos direkt auswerten und das o. g. Beispiel verkürzt sich folgendermaßen:

```
yval(wf(\"<circuit/AC>Ii(V1)\"),100000)
```

Die Möglichkeit zur Verwendung von TCL-Befehlen geht dadurch zwar verloren, doch da stattdessen Pythonbefehle verwendet werden können, sind sämtliche unter *4.3.3* genannten Vorteile wie zum Beispiel die Erweiterbarkeit, erhalten geblieben.
Um dem Modellierungsexperten die Implementierung der Extraktionsbedingungen noch weiter zu vereinfachen, bietet es sich an, die Kennlinienbezeichnung `wf(\"<circuit/AC>Ii(V1)\")` in eine Konstante, wie zum Beispiel `Ii_V1,` zu speichern. So sieht das besagte Beispiel dann folgendermaßen aus:

```
yval(Ii_V1, 100000)
```

Die Untersuchung von EZWave zeigte noch eine zweite Möglichkeit zur Auswertung der Kennlinien. Es existieren Klassen mit so genannten „nativen“ Methoden für die umfangreiche Pythonbibliothek. Dadurch kann direkt in JAVA auf diese Python-Funktionen zugegriffen werden. Der nächste JAVA-Abschnitt zeigt wiederum das Beispiel zur Stromextraktion:

```
1)   JwdbMgr.openWdb("circuit.wdb", null, null);
2)   JwdbObject wdb = JwdbMgr.findWdbObject("<circuit/AC>Ii(V1)");
3)   JwdbPyObject wv_i = new JwdbPyObject(wdb);
4)   PyList list_i = (PyList)JwdbPyWfFunc.yval(wv_i,100000);
5)   PyFloat I_100kHz = (PyFloat)list_i_out.__getitem__(1);
```

Diese JAVA-Befehle sind aufgrund ihrer Komplexität gänzlich ungeeignet zur Verwendung für die Eingabe der Extraktionsbedingungen des Modellierungsexperten. Doch durch diese kann sofort programmintern, ohne den Umweg über externe Dateien gehen zu müssen, auf die Messergebnisse zugegriffen werden. Dadurch können zum Beispiel die Kennlinien recht einfach graphisch dargestellt werden.

Fazit: Zur Eingabe der Extraktionsbedingungen durch den Modellierungsexperten sollen die oben gezeigten kurzen Extraktionskommandos verwendet werden, die durch den Pythoninterpreter von EZWave ausgewertet werden. Wenn programmintern auf die Simulationsergebnisse zugegriffen werden soll, kommen die nativen Methoden der EZWave JAVA-Bibliothek zum Einsatz.

4.4. Implementierung

4.4.1 Einleitung

Das Werkzeug Chameleon wurde nun nach der im Entwurf entwickelten Struktur implementiert. Es entstand eine Anwendung, die aus 74 Klassen, 5 Entwurfsmustern und ca. 8.000 Codezeilen besteht. Dieses Kapitel zeigt Bilder des Werkzeuges und stellt deren Bedienung vor.

4.4.2 Programmbedienung aus Sicht des Schaltungsentwicklers

Der Schaltungsentwickler startet das Programm und beginnt damit, ein neues Projekt zu erstellen. Dazu öffnet er eine Schaltung, die er modellieren möchte. Die Schaltung ist in einer bekannten Schaltungsbeschreibungssprache gegeben. Der Speicherort dieser Schaltung wird gleichzeitig der Projektordner. Der Schaltungsentwickler muss als nächstes ein passendes Modell auswählen. Dann gelangt er zum Übersichtsfenster der Schaltungsmodellierung. *Abbildung 34* zeigt das Fenster zur Modellierung einer Operationsverstärkerschaltung.

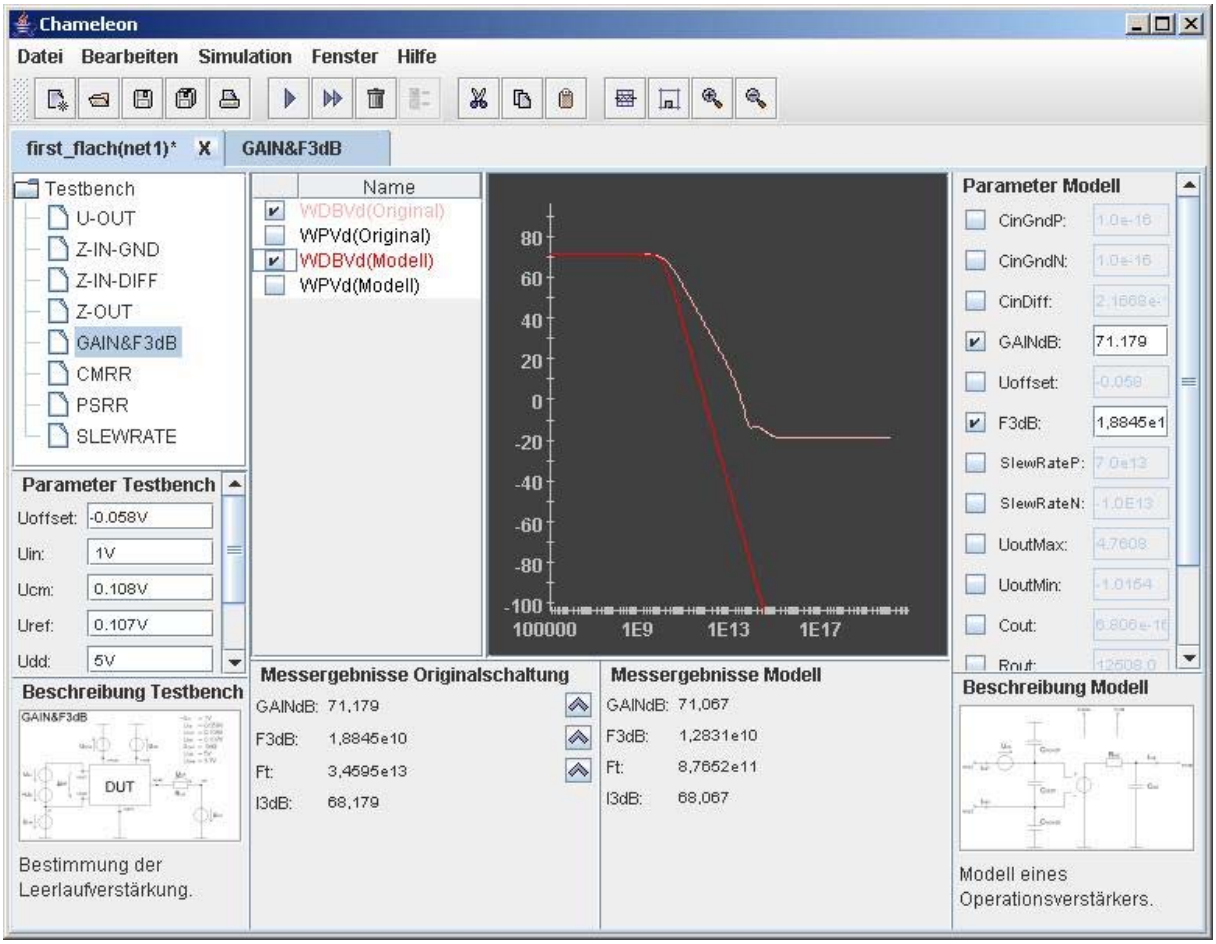

Abbildung 34 – Bildschirmfenster Schaltungsmodellierung

Links oben befindet sich eine Auswahl an Testbenches zur Charakterisierung der Schaltung. Es kann zum Beispiel zwischen der Bestimmung der Ausgangsimpedanz, der Eingangsimpedanz, der Verstärkung etc. ausgewählt werden. Darunter befinden sich die Betriebsparameter, wie zum Beispiel Versorgungsspannung, und die Extraktionsparameter, wie zum Beispiel Messfrequenz der ausgewählten Testbench. Die Standardwerte dieser Parameter können angepasst werden. Unterhalb dieser Parameter befindet sich ein Bild mit einer kurzen Erläuterung der Testbench und Parameterwerte. Mit Mausklick auf das Bild kann es vergrößert werden. Entsprechend gibt es rechts unten ein Bild mit einer Beschreibung des Modells.

Nach einer Simulation können, wie in der Mitte der Abbildung dargestellt, die Kennlinien eingeblendet werden. Damit können Unterschiede der Simulationsergebnisse von Originalschaltung und Modell verglichen werden. Unterhalb der Kennlinien sind die Extraktionsergebnisse der Kenngrößen von Originalschaltung und Modell angegeben. Neben den Kennwerten der Originalschaltung, die mögliche Parameterwerte des Modells zeigen,

befinden sich Schaltknöpfe, mit denen die Werte links oben in die Liste der Modellparameter übernommen werden können.

Oben befinden sich das Menü und die Werkzeugleiste. Dort sind alle möglichen Funktionen, wie zum Beispiel Speichern, Öffnen, Simulieren etc., erreichbar. Nach Starten der Simulationsfunktion wird ein Dialog geöffnet. *Abbildung 35* zeigt den Dialog zur Simulation der Originalschaltung.

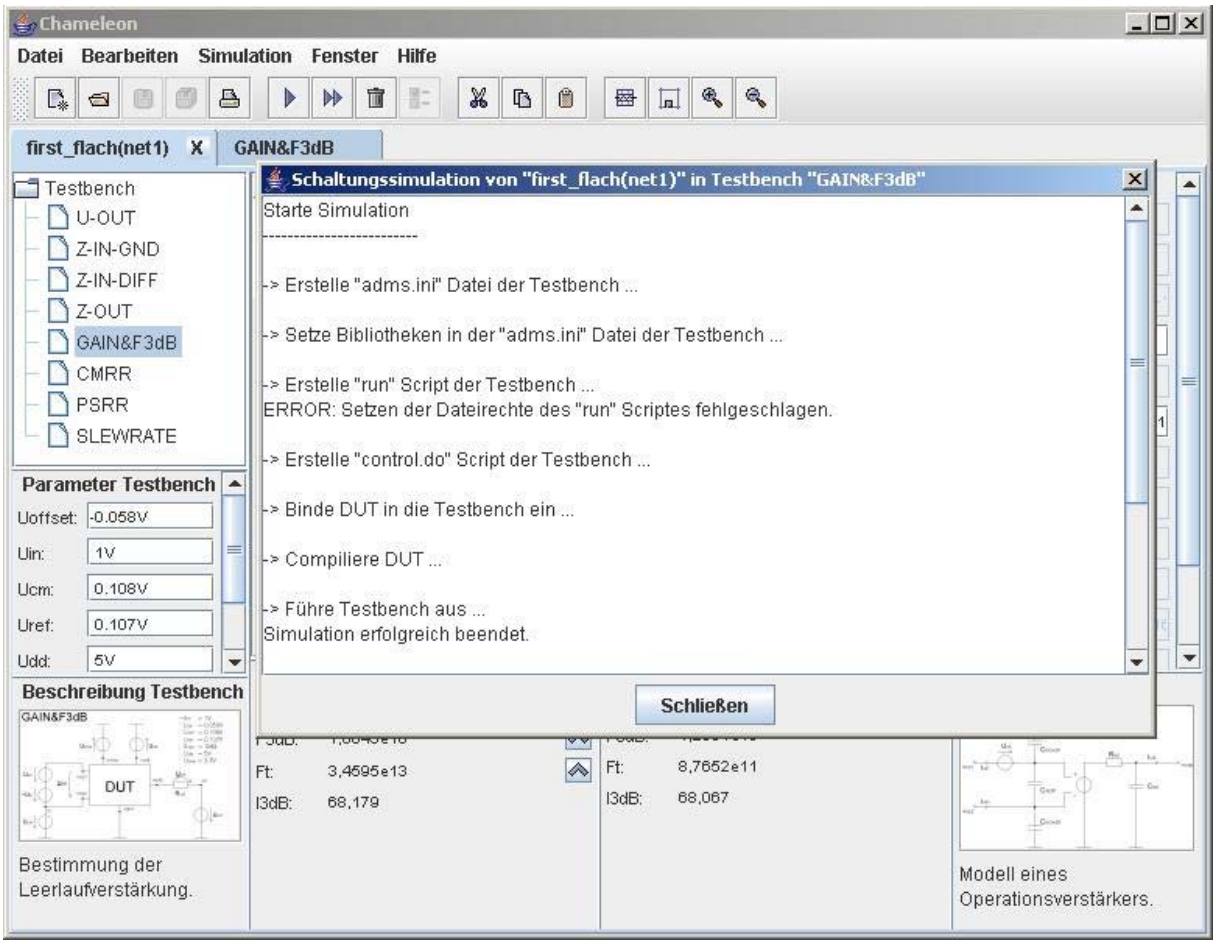

Abbildung 35 – Bildschirmfenster Simulation

Neben Statusmeldungen werden auch Fehlermeldungen wie zum Beispiel Syntaxfehler in den Schaltungen oder Extraktionsgleichungen ausgegeben.

Der Schaltungsentwickler findet das fertige Modell mit den ermittelten Parametern in seinem Projektordner.

4.4.3 Programmbedienung aus Sicht des Modellierungsexperten

Zum Erstellen eines neuen Modells einer Schaltungsklasse muss der Modellierungsexperte als erstes entsprechende Testbenches anlegen. Dazu öffnet er nach Starten des Programms die Testbenchverwaltung, vgl. *Abbildung 36*.

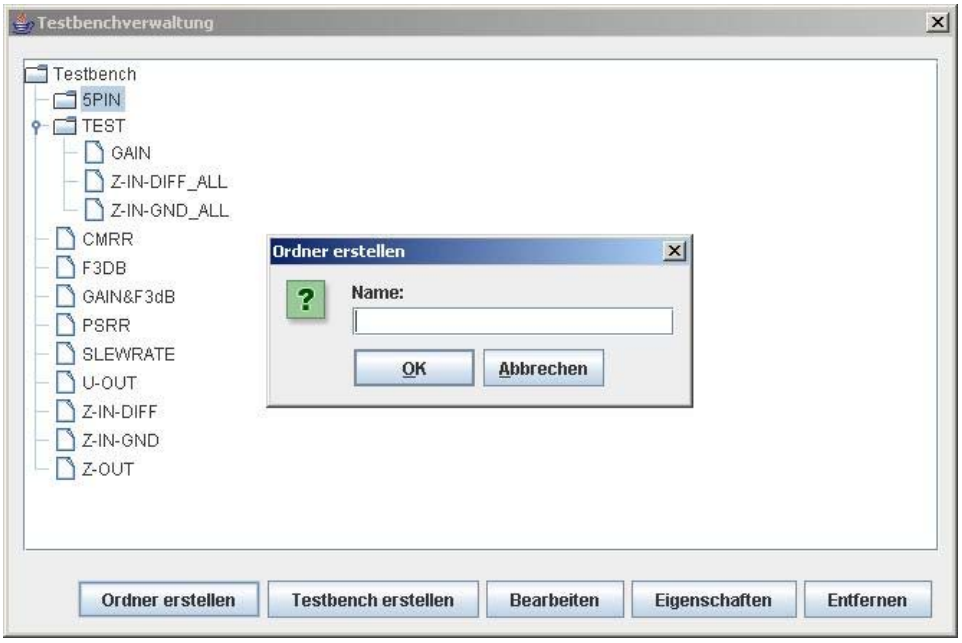

Abbildung 36 – Bildschirmfenster Testbenchverwaltung

Er kann durch Erstellung beliebiger Ordner und Unterordner die Verwaltungsstruktur der Messschaltungen anpassen. Beim Mausklick auf den Knopf „Testbench erstellen“ öffnet sich ein neuer Dialog, siehe *Abbildung 37*, in dem der Benutzer aufgefordert wird, die Eigenschaften wie Name, Beschreibung und Bild der neuen Testbench einzugeben.

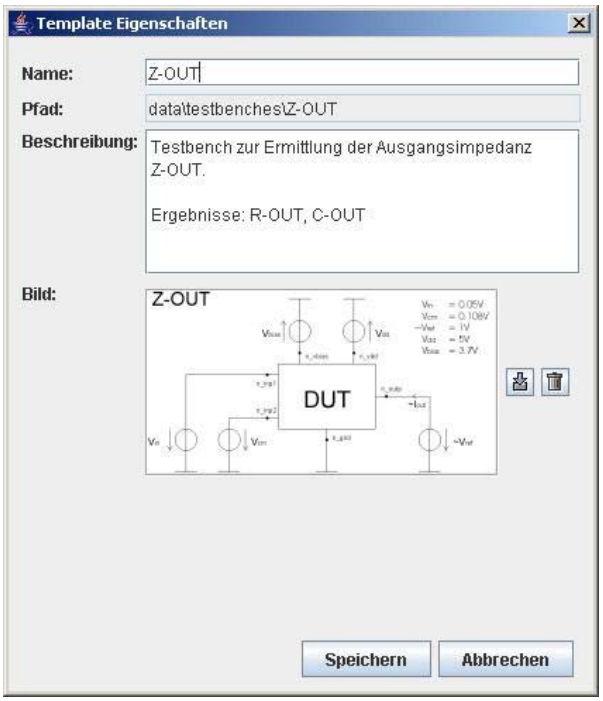

Abbildung 37 – Bildschirmfenster Eigenschaften einer Testbench

Nach dem Speichern dieser Daten landet der Modellierungsexperte im Bearbeitungsfenster. *Abbildung 38* zeigt das Fenster zum Bearbeiten der Messschaltungs- und der Extraktionsdatei.

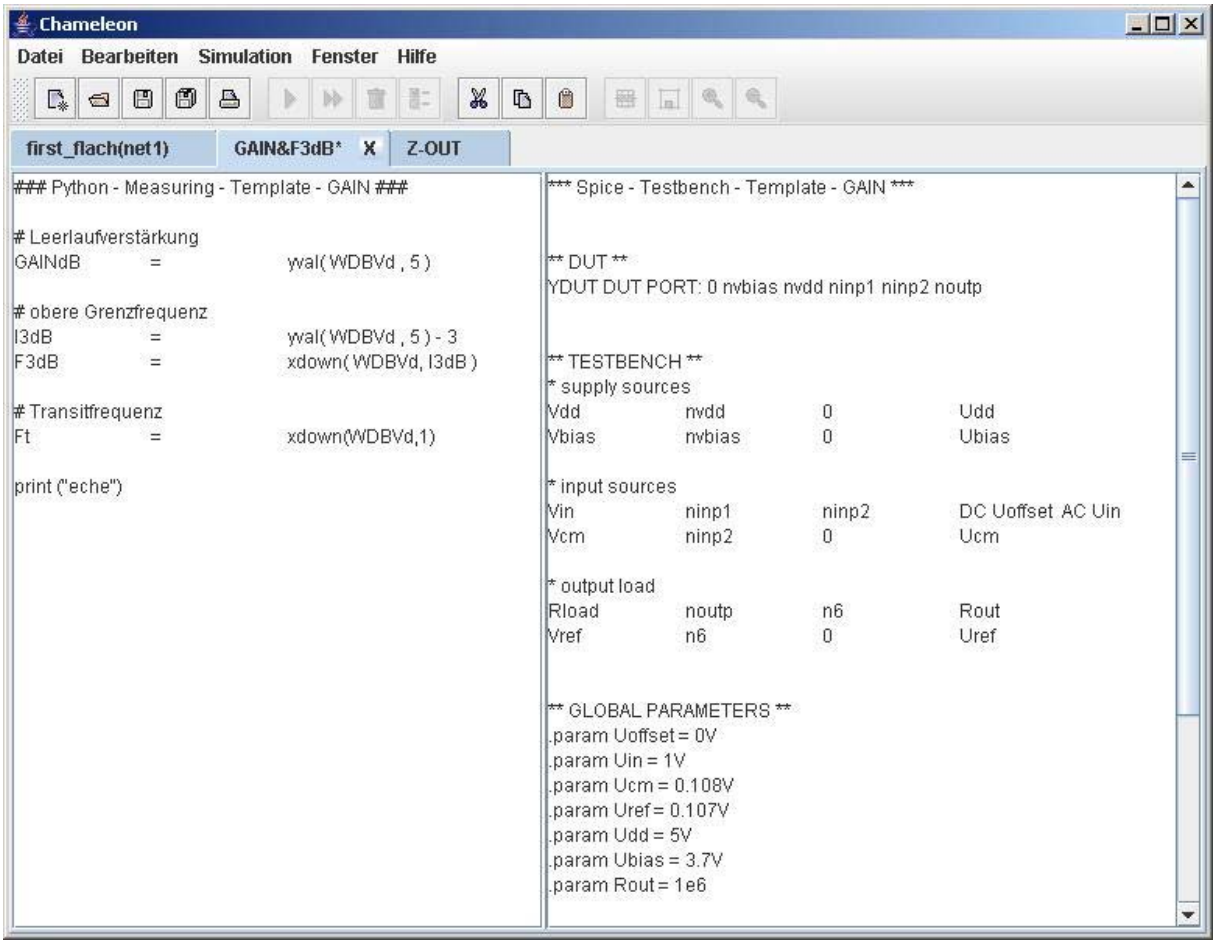

Abbildung 38 – Bildschirmfenster Messschaltung bearbeiten

In der Abbildung befindet sich rechts die Beschreibung der Messschaltung in SPICE und links die Extraktionsgleichungen in PYTHON. Nach dem Erstellen der Testbenches legt der Modellierungsexperte ein neues Modell an. Dazu öffnet er die Modellverwaltung, ähnlich zu *Abbildung 36* der Testbenchverwaltung. Hier kann er das neue Modell erstellen und wie bei der Testbench Name, Beschreibung und Bild vergeben. Im nächsten Dialog wählt er die zum Modell gehörigen Testbenches aus. Dann öffnet sich wieder ein Bearbeitungsfenster, ähnlich dem Fenster zum Bearbeiten einer Testbench, in dem der Modellierungsexperte das Modell in einer Hardwarebeschreibungssprache erstellt.

4.4.4 Beispiel einer Testbench

Zum genaueren Verständnis über die Eingabe einer Messschaltung und deren Extraktionsbedingungen sollen als Beispiel die Bestimmung der Leerlaufverstärkung und der Frequenzgang der Verstärkung eines Operationsverstärkers aus Kapitel *3.2.6* und *3.2.7* dem Programm hinzugefügt werden. *Abbildung 39* zeigt die Beschreibung des Messschaltungsaufbaus aus *Abbildung 14* in der Hardwarebeschreibungssprache SPICE.

```
1)   *** Spice-Messschaltungs-Template – Leerlaufverstärkung ***
2)
3)   ** Einbinden der Schaltung, die getestet werden soll **
4)   YDUT DUT PORT: 0 nvbias nvdd ninp1 ninp2 noutp
5)
6)   ** SCHALTUNG **
7)   * Versorgungsspannungen
8)   Vdd    nvdd        0           Udd
9)   Vbias  nvbias      0           Ubias
10)
11)  * Eingangsspannungen
12)  Vin    ninp1       ninp2       DC Uoffset  AC Uin
13)  Vcm    ninp2       0           Ucm
14)
15)  * Ausgangsgrößen
16)  Rload  noutp       n6          Rload
17)  Vref   n6          0           Uref
18)
19)  ** GLOBALE PARAMETER MIT DEFAULT-WERTEN**
20)  .param Uoffset     = 0V
21)  .param Uin         = 1V
22)  .param Ucm         = 0.108V
23)  .param Uref        = 0.107V
24)  .param Udd         = 5V
25)  .param Ubias       = 3.7V
26)  .param Rload       = 1e12
27)
28)  ** ANALYSE **
29)  .dc
30)  .ac dec 20 1 1e20
31)
32)  ** AUSGABE **
33)  .defwave Vd=V(noutp,n6)/V(ninp1,ninp2)
34)  .probe ac WDB(Vd)
35)
36)  .end
```

Abbildung 39 – Messschaltung Frequenzgang der Leerlaufverstärkung in SPICE

In Zeile 4 wird der DUT – das so genannte „Device under Test“, also die zu testende Schaltung, initialisiert. Es werden die Knotennamen angegeben, an denen die Schaltung angeschlossen ist. Ob es sich bei dem DUT um die Originalschaltung oder das Modell handelt, wird vom Programm automatisch je nach Bedarf ergänzt.

Die Zeilen 6 bis 17 beschreiben den Schaltungsaufbau der Messschaltung, wobei die Werte der Bauteile mittels Parameter angegeben werden. Die Zeilen 19-26 definieren diese Parameter und setzten Standardwerte. So definierte Parameter werden automatisch durch das Programm erkannt und können mit Hilfe der grafischen Oberfläche einfach durch den Schaltungsentwickler angepasst werden. Zeile 29 und 30 bestimmen die Analyseart der Schaltung, in diesem Fall eine Frequenzanalyse. In Zeile 34 wird angegeben, welche Kennlinien aufgezeichnet werden sollen. Dafür wird zuvor in Zeile 33 die

Verstärkungskennlinie `Vd` als Quotient der Ausgangsspannung über `Rload` zur Differenzeingangsspannung definiert.

Abbildung 40 zeigt die Messdatei, die die Extraktionsgleichungen der Testbench enthält. Sie ist in der durch das Programm EZWave erweiterten PYTHON-Sprache beschrieben. In Zeile 4 wird die Leerlaufverstärkung ermittelt. Dabei wird mittels `WDBVd` auf die aufgenommene Kennlinie `Vd` zugegriffen. Bis auf die Entfernung der Klammern entspricht diese Bezeichnung der der „.probe“ Anweisung aus *Abbildung 39* Zeile 34.

```
1)   ### Extraktions-Template - Leerlaufverstärkung ###
2)
3)   # Bestimmung der Leerlaufverstärkung
4)   GAINdB =     yval( WDBVd , 10 )
5)
6)   # Bestimmung der oberen Grenzfrequenz
7)   I3dB   =     yval( WDBVd , 10 ) - 3
8)   F3dB   =     xdown( WDBVd, I3dB )
9)
10)  # Bestimmung der Transitfrequenz
11)  Ft     =     xdown(WDBVd,1)
```

Abbildung 40 – Extraktionsvorschriften Frequenzgang der Leerlaufverstärkung in Python

4.5. Test und Wartung

4.5.1 Einleitung

In diesem Kapitel werden Testmethoden vorgestellt und eine Akzeptanzanalyse durchgeführt. Tests dienen zur Validierung des Programms. Eine vollständige Verifikation ist auf Grund ihrer Komplexität nicht durchführbar, darum werden Tests durchgeführt, die die Hauptfunktionalität des Programms überprüfen sollen.

Die Akzeptanzanalyse beschreibt, inwiefern die gemäß Kapitel *4.2.6* ermittelte Anforderungsspezifikation realisiert wurde.

Aus den Tests und der Akzeptanzanalyse ergaben sich neue Ideen und Vorschläge zur Programmverbesserung und -erweiterung. Diese werden im Unterkapitel *4.5.4* vorgestellt.

4.5.2 Tests

Die in *4.3.7* vorgestellte programminterne Datenstruktur wurde mit so genannten Regressionstests validiert. Bei der Regressionstestmethode werden Testfälle für die Software spezifiziert. Diese Testfälle sind mit einem Soll-Ergebnis versehen, welches mit dem Ist-Ergebnis des Testfalls verglichen wird. Durch Abweichung von Soll- und Ist-Ergebnissen können Fehler gefunden werden.

Dazu wurde das Test-Framework JUnit verwendet. Es wurden Testfälle für das Schreiben, Lesen, Löschen etc. der Datenobjekte angelegt. Weiterhin wurden abzutestende Bedingungen festgelegt. Folgender Vorgang ist zum Beispiel ein Testfall mit entsprechender Testbedingung: Eine Datei wird gelöscht. Dann wird überprüft ob die Datei existiert. Wenn die Existenzprüfung der Datei ein positives Ergebnis zeigt, schlägt der Testfall fehl und eine Fehlermeldung wird ausgegeben.

Nach der Implementierung der Datenobjekte wurden die Testfälle ausgeführt und die Objekte solange korrigiert, bis die Testfälle ohne Fehler ausgeführt werden konnten.
Die Programmoberfläche wurde durch Anwendung getestet. Dazu wurden sämtliche in Kapitel *3* vorgestellten Charakterisierungen sowie das Modell des Operationsverstärkers in das Programm eingearbeitet und per Simulation getestet.

4.5.3 Akzeptanzanalyse

In einer Akzeptanzanalyse wird überprüft, inwiefern die in der Anforderungsspezifikation, vgl. Kapitel *4.2.6*, definierten Anforderungen eingehalten wurden.
In dem Werkzeug Chameleon ist folgender Funktionsumfang realisiert:

- Schaltungscharakterisierung und graphische Darstellung der Ergebnisse ist möglich
- Verhaltensmodell, Messschaltungen und Extraktionsbedingungen des Operationsverstärkers sind vorhanden
- Verhaltensmodelle, Messschaltungen und Extraktionsbedingungen sind beliebig erweiterbar
- weitere Simulatoren, Auswertungsmöglichkeiten, Skript-, Hardwarebeschreibungssprachen und Modellierungsmethoden können ergänzt werden
- das Werkzeug ist plattformunabhängig, nur die verwendeten Simulatoren grenzen die Portierbarkeit ein
- die grafische Oberfläche ist einstellbar und auch an unterschiedliche Fenstersysteme anpassbar

Damit sind die gestellten funktionalen und die nichtfunktionalen Anforderungen gut erfüllt.

4.5.4 Ausblick in der Programmwartung

Dieses Kapitel zeigt Möglichkeiten zur Erweiterung des Werkzeuges Chameleon im Sinne der Programmwartung. Folgende Aufzählung zeigt die wichtigsten Punkte der Programmwartung:

- Pin-Kompatibilität
- direkte Modellgenerierung
- andere Modellierungsmethoden, speziell Tabellenmodellierung
- automatisierte Modellierung mehrerer Schaltungen
- Bibliothekserweiterung

Die Pin-Kompatibilität von zum Beispiel Modell und Originalschaltung oder von DUT und Messschaltung sollte überprüfbar und einstellbar sein. So könnten zum Beispiel bei der Auswahl eines Modells zur Originalschaltung nur Modelle mit gleicher Anzahl an Pins vorgeschlagen werden. Zusätzlich wäre eine Eingabemaske zur Zuordnung der Pins von Vorteil, um Simulationsfehlern vorzubeugen. Dasselbe gilt bei der Zuordnung von Testbenches zu Modellen.
Mit direkter Modellgenierung ist gemeint, dass die Verhaltensmodelle nicht nur parametrisiert, sondern auch generiert werden können. Um irrelevante Modellparameter zu entfernen, werden in Chameleon Parameterwerte verwendet, bei denen der Effekt dieser Parameter vernachlässigbar ist. So wird beispielsweise der Ausgangswiderstandswert sehr groß gewählt, um seinen Einfluss zu vernachlässigen. Nachteil dabei ist, dass die Gleichung, die den Widerstand beschreibt, nicht wirklich entfernt wird und weiterhin in die Simulationszeit des Modells eingeht. Besser wäre es, wenn Modellteile, zu denen kein Parameterwert gesetzt ist, bei der direkten Modellgenerierung vollständig aus dem Modell entfernt würden. Durch direkte Modellgenerierung könnten auch unterschiedliche Modellvarianten beliebig zusammengesetzt und ein individuelles Modell erzeugt werden.
Die Erzeugung von Tabellenmodellen, vgl. Kapitel *2.2.5*, wäre einfach zu realisieren, da bereits komplette Kennlinien im Werkzeug verwaltet werden.
Eine weitere Aufgabenstellung wäre die Realisierung der automatisierten Modellierung mehrerer Schaltungen. In einer entsprechenden Eingabemaske sollen Einstellungen wie Modell, verwendete Parameter, Reihenfolge der Charakterisierung etc. festgelegt werden können. Nach der Modellierung sollte eine zusammenfassende Ergebnispräsentation Abweichungen, Genauigkeit und Geschwindigkeit der Modelle zeigen.
Schließlich könnte die Modell- und Testbenchbibliothek auf weitere Schaltungsklassen wie zum Beispiel Oszillatoren erweitert werden.

5. Zusammenfassung

Beim Entwurf von Mixed-Signal-Schaltungen nehmen die analogen Komponenten von der Chipfläche her einen geringen Anteil, vom Entwurfsaufwand her aber einen wesentlich größeren Anteil als die digitalen Komponenten ein. Ziel der Verbesserung und Entwicklung neuer Entwurfsverfahren sind automatisierte Methoden zur Schaltungsauswahl, Schaltungsdimensionierung und Schaltungsverifikation. Im Rahmen der vorliegenden Arbeit wurden Modellierungsmethoden zur Unterstützung der Schaltungsverifikation untersucht und prototypisch implementiert.

Die Hauptaufgabe umfasste die Entwicklung eines Werkzeuges zur Unterstützung der Schaltungscharakterisierung. Aus den Ergebnissen der Schaltungscharakterisierung können vorgefertigte Modelle parametrisiert und per Simulation getestet werden. Der Operationsverstärker wurde anhand typischer Eigenschaften charakterisiert und modelliert. Aufbau der Messschaltungen und die Extraktionsgleichungen wurden in das Werkzeug eingepflegt. Weitere Schaltungsklassen können einfach dem Werkzeug hinzugefügt werden.

Das entwickelte Werkzeug stellt zwar keine Konkurrenz zu vergleichbarer kommerzieller Software dar, ist jedoch durch eine sehr gute Erweiterbarkeit, verbunden mit umfangreichen Möglichkeiten zur Schaltungscharakterisierung, gekennzeichnet. Weiterhin ist der Anwender an keine Hersteller und deren Werkzeuge gebunden, sondern kann beliebige eigene Vorstellungen mit dem Werkzeug umsetzen. Das Werkzeug Chameleon ist demzufolge sehr gut für Forschungs- und Testzwecke sowie zur Entwicklung von Modellierungstechniken geeignet.

Eine Funktionserweiterung des Werkzeuges Chameleon stellt durch die Verwendung moderner Softwaretechniken, wie beispielsweise Entwurfsmuster, kein Problem dar.
Perspektivische Aufgabengebiete liegen in der Weiterentwicklung des Werkzeuges, wie zum Beispiel in der Anbindung weiterer Modellierungsmethoden.

6. Literaturverzeichnis

[1] Gielen, G. G. E.; Rutenbar, R. A.:
„Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits”
Proc. IEEE, Vol. 88 Nr. 12, Dez. 2000, S. 1825-1852

[2] Gielen, G.G.E.:
„CAD tools for embedded analogue circuits in mixed signal integrated systems on chip”
IEE Proc.-Comput. Digit. Tech., Vol. 152, Nr. 3, Mai 2005, S. 317-332

[3] Kundert,K.:
„A Formal Top-Down Design Process for Mixed-Signal Circuits“
Analog Circuit Design, Kluwer Academic Publishers, www.designers-guide.com,
Nov. 2000

[4] Duyan, H.; Hahnloser G.; Traeger; D. H.:
„PSpice – Eine Einführung“
Teubner Studienskripte, 2. Aufl. Stuttgart, 1992, ISBN 3-519-10143-2

[5] Seifert, M.:
„Analoge Schaltungen”
Verlag Technik, 3. Aufl. Berlin, 1989, ISBN 3-341-00740-7

[6] Österreich, B.:
„Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language“
Oldenburg Verlag, 4. Aufl. München Wien, 1998, ISBN 3-486-24787-5

[7] Gamma, E.; Helm, R.; Johnson, J.; Vlissides, J.:
„Entwurfsmuster“
Addison Wesley Verlag, 2004, ISBN 3-8273-2199-9

[8] Welche, B.:
„Praktisches Programmieren in TCL und TK“
Prentice Hall Verlag, 1999, ISBN 3-8272-9529-7

7. Symbolverzeichnis

7.1. Einleitung

Dieses Kapitel gibt eine Zusammenfassung aller verwendeten Formel- und Funktionssymbole. Bei den Formelzeichen wird zwischen Kenngrößen, Betriebs- und Extraktionsparametern und sonstigen Formelzeichen unterschieden. Kenngrößen sind Modellparameter des Verhaltensmodells des Operationsverstärkers. Betriebsparameter sind die Betriebsgrößen zur Arbeitspunkteinstellung der Originalschaltung. Extraktionsparameter sind Parameter der Extraktionsgleichungen. Betriebs- und Extraktionsparameter sind durch den Anwender einzustellen. Sonstige Formelzeichen sind alle verwendeten Symbole, die zur Erläuterung und Herleitung der Messschaltungen und Extraktionsgleichungen dienen.

7.2. Formelzeichen

7.2.1 Kenngrößen

C_{inDiff}	Differenz-Eingangskapazität in F
C_{inGndN}	Gleichtakt-Eingangskapazität am negativen Eingang des OPV in F
C_{inGndP}	Gleichtakt-Eingangskapazität am positiven Eingang des OPV in F
CMRR	Gleichtaktunterdrückungsverhältnis in dB
C_{out}	Ausgangskapazität des OPV in F
f_o	obere Grenzfrequenz des OPV in Hz
PSRR	Betriebsspannungsunterdrückungsverhältnis in dB
R_{out}	Ausgangswiderstand in Ω
S_{RN}	Slew Rate bei negativer Sprungerregung des OPV in $\frac{V}{s}$
S_{RP}	Slew Rate bei positiver Sprungerregung des OPV in $\frac{V}{s}$
U_{os}	Offsetspannung in V
U_{outMin}	Untere Grenze des Aussteuerbereiches in V
U_{outMax}	Obere Grenze des Aussteuerbereiches in V
V_{d0}	Leerlaufverstärkung in dB

7.2.2 Betriebs- und Extraktionsparameter

f_m	Messfrequenz in Hz
U_{bias}	Biasspannung in V
U_{dd}	Versorgungsspannung in V
U_{cm}	Common-Mode-Spannung in V
U_{in}	Eingangsspannung in V
U_{ref}	Referenzspannung in V
R_{load}	Lastwiderstand am Ausgang in Ω
rate	Faktor zur Einstellung der Genauigkeit der Ausgangsspannungsgrenzen

7.2.3 Sonstige Formelzeichen

f	Frequenz in Hz
f_t	Transitfrequenz in Hz
I_{inN}	Eingangsstrom am negativen Eingangs des OPV in A
I_{inP}	Eingangsstrom am positiven Eingangs des OPV in A
U_{diff}	Eingangsdifferenzspannung in V
U_{out}	Ausgangsspannung in V
R_{inDiff}	Differenzeingangswiderstand des OPV in Ω
R_{inGndN}	Gleichtaktwiderstand am negativen Eingang des OPV in Ω
R_{inGndP}	Gleichtaktwiderstand am positiven Eingang des OPV in Ω
R_{out}	Ausgangswiderstand des OPV in Ω
V	Verstärkung in dB
V_d	Differenzverstärkung in dB
V_{gl}	Gleichtaktverstärkung in dB
V_{psr}	Betriebsspannungsverstärkung in dB
Y_{in}	Leitwert am Eingang des OPV in S
Y_{inN}	Gleichtakt-Leitwert am negativen Eingang des OPV in S
Y_{inP}	Gleichtakt-Leitwert am positiven Eingang des OPV in S
Y_{inDiff}	Differenzleitwert am Eingang des OPV in S

Y_{inGnd} Gleichtaktleitwert am Eingang des OPV in S

Y_{out} Leitwert am Ausgang des OPV in S

7.3. Funktionsbezeichnungen

$\underline{x}$	x ist komplexe Zahl
$dB\{\|x\|\} = 20 \cdot \log(\|x\|)$	Berechnung des dB-Wertes einer Zahl
$Re\{\underline{z}\} = Re\{x + j \cdot y\} = x$	Bestimmung des Realwertes einer komplexen Zahl
$Im\{\underline{z}\} = Im\{x + j \cdot y\} = y$	Bestimmung des Imaginärwertes einer komplexen Zahl
$\max\{y(x)\}$	Bestimmung des Maximums aus der Kennlinie y(x)

8. Bild- und Tabellenverzeichnis

8.1. Abbildungsverzeichnis

8.2. Tabellenverzeichnis