High-Level-Entwurf von Mikrosystemen

von der Fakultät Elektrotechnik und Informationstechnik

der Technischen Universität Chemnitz

genehmigte Dissertation

zur Erlangung des akademischen Grades

Doktor Ingenieur (Dr.-Ing.)

vorgelegt

von Dipl.-Ing. Erik Markert

geboren am 03. März 1979 in Greiz

Gutachter:

Prof. Dr.-Ing. Ulrich Heinkel

Prof. Dr.-Ing. habil. Jan Mehner

Dr.-Ing. Tom Barthel

eingereicht am 25. September 2009

Tag der Verleihung: 16. Februar 2010

Chemnitz, den 21. Februar 2010

Bibliographische Angaben

Titel: High-Level-Entwurf von Mikrosystemen

Erik Markert – 2009 – 156 Seiten

40 Abbildungen, 19 Tabellen, 152 Literaturquellen

Technische Universität Chemnitz, Fakultät für Elektrotechnik und Informationstechnik, Dissertationsschrift

Kurzreferat

Die Dissertationsschrift stellt eine Toolkette zum abstrakten Entwurf von Mikrosystemen vor. Mikrosysteme können aus Elementen verschiedener physikalischer Domänen bestehen und zusätzlich digitale Hardware sowie Software enthalten. Die Erfassung und Formalisierung dieser heterogenen Systeme stellt den ersten Schritt im Entwurfsprozess dar, die damit verbundene neue Methodik des Designs von Mikrosystemen bildet den Kern der vorliegenden Arbeit.

Zur Erfassung der analogen Spezifikationsteile enthält die Arbeit die Schilderung und Implementierung neuer Datenstrukturen, die ausgehend von einer ausführlichen Anforderungsanalyse geschaffen wurden. Das abstrakte Systemverhalten wird mit Hilfe hybrider Automaten modelliert, die sowohl mit speziellen hybriden Werkzeugen als auch mit SystemC-AMS simulierbar sind. Darüber hinaus beschäftigt sich die Arbeit mit der Erfassung von Signalverläufen und Schaltplaninformationen. Die formalisierten Anforderungen ermöglichen erste Prüfungen der Spezifikation auf Konsistenz.

Zur Unterstützung niedriger Abstraktionsebenen wie der Differentialgleichungsebene steht ein Wandler von SystemC-AMS nach VHDL-AMS bereit. In die Systembeschreibung mit SystemC-AMS ist die Definition und Verknüpfung von Kostenparametern integrierbar. Das daraus entstehende globale Gütemaß hilft dem Entwerferteam, die optimale Systemrealisierung zu finden.

Schlagwörter

Spezifikationserfassung, Entwurf heterogener Systeme, Mikrosysteme, hybride Automaten, SystemC-AMS, VHDL-AMS, Kostenparameter

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADeVA	Advanced Design and Verification of ASICs
ADU	Analog-Digital-Umsetzer
AMS	Analog and Mixed Signal
DTT	Data Transformation Table
EBNF	Extended Backus-Naur Form
FEM	Finite Elemente Methode
FFT	Fast Fourier Transformation
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GUI	Graphical User Interface
INS	Inertialnavigationssystem
IEEE	Institute of Electrical and Electronics Engineers
LRM	Langauge Reference Manual
MEMS	Mikro-Elektro-Mechanische Systeme
MTT	Mode Transition Table
PWL	Piecewise Linear (stückweise linear)
PWM	Pulsweitenmodulation
SAL	Symbolic Analysis Laboratory
SVE	System Verification Environment
TLM	Transaction Level Modeling
UBAS	Universelles Bewegungsanalysesystem
VHDL	Very high speed integrated circuit Hardware Description Language
XML	eXtensible Markup Language

Dank

Die vorliegende Arbeit entstand im Rahmen meiner Tätigkeit an der Professur Schaltkreis- und Systementwurf der TU Chemnitz. Ich möchte diese Stelle nutzen, um mich bei den Helfern herzlich zu bedanken, ohne deren Unterstützung die Anfertigung der Arbeit in dieser Form nicht möglich gewesen wäre.

Einen wesentlichen Anteil an dieser Arbeit hat mein Betreuer Prof. Ulrich Heinkel, der mir in fachlichen und organisatorischen Fragen zur Seite stand und mir die Möglichkeit der Forschung im Gebiet des Entwurfs von Mikrosystemen gab. Darüber hinaus bedanke ich mich bei Dr. Tom Barthel für die fruchtbaren fachlichen Diskussionen und die Übernahme des zweiten Gutachtens sowie bei Prof. Mehner für die Übernahme des dritten Gutachtens.

Weiterhin danke ich Prof. Dietmar Müller, der mir den Berufseinstieg an der TU Chemnitz ermöglichte, und Prof. Göran Herrmann, der mit seinem profunden Wissen auf den Gebieten der Elektrotechnik und der Typographie eine sehr große Hilfe war.

Die praktische Umsetzung der Ideen dieser Arbeit sowie die Evolution der Gedankengänge wären ohne ständigen Wissensaustausch und Diskussionen mit den Kollegen der Professur Schaltkreis- und Systementwurf und der Professur Mikrosystem- und Gerätetechnik nicht möglich gewesen. Daher gilt mein Dank den Kollegen der beiden Professuren, wobei ich zwei Gruppen besonders herausgreifen möchte: Erstens das Team zur Entwicklung des Werkzeugs „SpecScribe“ mit Uwe Proß, Jan Langer, Andreas Richter, Chris Drechsler, Claudia Tischendorf sowie Thomas Horn und zweitens die Gruppe zum Entwurf heterogener Systeme mit Dr. Marco Dienel, Dr. Michael Schlegel, Peter Wolf, Christian Roßberg und Hailu Wang.

Darüber hinaus möchte ich meiner Familie, insbesondere meiner Frau und meinen Kindern, für die aufgebrachte Geduld und die Schaffung zeitlicher Freiräume zur Anfertigung der Arbeit danken.

Die praktischen Anteile dieser Arbeit wurden gefördert von:

- der Deutschen Forschungsgemeinschaft (DFG) im Rahmen des Teilprojekts A 2 „Systementwurf“ des Sonderforschungsbereichs 379 „Mikromechanische Sensor- und Aktorarrays“ und
- dem Bundesministerium für Bildung und Forschung (BMBF) im Rahmen des Projekts „MxMobile - Multi-Standard Mobile Platform“ unter Förderkennzeichen 01BU0662.

1. Einleitung

1.1. Motivation

Die fertigungstechnisch mögliche Komplexität heutiger Systeme steigt stetig an. Im digitalen Bereich folgt dieser Anstieg dem Mooreschen Gesetz [1], das eine Verdopplung der Transistordichte in ICs aller 18 Monate vorhersagt. Auch im Analogbereich ist ein Anstieg feststellbar. So verdoppelt sich die Komplexität von A/D-Wandlern alle fünf Jahre [2]. Die damit mögliche Systemkomplexität ist selbst für größere Teams von Entwerfern kaum noch zu beherrschen, sich bietende Chancen durch höhere Integrationsdichten können mangels Entwurfsressourcen nicht mehr voll genutzt werden [3]. Das sich ausbildende *Design Gap* zwischen der Produktivität der Entwerfer und den technologischen Möglichkeiten zeigt diese Problematik in Bild 1.1 grafisch auf. Um hochkomplexe Schaltungen in ihrer Gesamtheit entwerfen und simulieren zu können, muss auf höhere Abstraktionsebenen ausgewichen werden.

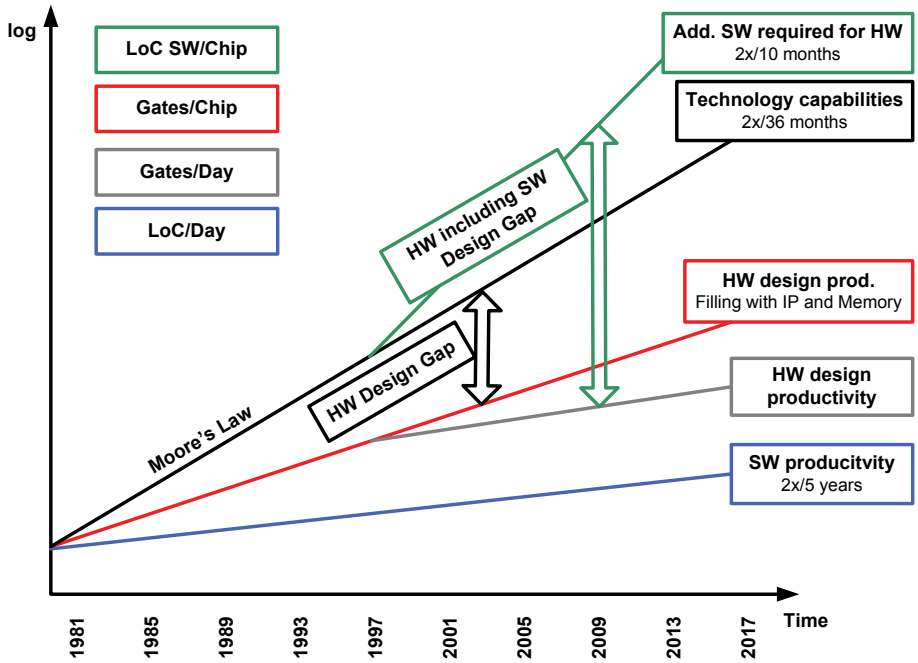


Bild 1.1.: *Design Gap im Jahr 2007 nach ITRS [3]*

Der Digitalentwurf besitzt im Bezug auf die abstrakte Systembeschreibung eine Vorreiterrolle. Durch die gestiegenen Transistordichten war es bereits in den 80er Jahren

des 20. Jahrhunderts nötig, automatisierte Entwurfsabläufe zu nutzen. So ist seit etwa 15 Jahren der Entwurf auf Register-Transfer-Ebene mit Hochsprachen wie VHDL (Very High Speed Integrated circuit Hardware Description Language) [4] und Verilog [5] üblich. Daraus werden Gatternetzlisten und für die Fertigung nötige Daten wie Belichtungsmasken automatisch erzeugt. Ein manueller Entwurf auf Transistorebene ist bei heute mehr als einer Milliarde Transistoren pro Chip (Beispiel Intel Itanium) nicht ökonomisch sinnvoll zu realisieren. Auch bildet die auf jeder manuellen Entwurfsebene nötige Fehlerabsicherung bei dieser Abstraktionsebene ein fast unüberwindbares Hindernis zur Gewährleistung der korrekten Funktion.

Im Entwurf von elektrisch analogen Schaltungsteilen sowie im Mikromechanikentwurf werden derartige Abstraktionsmöglichkeiten noch kaum genutzt. Es überwiegt der Entwurf „nahe am Silizium“ bzw. „nahe an der diskreten Schaltung“, der meist sogar in einem Bottom-Up-Entwurf mündet. Hier besteht die Gefahr, zwar eine perfekte Mikromechanik zu entwerfen, diese jedoch nicht passend zur elektrischen Umgebungsschaltung zu gestalten. Ein Entwurfsstart auf der Systemebene und ein anschließendes Verfeinern der Komponenten bis zur Fertigung vermeidet derartige Probleme. Außerdem kann durch die bereits zu Beginn feststehenden Schnittstellen zwischen den Teilbereichen eine größere Bearbeitungsparallelität erreicht werden. Die Entwerfer von digitaler und analoger Hardware sowie die Softwareentwickler sind so in der Lage, unabhängig vom Fortschritt der Mikromechanikentwerfer ihre Module zu erstellen und zu testen. Die Zeit von der ersten Projektidee bis zum fertigen Produkt wird so drastisch verkürzt, die sogenannte „Time-to-Market“ sinkt deutlich.

1.2. Begriffe und Grundlagen der Modellierung

1.2.1. Begriffe

Die wissenschaftlichen Untersuchungen zum Entwurf von Systemen zeichnen sich durch eine hohe Dynamik aus. Damit geht jedoch eine Diversifizierung von Begriffen einher, so dass die Notwendigkeit besteht, im folgenden die für diese Arbeit gültigen Begriffsdefinitionen darzustellen.

- Der Begriff **System** leitet sich aus dem griechischen systema, „das Gebilde, Zusammengestellte, Verbundene“ ab und bezeichnet ein Gebilde, dessen wesentliche Elemente (Teile) so aufeinander bezogen sind und in einer Weise wechselwirken, dass sie (aus einer übergeordneten Sicht heraus) als aufgaben-, sinn- oder zweckgebundene Einheit (d. h. als Ganzes) angesehen werden und sich in dieser Hinsicht gegenüber der Umwelt abgrenzen.

- Nach DIN 44300 bezeichnet **digital** eine Darstellungsart von Daten mit den Ziffern eines Zahlensystems, bei der die einzelnen Zeichen voneinander abgrenzbar sind. Eine **digitale Darstellung** bezeichnet nach [6] die Abbildung des Werteverlaufs einer Größe durch eine unstetige Funktion, die nur bestimmte diskrete Werte annehmen kann.

- **Analoge** Signale sind Größen, die einen stetigen Verlauf haben und innerhalb festgelegter Grenzen jeden Wert annehmen können [7].
- Die DIN-Norm 44300 definiert **Hardware** als „Gesamtheit oder Teil der apparativen Ausstattung von Rechensystemen“. Das Gegenstück dazu bildet **Software**, welche nach DIN 44300 „Gesamtheit oder Teil der Programme für Rechnersysteme“ umfasst. Hardware bezeichnet also die materiellen Systemkomponenten, wogegen Software die immateriellen Anteile darstellt.
- Ein **Mixed-Signal-System** im Sinne dieser Arbeit besteht aus analogen und digitalen elektrischen Komponenten.
- Ein **heterogenes System** im Sinne dieser Arbeit erweitert ein Mixed-Signal-System um nichtelektrische Komponenten.
- Ein **Mikrosystem** ist im Rahmen des 4. EU-Forschungsrahmenprogramms [8] definiert als ein intelligentes miniaturisiertes System, das sensorische, datenverarbeitende und aktuatorische Funktionen umfasst [9].
- Ein **Makromodell** bildet die Funktion einer Schaltung durch Zusammenschaltung von simulatoreigenen Grundelementen nach, wobei eine hinreichend genaue Beschreibung des Klemmenverhaltens unter allen relevanten Betriebsbedingungen angestrebt wird [10].
- Ein **Verhaltensmodell** ist die Beschreibung des physikalischen Verhaltens einer Komponente (physikalisches Verhaltensmodell) oder die Beschreibung des Klemmenverhaltens einer Komponente (empirisches Verhaltensmodell) [11].
- Im Rahmen dieser Arbeit wird der Begriff **Strukturmodell** wie schon in [11] mit der Bedeutung „hierarchisches Modell“ verwendet.
- **Simulation** ist die Nachbildung des Ist-Verhaltens der entworfenen Systeme, Komponenten oder (Teil-)Schaltungen mit softwaretechnischen Mitteln [12].
- **Emulation** ist die Nachbildung des Ist-Verhaltens der entworfenen Systeme, Komponenten oder (Teil-)Schaltungen mit hardwaretechnischer Unterstützung [12].
- **Verifikation** ist der Vergleich des geforderten Soll-Verhaltens mit dem im Ergebnis der Entwurfsaktivitäten entstandenen Ist-Verhalten [12].
- Bei einer **Abstraktion** werden reale Zusammenhänge vereinfacht dargestellt. Damit einher geht meist ein Verlust an Genauigkeit, wobei jedoch der Simulationsaufwand sinkt. Im Systementwurf unterscheidet man mehrere Abstraktionsebenen, die im folgenden Abschnitt kurz erläutert werden.
- Eine **Synthese** im Sinne des Systementwurfs bezeichnet die Transformation einer System- oder Komponentenbeschreibung von einer höheren Entwurfsebene in eine niedrigere.

1.2.2. Abstraktionsebenen und Entwurfsmodelle

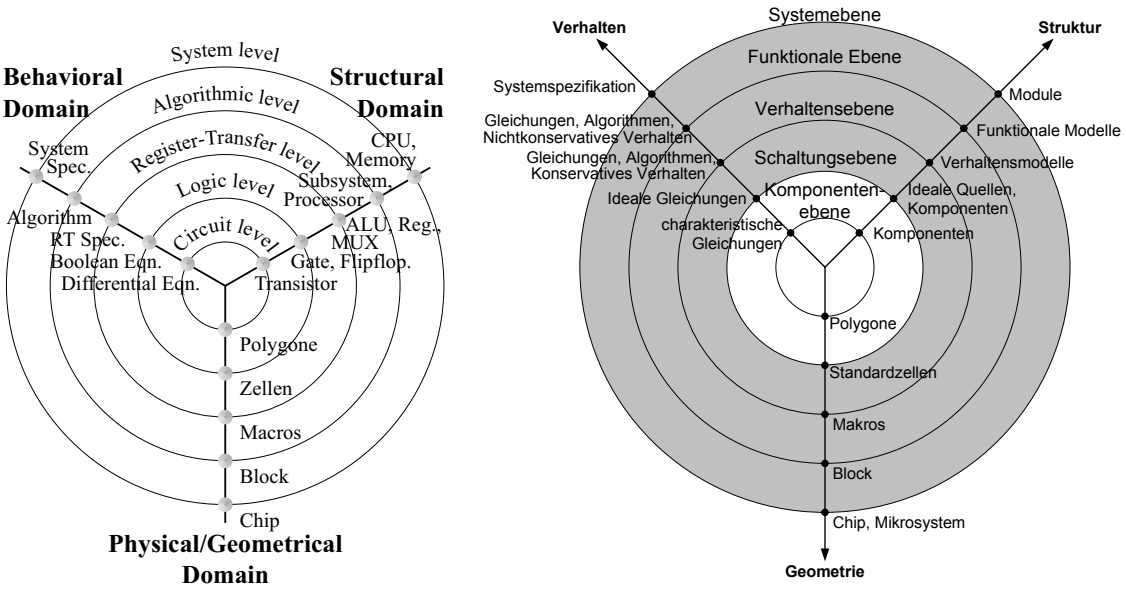
Y-Diagramme

Für den Entwurf und die Simulation von Systemen existieren mehrere Abstraktionsebenen. Jede Ebene beschreibt einen gewissen Abstraktionsgrad der enthaltenen Modelle von der Realität. Eine Simulation auf unterster Abstraktionsebene entspricht einer Darstellung aller physikalischer Zusammenhänge, die das Modell direkt oder indirekt beeinflussen. Dies führt zu enormen Simulationszeiten, sofern eine vollständige Beschreibung auf dieser Ebene überhaupt möglich ist. Daher muss das Systemverhalten abstrahiert werden. Zur Veranschaulichung der Zusammenhänge zwischen den Ebenen verwendet man grafische Darstellungen.

Beim Entwurf digitaler Schaltungen greift man meist auf das in Bild 1.2(a) dargestellte Y-Diagramm von Gajski und Walker [13] zurück. Gajski und Walker unterteilen den Entwurf in fünf Abstraktionsebenen [12]: Auf Systemebene erfolgt die Festlegung globaler Eigenschaften mit einer ersten groben Partitionierung. Die Verarbeitungsalgorithmen sowie die Festlegung von Befehlssätzen und Datenbreiten etc. sind Gegenstand der Algorithmischen Ebene. Die beiden abstraktesten Ebenen besitzen noch kein Zeitmodell sondern basieren auf Kausalitäten. Der Entwurf digitaler Hardware findet heute meist auf Register-Transfer-Ebene (RT-Ebene) statt. Hier wird das Systemverhalten durch Einsatz von Verarbeitungseinheiten und von Registern/Speichern auf reale Hardwarestrukturen mit Hilfe von Hardwarebeschreibungssprachen (Hardware Description Languages - HDL) abgebildet bzw. implementiert. Die Transformation von RT-Ebene zur Logikebene erfolgt in der Regel automatisiert. Eine Logikebenen-Systembeschreibung besteht aus booleschen Gleichungen, wobei für die verwendeten Grundgatter Makrobibliotheken zum Einsatz kommen. Die unterste Abstraktionsebene umfasst die Transistorrepräsentation des Systems. Dies kann in Form einer Netzliste oder in Form von fertig dimensionierten Layoutdaten geschehen.

Die Achsen des Y werden von den drei Domänen Struktur, Verhalten und Geometrie gebildet. Beim Entwurf eines Schaltkreises durchfährt der Designer die Abstraktionsebenen und wechselt nach Bedarf die Domänen bis die Implementierung durch Festlegung aller Domänenparameter abgeschlossen ist. So wird ausgehend von der Spezifikation (Verhaltensdomäne) die Basisarchitektur eines Systems (Prozessorsystem, rekonfigurierbare oder kundenspezifische Lösung) in der Strukturdomäne festgelegt, welche wiederum als Basis für die notwendigen Algorithmen (Verhaltensdomäne) dient.

Zum Y-Diagramm von Gajski-Walker exitistiert im Analogbereich eine von Schlegel [11] entwickelte Entsprechung als Kombination der Arbeiten von Huss [14] und Moser [15]. Das in Bild 1.2(b) dargestellte Diagramm zeigt gegenüber dem digitalen Diagramm zwar denselben geometrischen Aufbau, die Entwurfsschritte in Verhalten und Struktur differieren jedoch. So erfolgt die Verhaltensbeschreibung prinzipiell durch Angabe von Gleichungen, eine ereignisbasierte Beschreibung wie im Digitalbereich ist hier nicht möglich.



(a) digitale Systeme [12]

(b) analoge Systeme [11]

Bild 1.2.: *Y-Diagramme für den Entwurfsablauf*

Auf funktionaler Ebene kommen zur Systembeschreibung nichtkonservative gerichtete rückwirkungsfreie Netze zum Einsatz, welche auf Verhaltensebene in konservative Netze, wo Maschen- und Knotenpunktsatz gelten, überführt werden. Damit einher geht in der Regel ein Wechsel der Beschreibungsmittel von abstrakten gerichteten Simulationswerkzeugen wie Matlab/Simulink zu Netzwerklösern wie Spice. Eine Brücke zwischen diesen beiden Ebenen stellen AMS-Sprachen wie SystemC-AMS und VHDL-AMS dar, die sowohl gerichtete als auch konservative Datenströme erlauben. Diese beiden Sprachen sind Gegenstand des Kapitels 4.2. Die Simulation von mechanischen Strukturen sowie von elektrischen oder magnetischen Feldeffekten auf Komponentenebene ist Gegenstand von Spezialsimulatoren, die in der Regel auf der Finite-Elemente-Methode (FEM) aufsetzen.

Die grau gezeichneten Kreise kennzeichnen die in dieser Arbeit bearbeiteten Entwurfsebenen Spezifikation, Funktion und Verhalten. Für die Entwurfsphasen auf Schaltungs- und Komponentenebene existieren ausgereifte Werkzeuge wie beispielsweise diverse Spice- und FEM-Simulatoren. Die neue Entwurfsmethodik für Mikrosysteme beginnend mit der Spezifikation bis hin zur Verhaltensbeschreibung bildet den Gegenstand des Kapitels 8.

Entwurfsverfahren

Die beiden Y-Diagramme bieten die Möglichkeit, die Entwurfsverfahren kurz zu erläutern. Der *Top-Down-Entwurf* durchläuft die Diagramme von außen nach innen und arbeitet damit die Ebenen von oben nach unten („top-down“) ab. Ausgehend von der Spe-

zifikation entsteht zunächst eine Systembeschreibung welche ggf. durch Wechsel der Achsen bis hin zum fertigen Layout auf Geometrieebene konkretisiert wird. Dies bietet den Vorteil, das System als Ganzes betrachten zu können, jedoch ist es insbesondere im Analogbereich möglich, dass sich die abstrakten Systembeschreibungen mangels technologischer Mittel nicht in die Implementierung umsetzen lassen.

Dieses Problem umgeht der *Bottom-Up-Entwurf*, indem er ausgehend von den technologischen Möglichkeiten auf Geometrieebene ein mögliches System erstellt. Von Nachteil ist hierbei, dass die Schnittstellen zu anderen Modulen erst spät feststehen und damit das Risiko einer Nichtverbindbarkeit besteht bzw. der Entwurf der Umgebungs- und Auswerteschaltung erst spät beginnen kann.

Im Entwurf heterogener Systeme hat sich daher eine Mischung aus beiden Ansätzen etabliert, der als *Meet-in-the-Middle* oder *iteratives Vorgehen* bezeichnet wird. Hier beginnt der Entwurfsprozess wie beim Top-Down-Entwurf, die Einzelkomponenten werden jedoch Bottom-Up entworfen.

Das V-Modell

Als Spezialisierung des Wasserfallmodells hat sich in Deutschland das V-Modell [16] etabliert. Es beschreibt den allgemeinen Entwurfsablauf für technische Systeme und wird bei öffentlichen Aufträgen seitens der Bundesbehörden vorausgesetzt. Bei den zivilen und militärischen Vorhaben des Bundes ist dieser Entwicklungsstandard durch die Koordinierungs- und Beratungsstelle für Informationstechnik in der Bundesverwaltung (KBSt) empfohlen worden. Damit ist das V-Modell XT für alle Ressorts der Bundesbehörden anzuwenden. Aus der Ursprungsversion entstanden mehrere abgeleitete Diagramme, in dieser Arbeit soll das in Bild 1.3 dargestellte V-Modell von Heinkel [17] betrachtet und genutzt werden.

Die linke Seite des Modells beschreibt die Phase der Systementwicklung vom Allgemeinen hin zum Speziellen (Top-Down-Entwurf). Abgeleitet von den Anforderungen des Auftraggebers bzw. Kunden, welche in der Regel in Form eines „Lastenhefts“ vorliegen, wird zunächst ein Systemkonzept erstellt. Die Überführung der Kundenanforderungen in funktionale Systemanforderungen ermöglicht meist auch ein erstes simulierbares Systemmodell auf hoher Abstraktionsebene. Ausgehend von dieser Funktionsbeschreibung erfolgt die Auswahl der Zielarchitektur sowie die Partitionierung in analoge und digitale Hardware sowie Software. Auf dieser Ebene können bereits getestete Elemente aus vorhergehenden Projekten im Rahmen des „Re-Use“ in das System integriert werden. Dies spart Entwurfs- und Testkosten und beschleunigt den Designprozess. Die Partitionierung bildet den Ausgangspunkt für die Definition von Subsystemen und dem als Boden des ’V’ aufgeführten Entwurf der Komponenten.

Der rechte Ast bezeichnet die Phase der Integration der Teilmodule in das Gesamtsystem. Hier wird in umgekehrter Reihenfolge zum linken Ast das System aus seinen Komponenten schrittweise zusammengesetzt und jedes entstehende strukturelle Modul

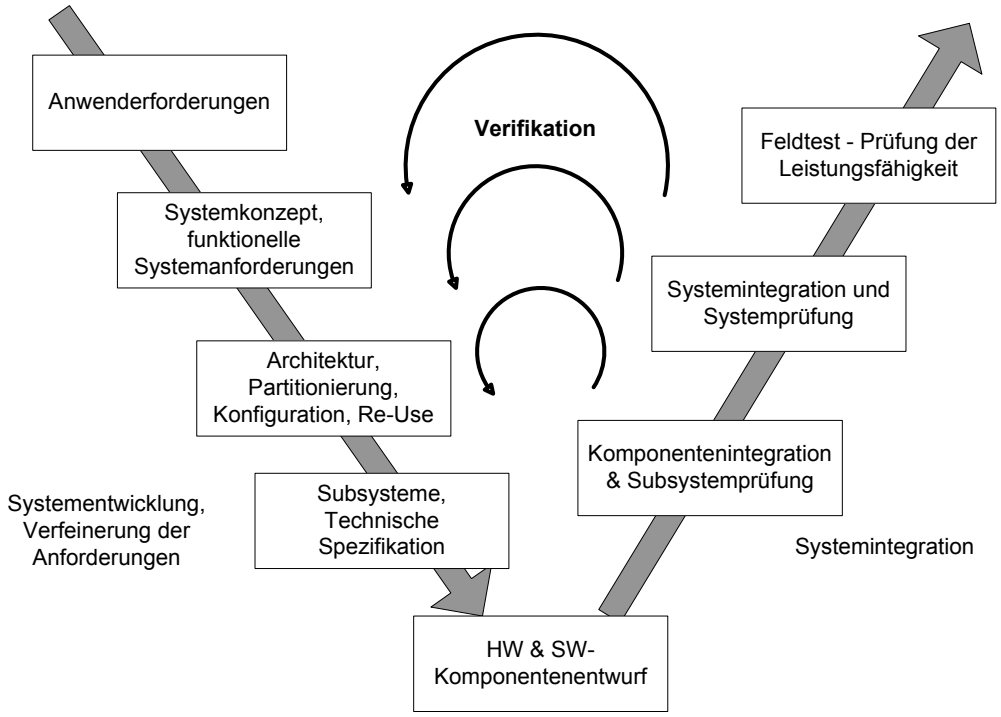


Bild 1.3.: *V-Modell für den Systementwurf [17]*

umfangreich getestet. Je früher ein Fehler entdeckt wird, umso geringere Kosten fallen zu seiner Beseitigung an. Spätestens der Feldtest sollte alle Schwächen des Systems aufzeigen und somit ein im Idealfall fehlerfreies System ermöglichen.

Zwischen beiden Ästen muss die Konsistenz durch Verifikation sichergestellt werden. Dabei sollten immer die Ausgangsanforderungen im Blick behalten werden, da sonst die Gefahr besteht, eine falsche Komponentenbeschreibung zwar richtig zu implementieren aber trotzdem das Entwurfsziel nicht zu erreichen. Das Modell hilft, einen der Spezifikation entsprechenden und nachprüfbaren Entwurfserfolg zu erreichen und dient damit auch der Entwurfsdokumentation.

2. Ausgangspunkt und Ziel der Arbeit

2.1. Ausgangspunkt

In der Mikrosystemtechnik ist ein gemischter Ansatz aus Top-Down und Bottom-Up üblich [18], der auch als *iteratives Vorgehen* bezeichnet wird. Darüber hinaus existieren einige Ansätze zur reinen Top-Down Entwurfsmethodik. Ein durchgehender Entwurfsfluss wird z. B. von Swart [19] vorgestellt. Dieser nutzt als höchste maschinenlesbare Sprache Spice [20], einen reinen Analogsimulator. Mikrosysteme (microelectromechanical systems - MEMS) sind im Sinne dieser Arbeit nur die analogen elektrischen und nichtelektrischen Anteile, jedoch nicht die dazugehörige digitale Auswertung und die darauf gegebenenfalls laufende Software. Ebenfalls nur auf die Analogkomponenten von MEMS konzentriert sich Mukheljee [21]. Einen Ansatz zur Synthese von analogelektrischen Systemteilen schildert Kampe [22], er gibt gleichzeitig einen Überblick über den prinzipiellen Ablauf von Synthesen.

Im Gegensatz zu diesen Arbeiten nutzen Grimm et al. [23] SystemC-AMS zur Beschreibung aller Systemkomponenten unter Verwendung einer Simulatorkopplung mittels polymorpher Signale. Eine automatisierte Konvertierung der Modelle von SystemC-AMS in die entsprechenden Spezialsimulatoren ist nicht Teil von [23], hier muss der Code jeweils manuell neu erstellt werden. Im Rahmen der Untersuchungen zur Simulatorkopplung laufen auch Untersuchungen zur Integration hybrider Graphen; es wurde bisher jedoch kein Zusammenhang zur Spezifikationsebene dargestellt, und die Verzahnung der hybriden Graphen mit SystemC-AMS ist nur als Theorie veröffentlicht worden. Im Vorfeld der Arbeiten zu SystemC-AMS entstand das Werkzeug KANDIS [24], das ausgehend von Komponentenbeschreibungen in VHDL-AMS (siehe Abschnitt 4.2) unter Nutzung hybrider Datenflussgraphen eine Spezifikationsumgebung für heterogene Systeme anbietet.

Den Ausgangspunkt der vorliegenden Arbeit bilden die Untersuchungen von Barthel zur Spezifikationserfassung heterogener Systeme [10]. Die textuelle Spezifikation wurde in dieser Arbeit unter Nutzung von HDL-A [25] und eines Hypertext-Konzepts formalisiert und simulierbar gemacht. HDL-A als Vorgängersprache von VHDL-AMS erlaubt die Abbildung von digitaler und analoger Hardware auf Basis von Differentialgleichungen. Zusätzlich werden die Entwurfsentscheidungen durch Entwicklung eines Entscheidungsmodells erleichtert und dokumentiert. Aus der HDL-A-Beschreibung heraus erfolgt eine automatisierte Dokumentengenerierung im Hypertextformat für die Schnittstellen und Modulparameter, das Werkzeug ASPECTOR [10] verbindet Spezifikationseingabe, Simulation und Dokumentation.

Mit zunehmender Komplexität heterogener Systeme reicht die Abstraktionsebene der Differentialgleichungen nicht mehr aus, die Zusammenhänge aller Systemteile in annehmbarer Zeit zu modellieren und zu simulieren. Daher ist sowohl ein Wechsel der Beschreibungsebene als auch ein Wechsel der Beschreibungsmittel notwendig, die bisher genutzten Sprachen zur Modellierung der Komponenten erfüllen die Anforderungen hinsichtlich Rechenzeit und Entwurfsaufwand nicht mehr.

2.2. Verwandte Projekte

Der Entwurf heterogener Systeme wird von aktuell verfügbaren Tools kaum unterstützt. Meist konzentrieren sich die Hersteller auf eine Domäne, stellen also entweder den Analogentwurf oder den Digitalentwurf in den Mittelpunkt. Es befinden sich jedoch mehrere Projekte in Arbeit, die Entwicklung und Verifikation heterogener Systeme verbessern sollen.

- Das im Juni 2006 gestartete VeronA-Projekt [26] beschäftigt sich mit der Verifikation analoger Schaltungen. Dabei sollen Elemente einer durchgängigen Verifikationsmethodik integrierter analoger Schaltungen geschaffen werden.

- Seit Juli 2004 erforscht das Team des FEST-Projektes [27] die funktionale Verifikation von heterogenen Systemen. Für analoge Sachverhalte kommt dabei eine Erweiterung der Spezifikationssprache CTL (Computational tree logic) zum Einsatz [28]. Ein Link zu SystemC-AMS befindet sich in der Entwicklung [29].

- Das Projekt DETAILS [30] erforscht die Unterstützung des Entwurfs von Mobilfunkanwendungen. Dabei wird SystemC-AMS zur Modellierung eines Gigabit radio transceivers [31] eingesetzt.

- Am Forschungszentrum Informatik (FZI) Karlsruhe entsteht mit KaSCPar (Karlsruhe SystemC Parser Suite) [32] ein Tool, das SystemC nach XML (eXtensible Markup Language) [33] konvertiert. Die Lizenz des Programms verbietet jedoch Veränderungen im Programmcode, so dass eine Erweiterung für SystemC-AMS nur schwer zu realisieren ist.

- Im Rahmen eines Transregio-Sonderforschungsbereichs bearbeiten die Universitäten Freiburg, Saarbrücken und Oldenburg das Projekt AVACS [34], das auf Basis Hybrider Automaten die Verifikation heterogener Kontrollstrukturen in Fortbewegungsmitteln (Flugzeug, Kfz, Bahn) untersucht.

- Im Juli 2009 startete das EDA-Clusterforschungsprojekt ROBUST [35], welches die Verbesserung der Robustheit analoger und digitaler Schaltungen der Nanoelektronik zum Ziel hat. Das Projekt will dazu SystemC-AMS als Systembeschreibungssprache nutzen und in diesem Zusammenhang auch hybride Automaten untersuchen. Dazu ist es nötig, SystemC-AMS um Möglichkeiten zur Parametervariation zu erweitern.

2.3. Ziel der Arbeit

Ziel der Arbeit ist es, die Grundlagen für einen durchgängigen *Top-Down-Entwurfsfluss*, beginnend bereits mit der Spezifikation, zu schaffen. Dabei soll in den einzelnen Entwurfsphasen eine Prüfung der Systemfunktion durch Simulation stets möglich sein. Die Spezifikation soll als simulative Testumgebung (sog. „ausführbare Spezifikation“) für das System fungieren. Ein inertiales Navigationssystem (INS) mit Komponenten aus der Mikrosystemtechnik sowie analoger und digitaler Elektronik zeigt die Methodik an einem realen Beispiel. Darüber hinaus enthält das INS Software und bildet damit ein realistisches heterogenes System.

Der durchgängige Entwurfsfluss bedingt die Unterstützung verschiedener Ebenen im Designprozess. Bild 2.1 zeigt dazu die beteiligten Entwurfsschritte und Simulationsverfahren, dabei stellt der grau hinterlegte Bereich die im Rahmen der Arbeit erstellten bzw. bearbeiteten Anteile dar.

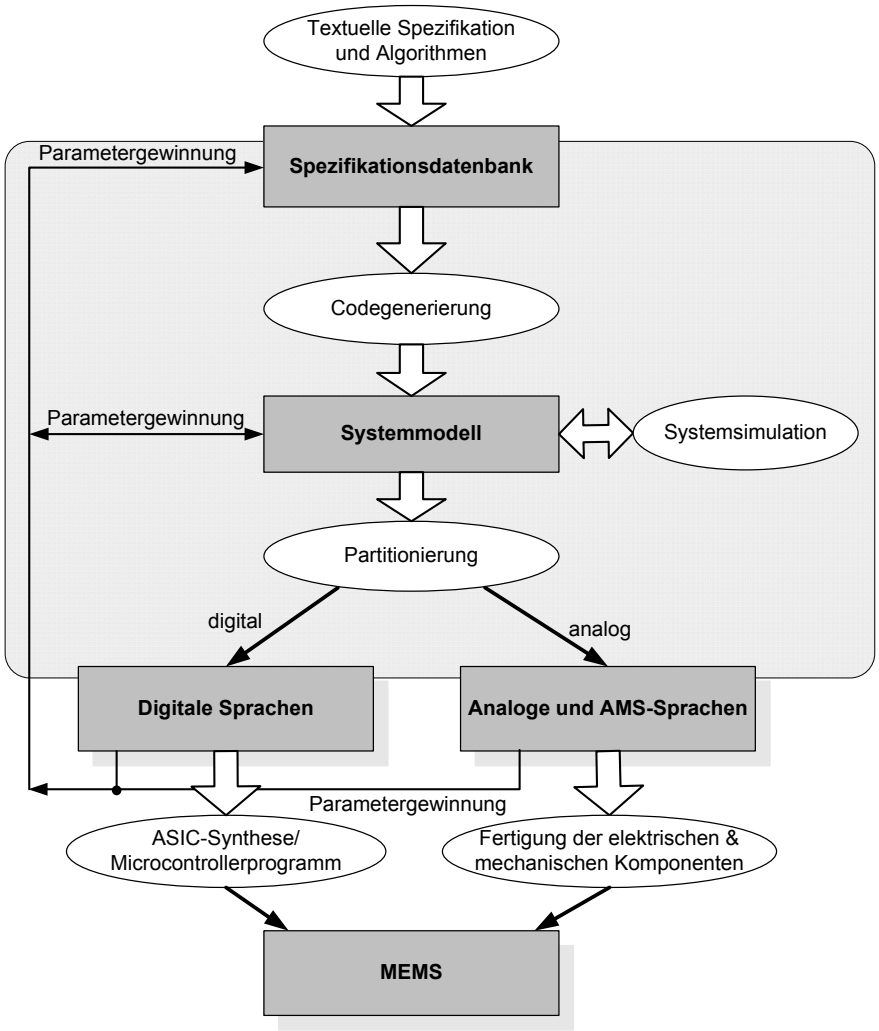


Bild 2.1.: *Abstrakter simulationsbezogener Entwurfsfluss für heterogene Systeme*

Die Grundvoraussetzung für eine weitere Verarbeitung stellt die Formalisierung der textuellen Spezifikation dar, was bei einem Spezifikationsumfang in der Industrie von teilweise mehr als 1000 Seiten die Nutzung einer Datenbank erfordert. Die Formalisierung ermöglicht eine weitere rechentechnische Verarbeitung der Texte und damit eine Umsetzung in simulierbaren Code. Diese Systemebenenbeschreibung erlaubt eine Simulation des Gesamtsystemverhaltens auf hoher Abstraktionsebene, auf deren Basis eine Systempartitionierung erfolgt. Im Rahmen der Partitionierung sollten die Realisierungskosten für unterschiedliche Implementierungsvarianten berücksichtigt werden, die Weiterbearbeitung der einzelnen Systemanteile ist Gegenstand der Spezialwerkzeuge der Domänen, wie z. B. Spice-Simulatoren für elektrische Netze oder der Anschluss eines VHDL-basierten Entwurfsprozesses für digitale Teile. Die bei der Verfeinerung der Module gewinnbaren Parameter sind in die Entwurfsdatenbank zu überführen und können so zur Dokumentation des Systems beitragen.

Der weitere Entwurf und die Fertigung der analogen elektrischen und nichtelektrischen Komponenten stellt ein komplexes und umfangreiches Arbeitsgebiet dar, was jedoch nicht Gegenstand dieser Arbeit ist. Wie zu Beginn dieses Kapitels erläutert, überwiegt hier ein iteratives Vorgehen als Mischung von Top-Down und Bottom-Up Entwurf. In dieser Phase kommt eine fast unüberschaubare Anzahl an Spezialwerkzeugen für die jeweiligen Domänen, auch abhängig von der Vorliebe der Entwerfer, zum Einsatz. Eine generische Unterstützung dieser Entwurfsphase erscheint aufgrund der Vielzahl der nötigen Lösungen nicht sinnvoll umsetzbar.

Zu lösende Aufgaben

Diese Arbeit nutzt soweit wie möglich vorhandene Werkzeuge, da diese schon eine gewisse Akzeptanz bei den Entwerfern besitzen und sich damit Anknüpfungspunkte an vorhandene Designflows ergeben. Für die Verbindung dieser Werkzeuge sowie für die Aufnahme der Spezifikation in formaler Form mussten jedoch neue Lösungen gefunden werden. Dabei erlauben die Schnittstellen der neugeschaffenen Tools eine Erweiterung hin zu anderen Werkzeugen, um flexibel auf diesen sich ständig in Bewegung befindlichen Bereich der Technikentwicklung reagieren zu können.

Zur Spezifikationserfassung soll das Werkzeug SpecScribe dienen, dessen Entwicklung im Rahmen dieser Arbeit wesentlich vorangetrieben wird. Für die Problemstellung des Entwurfs von Mikrosystemen sind im Speziellen Erweiterungen um Schnittstellen zu Simulationssprachen sowie um Konstrukte zur Erfassung und Verarbeitung heterogener Systeme nötig. Dies beschränkt sich nicht auf funktionale Parameter des elektrischen Teilsystems, sondern umfasst auch Daten der nichtelektrischen Teile und der die Teilsysteme verbindenden Elemente wie beispielsweise Leiterplatten. Das aus den formalisierten Anforderungen entstehende Systemmodell stellt im Rahmen der neu entwickelten Entwurfsmethodik eine ausführbare Spezifikation dar, die in Zusammenhang mit den textuellen Anforderungen eine erste Prüfung des gewünschten Verhaltens ermöglicht und damit hilft, frühzeitig Fehler in der Spezifikation aufzudecken.

Eine automatisierte Codegenerierung soll den Weg zum ersten Systemmodell stark vereinfachen. Mit der nutzergesteuerten Verfeinerung des Systemmodells geht eine Aufteilung in die einzelnen Domänen (Software, digitale und analoge Hardware sowie nichtelektrische Teile) einher. Dies unterstützt die Arbeit durch Integration einer Kostenparameterverknüpfung bei der Systemsimulation. Zur Anknüpfung an vorhandene Designflows soll im Rahmen der Arbeit ein Konvertierungswerkzeug entstehen, welches die Teile des Systemmodells in die entsprechenden Spezialsimulatoren überführt.

Die folgenden Kapitel beschreiben die Umsetzung dieses Flows mit derzeit (Stand: 2008) verfügbaren Werkzeugen. Dabei werden zuerst die prinzipiellen Anforderungen an die Modellierung heterogener Systeme ausgehend von den beteiligten Domänen analysiert. Diese Anforderungen bilden die Basis für die Untersuchung der verfügbaren Werkzeuge in Kapitel 4. Das Kapitel 5 schildert die Erweiterung des Spezifikationswerkzeugs SpecScribe für heterogene Systeme, an das sich Beschreibungen der darüber hinaus notwendigen Konvertierungswerkzeuge und Kostenparameterverknüpfungen anschließen. Der in Kapitel 8 vorgestellte konkrete Entwurfsablauf wird in Kapitel 9 auf das Beispiel Inertialnavigationssystem angewandt. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick.

3. Bestandteile von Mikrosystemen, Modellierungsanforderungen

Mikrosysteme im Sinne dieser Arbeit verbinden alle physikalischen Domänen miteinander. Damit spannt sich für die Modellierung ein großer Verknüpfungsraum auf. Im folgenden sollen die wichtigsten Wechselwirkungen erläutert und daraus Anforderungen an die Beschreibungsmittel auf Systemebene abgeleitet werden, eine Kurzübersicht der Verknüpfungen in englischer Sprache befindet sich in [36].

3.1. Physikalische Domänen in Mikrosystemen

Mechanische Domäne

Die breite Nutzung von Mikrosystemen wurde erst durch die Anwendung der Technologien der Mikroelektronik auf die Fertigung von Mikromechaniken und die damit einher gehende Senkung der Produktionskosten möglich. Vorher dominierten rein elektrische Systeme mit externen mechanischen Signalaufnehmern. Die heute mögliche Integration kleinster Mechaniken in Silizium lässt die Mechanik mit der zugehörigen Elektronik zu einem Gesamtsystem verschmelzen, wenn auch die Zusammenfassung beider Teile auf einem gemeinsamen Substrat noch die Ausnahme darstellt. Der Prozess der Integration macht es natürlich auch nötig, das System vom Entwurf her ganzheitlich zu betrachten, da zwischen den Systemkomponenten Wechselwirkungen auftreten.

Die mechanische Domäne unterteilt sich in zwei Bereiche: die translatorische Mechanik, basierend auf geradlinigen Bewegungen, und die rotatorische Mechanik für Kreisbewegungen. Auch die Gesetze von mechanischen Wellen sind Teil dieser Domäne.

Elektrische Domäne

Neben der Mikromechanik bildet die elektrische Domäne den Kernbereich von Mikrosystemen. Dies wird auch in der im Englischen üblichen Bezeichnung für Mikrosysteme, MEMS (microelectromechanical systems) deutlich. Viele der heute gefertigten Mikrosysteme wandeln mechanische Erregungen in elektrische Signale (Sensorprinzip) oder setzen elektrische Steuersignale in mechanische Bewegungen um (Aktorprinzip). Die elektrische Domäne wird für den Entwurf in den analogen und digitalen Bereich unterteilt, wobei der digitale Bereich im Prinzip lediglich eine Abstraktion des analogen Teilbereichs darstellt.

Magnetische Domäne

Eng mit der elektrischen ist die magnetische Domäne verbunden. Jedes sich ändernde elektrische Feld verursacht magnetische Effekte, und Magnetismus führt zur elektrischen Induktion. Durch neue Produktionsprozesse, wie beispielsweise LIGA [37], ist es möglich, in Mikrosystemen durch Integration von Spulen Magnetfelder zu erzeugen. Das Einfügen von miniaturisierten Permanentmagneten ist ebenfalls möglich.

Thermische Domäne

Die thermische Domäne hat (oft unerwünschten) Einfluss auf alle anderen Domänen. So laufen chemische Reaktionen meist nur in einem bestimmten Temperaturbereich ab. Die elektrischen, magnetischen und mechanische Eigenschaften eines Stoffes ändern sich ebenfalls mit wechselnden Temperaturen. Bei allen Mikrosystemen ist eine Berücksichtigung der thermischen Eigenschaften unerlässlich für eine Beschreibung des realen Systemverhaltens, dabei stellt die Eigenerwärmung durch Verlustleistung im elektrischen Bereich einen sehr kritischen Punkt im Entwurf dar. Der Aufwand zur Kühlung von Bauteilen steigt mit wachsendem Integrationsgrad und damit steigender Verlustleistungsdichte immer weiter an und muss bereits auf Systemebene berücksichtigt werden. Aber auch aktive thermische Prozesse stehen im Blickpunkt von Mikrosystemen, so kommen z. B. Mikroheizungen in Kleinstreaktoren der chemischen Analytik zum Einsatz.

Chemische Domäne

Der chemische Bereich rückte in den vergangenen Jahren verstärkt in den Blickpunkt der Mikrosystemtechnik. Bei Untersuchungen chemischer Produktionsprozesse zeigte sich, dass durch die parallele Nutzung vieler kleiner Mikroreaktoren eine Verbesserung der Ausbeute und eine deutlich feinere Prozesssteuerung als in den herkömmlichen makroskopischen Anordnungen möglich ist. Auch in der Analytik sind Mikrosysteme inzwischen weit verbreitet. Sie benötigen weniger Analysestoff als herkömmliche Analysemethoden und ermöglichen neue Verfahren z. B. zur Separation von verschiedenartigen Molekülen.

Strahlung

Unter Strahlung versteht man nicht nur das für den Menschen sichtbare Licht, sondern auch nichtsichtbare Bestandteile des Spektrums. Dies reicht von Infrarotstrahlung (IR) über den ultravioletten Wellenbereich (UV) bis hin zu kosmischen und atomaren Strahlungen wie Alpha-, Beta- und Gammastrahlung. Für alle Wellenbereiche wurden im Laufe des letzten Jahrhunderts Detektoren entwickelt, die durch Weiterentwicklung

und Miniaturisierung in den Bereich der Mikrosystemtechnik vorstoßen. Die Hauptanwendung vom Mikrosystemen für Strahlungen besteht jedoch in der Generierung, Detektion und Veränderung von Licht im sichtbaren sowie im angrenzenden IR- und UV-Wellenbereich, der sogenannten Mikrooptik.

3.2. Berechnung von Netzwerken aus diskreten Elementen

Für die Systemsimulation konzentriert man üblicherweise das Verhalten eines Volumens in einem Punkt und kann so mit diskreten Elementen rechnen. Dies stellt eine weitere Abstraktion der Finiten-Elemente-Methode dar, welche kontinuierliche Strukturen mit einem Gitter aus kleinen Einzelelementen überzieht. Zur Verbindung der diskreten Elemente werden komplexe Flussgrößen λ und Differenzgrößen μ definiert, für die sich im allgemeinen Fall in der Netzwerkdarstellung ein Knotenpunkt- und ein Maschensatz aufstellen lassen (Bild 3.1). Diese Sätze tragen für die elektrische Domäne die Bezeichnung „Kirchhoffsche Sätze“. Für andere Domänen lassen sich daraus Analogien ableiten, die z. B. in Spice-Simulatoren zur Beschreibung nichtelektrischen Verhaltens dienen.

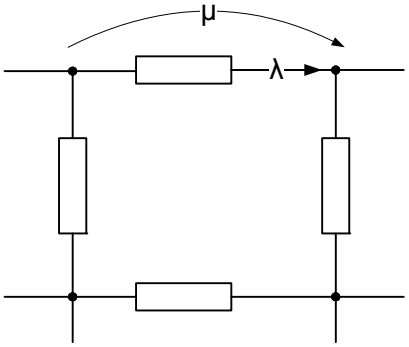


Bild 3.1.: *konservatives Netzwerk mit Fluss- und Differenzgröße*

In einem kirchhoffschen Netzwerk beträgt die Summe aller an einem Knoten anliegenden Flussgrößen λ gleich 0.

$$\sum_{Knoten} \lambda_{anliegend} = 0 \tag{3.1}$$

Innerhalb einer Masche des Netzwerks addieren sich die Differenzgrößen μ zu 0.

$$\sum_{Masche} \mu_{Maschenelement} = 0 \tag{3.2}$$

Netzwerke, in denen diese Gesetze gelten, heissen „konservative Netze“. Die komplexen Fluss- und Differenzgrößen formen darüber hinaus einen aus dem Maschen- und

Knotenpunktsatz herleitbaren grundlegenden Zusammenhang für die Berechnung in linearen Netzen:

$$\alpha \cdot \dot{\lambda} + \beta \cdot \lambda + \gamma \cdot \int \lambda dt = 0 \tag{3.3}$$

Die Parameter α, β und γ beschreiben das Verhalten der linearen Netzwerkelemente. Tabelle 3.1 zeigt die gebräuchlichen Entsprechungen der Parameter in den einzelnen Domänen.

	elektrisch	magnetisch	mechanisch-rotatorisch	mechanisch-translatorisch	thermisch
μ	Spannung	Durchflutung	Winkelgeschwindigkeit	Geschwindigkeit	Temperatur
λ	Strom	magnetischer Fluss	Drehmoment	Kraft	Wärmefluss
α	Induktivität		Drehnachgiebigkeit	Federkonstante	
β	Widerstand	magnetischer Widerstand	Drehreibung	Dämpfung	thermischer Widerstand
γ	Kapazität		Trägheitsmoment	Masse	Wärmekapazität

Tabelle 3.1.: *Analogien in den Domänen (nach [10])*

3.3. Wichtige Gesetze und Wechselwirkungen der Domänen

Zur Beschreibung von hochgradig heterogenen Systemen, wie sie Mikrosysteme darstellen, ist eine hohe Abstraktion der physikalischen Zusammenhänge nötig, da anderenfalls Simulationszeiten und Modellierungsaufwand in intolerable Bereiche ansteigen. Einen Kernpunkt der Systemsimulation stellt die Vereinfachung der physikalischen Grundgesetze dar, so dass auf rechenaufwändige Randbereichsberechnungen weitestgehend verzichtet wird. Dies erfordert immer die Prüfung, ob die gewählte Abstraktion noch zulässig ist. Im folgenden Abschnitt werden die auf Systemebene wichtigsten Berechnungsgesetze und Zusammenhänge dargestellt. Eine umfangreichere Nennung der Koppeleffekte umfasst Paul [36], jedoch fehlen dort die zugehörigen Erläuterungen und Berechnungsvorschriften. Herleitungen für einen Teil der Gleichungen geben Pelesko et al. [38]. In der folgenden Übersicht sind Wechselwirkungen immer der verursachenden Domäne zugeordnet. Die Berechnungsgleichungen wurden, soweit nicht anders angegeben, aus „Technische Formeln“ [39] entnommen.

3.3.1. Mechanik

Beeinflussung innerhalb der Domäne

Eine sich mit der Geschwindigkeit v bewegende Masse m besitzt einen Impuls $p = m \cdot v$. Die Summe aller Impulse in einem abgeschlossenen System ist konstant, sofern keine Energieumwandlung (plastische Verformung) auftritt (*Impulserhaltung*).

Die Formänderungsarbeit (*Deformation*) beim Stoß zweier Massen m_1 und m_2, die sich mit den Geschwindigkeiten v_1 und v_2 bewegen, entspricht dem Verlust an kinetischer Energie E_{kin} bei einem Stoß. Sie kann mittels der Stoßzahl k berechnet werden:

$$\Delta E_{kin} = \frac{m_1 m_2}{2(m_1 + m_2)} \cdot (v_1 - v_2)^2 \cdot (1 - k^2) \tag{3.4}$$

Für den Bereich der *Fluidik bzw. Strömungslehre* existiert eine Analogiebeziehung zur Tabelle 3.1. Dabei entspricht der Massenstrom $\dot{m}$ dem elektrischen Strom I und der Druck p der Spannung U. Damit lassen sich der fluidische Widerstand R_{fluid} mit

$$R_{fluid} = \frac{\Delta p}{\dot{m}}, \tag{3.5}$$

die fluidische Trägheit L_{fluid} mit

$$\Delta p = L_{fluid} \frac{dm}{dt} \tag{3.6}$$

und die fluidische Kapazität C_{fluid}

$$\dot{m} = C_{fluid} \frac{dp}{dt} \tag{3.7}$$

berechnen. Insbesondere in der Fluidik hat die Miniaturisierung einen großen Einfluss auf die Berechnungsgesetze, wie Nguyen [40] schildert.

Beeinflussung anderer Domänen

Der *direkte Piezoeffekt*, wie er in Piezo-Sensoren ausgenutzt wird, erzeugt bei mechanischer Beanspruchung eines Materials eine Potentialdifferenz [41]. Die sich einstellende Polarisation P berechnet sich aus der auftretenden mechanischen Spannung T und dem Tensor der piezoelektrischen Spannungskoeffizienten e

$$P = e \cdot T. \tag{3.8}$$

Durch die Zuführung mechanischer Energie in Form einer Volumenverkleinerung δV bzw. Druckerhöhung δp erhöht sich in gasförmigen Stoffen die *Temperatur* durch die Verrichtung von Volumenänderungsarbeit W (Kompressionsarbeit) [39]

$$W = -\int V_1 V_2 p dV. \tag{3.9}$$

Eine makroskopische Anwendung dieses Effekts stellt der Dieselmotor dar, der ein Gasgemisch bis zur Selbstentzündung verdichtet.

Das *elektrische Feld* ändert sich mit einer Bewegung der Elektroden. Die elektrische Feldstärke E ist abhängig vom Elektrodenabstand s bzw. der Elektrodenüberdeckung und der anliegenden elektrischen Spannung U in der Form

$$E = \frac{dU}{ds}. \tag{3.10}$$

Dieser Effekt ist einfach zu detektieren und findet daher häufig Anwendung in MEMS wie z. B. bei Beschleunigungssensoren.

Darüber hinaus treten Wechselwirkungen mit der thermischen Domäne durch *Thermoelastik* auf, wo z. B. die Biegung eines Siliziumbalkens eine Temperaturänderung hervorruft [42]. Einen vergleichbaren Effekt stellt die *Photoelastizität* dar, wo sich durch Druckeinwirkung die optischen Eigenschaften eines Materials ändern. So können beispielsweise auf eine optische Faser einwirkende Drücke durch zwei unterschiedlich polarisierte Wellen detektiert werden [43].

3.3.2. Elektrik

Beeinflussung innerhalb der Domäne

In Halbleitern nutzt man sogenannte *pn-Übergänge* um Widerstände (bis hin zu Schaltern) zu realisieren [7]. Diese werden in Form von Dioden und Transistoren sowohl im analogen als auch im digitalen Bereich eingesetzt. Die ideale Kennliniengleichung für einen pn-Übergang stellt die Abhängigkeit des Stromflusses I von der anliegenden Spannung U, dem Sperrsättigungsstrom I_S und der Temperaturspannung U_T

$$I = I_S \cdot e^{\frac{U}{U_T}-1} \text{ dar.} \tag{3.11}$$

Neben den pn-basierten Bipolartransistoren kommen spannungsgesteuerte Feldeffekttransistoren zum Einsatz, die die Verlustleistung stark reduzieren.
In einem elektrischen Feld findet eine *Energiespeicherung* statt. Am Beispiel eines Kondensators ergibt sich die gespeicherte Energie W aus der angelegten Ladespannung U_0 und der Kapazität C

$$W = \frac{1}{2} \cdot C \cdot {U_0}^2, \tag{3.12}$$

wobei die Kapazität C von der Geometrie der Anordnung abhängt und gleichzeitig die Ladung durch $Q = C \cdot U$ beschreibt [7].

Beeinflussung anderer Domänen

Die Kopplung zwischen elektrischem und magnetischem Feld beschreiben die *Maxwell-Gleichungen* [7]. In der allgemeinen integralen Form lauten diese:

$$\text{Physikalischer Gaußscher Satz:} \quad \oint\limits_{\delta V} \vec{D} \cdot d\vec{A} = \int\limits_{V} \rho \cdot dV \tag{3.13}$$

$$\text{Quellenfreiheit des Magnetfeldes:} \quad \oint\limits_{\delta V} \vec{B} \cdot d\vec{A} = 0 \tag{3.14}$$

$$\text{Induktionsgesetz:} \quad \oint\limits_{\delta A} \vec{E} \cdot ds = -\frac{d}{dt} \int\limits_{A} \vec{B} \cdot d\vec{A} \tag{3.15}$$

$$\text{Durchflutungsgesetz:} \quad \oint\limits_{\delta A} \vec{H} \cdot ds = \int\limits_{A} \left(\vec{J} + \frac{\delta \vec{D}}{\delta t} \right) d\vec{A} \tag{3.16}$$

Zwischen Ladungen q treten *elektrostatische* und *magnetische Kräfte* in der Form

$$F = q \cdot (\vec{E} + \vec{v} \times \vec{B}) \tag{3.17}$$

auf [7], wobei $\vec{E}$ die auf die Ladung wirkende elektrische Feldstärke, $\vec{v}$ die Geschwindigkeit sowie $\vec{B}$ die magnetische Flussdichte darstellen.
Bei der Simulation auf Systemebene vereinfachen sich diese Gleichungen durch Konkretisierung auf bestimmte geometrische Anordnungen bzw. der Annahme homogener Felder.

Bewegungen elektrischer Ladungsträger erzeugen ein Magnetfeld. Dies kann mit Hilfe des Biot-Savart-Gesetzes [7] berechnet werden.

$$\vec{B}(\vec{r}) = \frac{\mu \cdot Q \cdot \vec{v} \times (\vec{r} - \vec{r_Q})}{4 \cdot \pi \cdot ||\vec{r} - \vec{r_Q}||^3} \tag{3.18}$$

Ein stromdurchflossener Leiter erwärmt sich gemäß des *Jouleschen Gesetzes* [7]. Die elektrische Energie W_{el} wird durch innere Reibung in Wärmeenergie W_{th} umgesetzt.

$$W_{th} = W_{el} = \int U \cdot I \cdot dt. \tag{3.19}$$

Diese *Eigenerwärmung* stellt bei integrierten Schaltungen einen sehr kritischen Systemparameter dar, da die entstehende Wärme nur schlecht abgeführt werden kann, gleichzeitig aber die genutzten Halbleiter empfindlich auf zu hohe Temperaturen reagieren. In CMOS-basierten Digitalschaltungen konkretisiert sich Gleichung 3.19 für die Verlustleistung P_V [7] bei der Betriebsspannung U, der umzuladenden Kapazität C und der Schaltfrequenz f zu

$$P_V = U^2 \cdot C \cdot f. \tag{3.20}$$

Verschiedene nichtleitende Kristallstrukturen ändern ihre Länge bei Anlegen einer Potentialdifferenz. Der *inverse Piezoeffekt* findet in Form von Piezo-Aktoren Anwendung. Die Verformung S berechnet sich anhand des angelegten elektrischen Feldes E und des als Tensor anzugebenden piezoelektrischen Verzerrungskoeffizienten d

$$S = d \cdot E. \tag{3.21}$$

Die *Elektrolumineszenz* bezeichnet das Aussenden von Strahlung aufgrund eines elektrischen Stromflusses [44]. Dies kann man z. B. bei LEDs beobachten. Die Wellenlänge der emittierten Strahlung ist dabei abhängig vom verwendeten Material. Mit Hilfe des Leistungswirkungsgrades η_P kann die bei einer bestimmten elektrischen Eingangsleistung P_{el} emittierte Strahlungsleistung Q_e bestimmt werden

$$Q_e = \eta_P \cdot P_{el}. \tag{3.22}$$

Die *Elektrolyse* trennt Stoffe aus einem gemeinsamen Verbund (z. B. Wasserstoff und Sauerstoff aus Wasser). Die dazu nötige Energie entspricht der Differenz der Enthalpien der Ausgangsstoffe und Produkte (siehe endotherme Reaktion).

Der *Peltier-Effekt* [45] beschreibt die Umkehrung des im Abschnitt Thermodynamik beschriebenen Seebeck-Effekts. Ein Stromfluss zwischen zwei Leitern bewirkt eine Änderung des Wärmetransports. Eine Anwendung dieses Effekts liegt in der Wärmeableitung von Bauelementen. In diesem Zusammenhang steht der *Thomson-Effekt* [45], der den geänderten Wärmetransport entlang eines stromdurchflossenen Leiters, in dem ein Temperaturgradient vorliegt, darstellt.

3.3.3. Magnetik

Beeinflussung innerhalb der Domäne

Die *magnetische Suszeptibilität* beschreibt die Materialeigenschaft, sich bei Anlegen eines externen Magnetfeldes zu magnetisieren. Dabei gibt die magnetische Suszeptibilität χ_M [46] die Änderung der Magnetisierung M bei einer sich ändernden Feldstärke H an

$$\chi_M = \frac{\delta M}{\delta H}. \tag{3.23}$$

Um magnetische Anordnungen mit denselben Lösungsmethoden wie elektrische Schaltungen berechnen zu können, entstand die Theorie der *Magnetkreise*. Die Einzelelemente wie Luftspalt oder Spule werden dabei durch Analogien wie magnetische Widerstände und magnetische Spannungsquellen ersetzt.

Beeinflussung anderer Domänen

Der *Faraday-Effekt* [47] beschreibt die Drehung der Polarisationsebene von polarisiertem Licht beim Durchgang durch ein transparentes Medium, an das ein Magnetfeld parallel zur Ausbreitungsrichtung der Lichtwelle angelegt wurde. Der Drehwinkel β, um den sich die Polarisationsebene dreht, berechnet sich wie folgt:

$$\beta = V \cdot d \cdot B. \tag{3.24}$$

d ist die Länge des Lichtweges durch die Substanz, B die magnetische Flussdichte und V die vom Medium und der Wellenlänge abhängige Verdet-Konstante.

Ein Magnet der Polfläche A übt bei einer magnetischen Flussdichte B auf seine Umgebung eine *Magnetkraft* F [7] von

$$F = \frac{B^2 \cdot A}{2\mu_0} \tag{3.25}$$

aus, wobei μ_0 der Induktionskonstanten entspricht.

3.3.4. Thermodynamik

Beeinflussung innerhalb der Domäne

Der Transport thermischer Energie erfolgt durch *Wärmeleitung, Wärmestrahlung* oder *Wärmeströmung* [39]. Zur Beschreibung dieser Effekte dient der Wärmestrom $\dot{Q}[W]$.

Die *Wärmeleitung* durch eine ebene Schicht ist abhängig von der wärmedurchströmten Fläche A und der Schichtdicke d sowie der Wärmeleitfähigkeit λ

$$\dot{Q} = \frac{\lambda}{d} \cdot A \cdot (T_1 - T_2), \tag{3.26}$$

wobei T_1 bzw. T_2 die Temperaturen an den Grenzflächen angeben.

Bei der *Wärmestrahlung* folgt der Wärmestrom zwischen zwei parallelen Flächen A der Temperaturen T_1 und T_2 mit dem Strahlungskoeffizienten C' der Gleichung 3.27

$$\dot{Q} = C' \cdot A \left[\left(\frac{T_1}{100} \right)^4 - \left(\frac{T_2}{100} \right)^4 \right]. \tag{3.27}$$

Bei *Wärmeströmung* (Konvektion) berechnet sich der Wärmestrom durch eine Fläche A unter Verwendung des Wärmeübergangskoeffizienten α ähnlich wie bei der Wärmeleitung zu

$$\dot{Q} = \alpha \cdot A \cdot (T_1 - T_2). \tag{3.28}$$

Beeinflussung anderer Domänen

Die mit Gleichung 3.20 berechnete Verlustleistung muss von den in integrierten Schaltungen vorhandenen Halbleiterstrukturen abgeführt werden, um eine starke Erhöhung der *Grenzflächentemperatur* $T_{Junction}$ des pn-Übergangs zu vermeiden. Die Überschreitung eines materialabhängigen Grenzwerts führt zu einer dauerhaften Veränderung der elektrischen Eigenschaften der Halbleiter und damit zur Zerstörung des Bauelements. Thermische Simulationen spielen deshalb bereits auf Systemebene eine große Rolle im Entwurf von Schaltungen.

Der *Seebeck-Effekt* [45] beschreibt das Entstehen einer Spannung an einem Temperaturgradienten an einem Leiter. Technisch kann dieser Effekt nur durch zwei unterschiedliche Leitermaterialien genutzt werden. Die Seebeck-Spannung ergibt sich aus

$$U_{Seebeck} = \alpha \cdot \Delta T \tag{3.29}$$

mit α als „Seebeck-Koeffizient“ und ΔT als Temperaturdifferenz.

Chemische Reaktionen mit *endothermem* Charakter laufen nur unter Energiezufuhr ab. Diese Energie wird meist in Form von Wärme zugeführt und entspricht der Reaktionsenthalpie (unter der Bedingung des konstanten Drucks während der chemischen Reaktion) [48]. Die Berechnung dieser Enthalpie wird im Abschnitt 3.3.5 behandelt.

Eine Temperaturänderung $\delta\theta$ bewirkt in Materialien eine *Längenänderung* δl gegenüber der Ausgangslänge l_0. Die Berechnung erfolgt mittels des Längenausdehungskoeffizienten α_{mech} [39]:

$$\delta l = l_0 \cdot \alpha_{mech} \cdot \delta\theta. \tag{3.30}$$

Eine Anwendung dieser Beziehung stellen Bimetallstreifen dar.

Der spezifische Widerstand von Stoffen ändert sich temperaturabhängig [7]. Bei den meisten Metallen nimmt der Widerstand mit steigender Temperatur aufgrund der zunehmenden Teilchenbewegung und der damit verbundenen erhöhten Anzahl der Kollisionen zu. Ein Maß für diese temperaturabhängigen Widerstände stellt der *Temperaturkoeffizient* α_{elec} dar, mit dessen Hilfe sich der spezifische Widerstand ρ_θ bei einer Temperatur θ gegenüber einer Referenztemperatur θ_{ref} mit dem spezifischen Widerstand ρ_{ref} wie folgt berechnet:

$$\rho_\theta = \rho_{ref} \left[1 + \alpha_{elec}(\theta - \theta_{ref})\right]. \tag{3.31}$$

Analog dazu ändert sich mit steigender Temperatur T auch die magnetische Stoffeigenschaft der Suszeptibilität χ_M gemäß des *Curie-Weiss-Gesetzes* [46]

$$\chi_M = \frac{C}{T - T_C}, \tag{3.32}$$

wobei die Curie-Temperatur T_C und die Curie-Konstante C Materialkonstanten darstellen.

3.3.5. Chemie

Beeinflussung innerhalb der Domäne

Chemische Reaktionen verändern die Eigenschaften der Eingangsstoffe und beeinflussen dabei ihre Umgebung. Es bilden sich neue Moleküle, wobei Energie aufgenommen (endotherm) oder abgegeben (exotherm) wird [48]. Die Energiedifferenz, auch Reaktionsenthalpie $\Delta_r H$ genannt, ist eine Differenz aus den Enthalpien H_i der Ausgangsstoffe (Produkte) und Endstoffe (Edukte) einer chemischen Reaktion:

$$\Delta_r H = \sum_{Produkte} H_i - \sum_{Edukte} H_i. \tag{3.33}$$

Beeinflussung anderer Domänen

Der Stoffumsatz bei chemischen Reaktionen führt meist auch zu einer Änderung des Volumens im Prozess und damit zur Verrichtung von *mechanischer Arbeit*. Einen allgegenwärtigen Vertreter dieses Effekts stellt der Verbrennungsmotor dar, der durch Oxidation eines Treibstoffs (beispielsweise Benzin oder Diesel) einen Kolben bewegt. Verläuft die Volumenänderung sehr schnell, spricht man von einer Explosion. Das Zusammenspiel von chemischen Reaktionen und mechanischen Effekten bildet ein eigenes Fachgebiet, die chemische Kinetik bzw. Reaktionskinetik.

In einer durch eine Membran geteilten Flüssigkeit baut sich bei Einbringen von gelösten Fremdmolekülen in eine Hälfte ein *Osmotischer Druck* auf, wenn die Membran zwar für die Flüssigkeit, aber nicht für die Fremdmoleküle durchlässig ist. Der Osmotische Druck Π kann in Analogie zum Gasdruck gesehen werden [49] und berechnet sich gemäß des van't Hoffschen Gesetzes:

$$\Pi = c \cdot R \cdot T \tag{3.34}$$

mit der Stoffmengenkonzentration c, der universellen Gaskonstante R und der Temperatur T.

Eine *Galvanische Zelle* wandelt chemische in elektrische Energie um. Sie besteht aus zwei Elektroden und einem Elektrolyt. Das Redoxpotential kann mit Hilfe der Nernstschen Gleichung nach [50] ermittelt werden:

$$E = E_0 + \frac{RT}{zF} lg \frac{c_{Ox}}{c_{Red}}. \tag{3.35}$$

R ist dabei die Gaskonstante, T die Temperatur, F die Faradaykonstante und z die Anzahl der Elektronen, die bei der Redoxgleichung beteiligt sind. Bei einer Raumtemperatur von $25\,°C$ erzeugt eine Kupferhalbzelle mit einer Konzentration von Kupfersulfit $CuSO_3$ von 2 mol/l ein Potential von 0,3488 V.

Chemische Reaktionen mit *exothermem* Charakter geben während des Reaktionsverlaufs Energie, meist in Form von Wärme, ab. Diese Energie entspricht der Reaktionsenthalpie (unter der Bedingung des konstanten Drucks während der chemischen Reaktion) [48].

Emittieren Atome oder Moleküle, die sich als direkte Folge einer stark exergonen Reaktion (starke Abnahme der freien Enthalpie G, Spontanreaktion) in einem elektronisch angeregten Zustand befinden, eine elektromagnetische Strahlung, so spricht man von *Chemolumineszenz* [51]. Dieser Effekt wird in Mikrosystemen in der biologischen und chemischen Analytik genutzt um bestimmte Substanzen nachzuweisen bzw. sichtbar zu machen. Im Makroskopischen tritt Chemolumineszenz z. B. in „Knicklichtern“ auf. Damit eine Strahlungsemission erfolgen kann, muss dem System ein exergoner Prozess zugrunde liegen, der eine wellenlängenabhängige Reaktionsenthalpie gemäß der Einsteinschen Gleichung

$$E = h \cdot \nu = \frac{h \cdot c}{\lambda} \tag{3.36}$$

liefert, wobei h das Plancksche Wirkungsquantum darstellt. Für blaues Licht der Wellenlänge 450 nm sind demzufolge ca. 254 kJ/Mol bzw. 2,75 eV erforderlich.

3.3.6. Strahlung

Beeinflussung innerhalb der Domäne

Für elektromagnetische Wellen, wie sie auch Licht darstellt, gelten die Maxwell-Gleichungen. Die Polarisation von Wellen lässt sich durch die beiden *Kerr-Effekte* beeinflussen, die eine Drehung der Polarisationsebene bei Reflektion an ferromagnetischen Oberflächen (magnetooptischer Kerr-Effekt [52]) bzw. die Änderung der optischen Eigenschaften eines Stoffes durch Anlegen eines elektrischen Feldes (elektrischer Kerr-Effekt [53]) bewirken. Das magnetische Analogon zum elektrischen Kerr-Effekt bildet der Faraday-Effekt.

Beeinflussung anderer Domänen

Strahlungsdruck [54] ist der Druck, der durch absorbierte, emittierte oder reflektierte elektromagnetische Strahlung auf eine Fläche wirkt. Im Wellenbereich kann dieser durch die Maxwellschen Gleichungen beschrieben werden, wogegen im Teilchenbereich eine einfachere Erklärung durch den Impuls eines Photons möglich ist. Ein Photon der Frequenz ν besitzt den Impuls $\vec{p}$ mit dem Betrag

$$|\vec{p}| = \frac{h\nu}{c^2}c = \frac{h\nu}{c} = \frac{h}{\lambda}. \tag{3.37}$$

Ein Photonenstrom mit der Bestrahlungsstärke E_e erzeugt bei der Lichtgeschwindigkeit c so den Druck

$$p = \frac{E_e}{c}. \tag{3.38}$$

Die *Photoakustik* basiert auf der Erkennung von akustischen Wellen, die durch die Absorption von gepulstem oder moduliertem monochromatischem Licht in einem Gas, einer Flüssigkeit oder einem festen Stoff erzeugt werden. Der Effekt wurde von Bell im Jahr 1880 entdeckt. Eine Anwendung dieses Effekts als Gassensor zeigt [55].

Eine auf einen Halbleiter einwirkende Strahlung generiert Elektronen-Loch-Paare [44]. Diese beispielsweise bei Solarzellen genutzten *photoelektrischen Effekte* generieren einen Stromfluss aus dem einstrahlenden Licht. Der generierte Fotostrom I_{ph} ist wellenlängenabhängig und berechnet sich bei vernachlässigbarer Reflexion gemäß [44]

$$I_{ph,\lambda} = q \cdot \frac{\Phi_{p0}(\lambda)}{A} \cdot e^{-\alpha(\lambda) \cdot d_{em}} \cdot \frac{\alpha(\lambda) \cdot L_n}{1 + \alpha(\lambda) \cdot L_n} \tag{3.39}$$

mit der Elementarladung q, dem spektralen Photonenstrom $\Phi_{p0}(\lambda)$, dem Absorptionskoeffizienten $\alpha(\lambda)$, der Diffusionslänge L_n und der Emitterdicke d_{em}.

Der *Fotomagnetische Effekt* [56] beschreibt den Einfluss von Strahlung (Licht) auf die Magnetisierung eines Stoffes. Dieser Effekt kann bei bestimmten Temperaturen zu dauerhaften Veränderungen der Stoffmagnetisierung führen.

Unter dem Begriff *Photochemie* versteht man chemische Reaktionen, die durch Einwirkung von Licht initiiert werden. Die Grundvoraussetzung hierfür ist eine Absorption des Lichtes durch das zu reagierende Molekül, d. h. das Absorptionsverhalten des Moleküls muss zu der Wellenlänge des verwendeten Lichtes passen.

3.4. Besonderheiten der elektrischen Domäne

Die elektrische Domäne unterteilt sich vom Entwurf her in den digitalen und den analogen Teilbereich. Im Analogbereich müssen alle Verbindungen sowohl zeit- als auch wertkontinuierlich betrachtet werden. Dies führt zu einem hohen Entwurfsaufwand und einer aufwändigen Fehlerabsicherung, so dass soweit als möglich digitale Hardware zum Einsatz kommt. Bei dieser liegen die Signalwerte diskretisiert, d. h. nur in bestimmten Stufen, vor. Fehlerhafte Werte dazwischen werden technologieabhängig den erlaubten Stufen zugeordnet, so dass sich gegenüber dem Analogbereich ein größerer Toleranzbereich ergibt.

Seit der Entwicklung des ersten Transistors hat der Digitalentwurf eine rasante Entwicklung genommen. Zahlreiche Werkzeuge und vorgefertigte Schaltkreise erleichtern das Umsetzen von Algorithmen in Hardware erheblich. Durch den Einsatz von Speicherelementen können komplizierte Berechnungsalgorithmen über mehrere Takte hinweg schrittweise abgearbeitet werden, die Tiefe der nötigen Kombinatorik sinkt und damit sowohl die Ausgangslast pro Gatter als auch die Länge des die Taktfrequenz bestimmenden längsten Pfades. Darüber hinaus ermöglichen Pipelining-Techniken eine überlappende Bearbeitung von Algorithmenteilen bzw. Befehlen.

Eine weitere Flexibilisierung setzte durch den Einsatz von rekonfigurierbaren Architekturen wie FPGA (Field Programmable Gate Array) sowie durch den Einsatz von Software (unter Verwendung von General-Purpose-Hardware wie Prozessoren) ein. Der Umfang der drei Teilbereiche Software und digitaler sowie analoger Hardware führte zu einer Separation des Entwurfsprozesses. Für jeden Bereich, im Analogbereich oft sogar domänenspezifisch, dominieren heute Spezialsimulatoren, die jedoch eine Gesamtsystemsimulation nur durch komplizierte Simulatorkopplungen zulassen. Daher entstanden mehrere Sprachen zur Modellierung dieser heterogenen Systeme, welche im Kapitel 4.2 kurz erläutert werden.

3.5. Anforderungen für die Modellierung des Gesamtsystems

3.5.1. Anforderungen aus der physikalischen Beschreibung

Die in den vorangehenden Abschnitten dargestellten physikalischen Effekte weisen in der Berechung verschiedene Komplexitäten auf. Dies reicht von einfachen Proportional-Zusammenhängen bis hin zu Differentialgleichungssystemen bei der Analyse von elektrischen und magnetischen Feldern. Damit ergeben sich folgende Anforderungen an Werkzeuge für die vollständige Systemmodellierung:

1. Darstellung von beliebigen arithmetischen zeitunabhängigen Zusammenhängen (e-Funktion, Potenzfunktionen, Matrixoperationen),
2. Darstellung von direkten Zeitabhängigkeiten,
3. Darstellung von Zeitabhängigkeiten bis zur zweiten Ableitung,
4. Nichtlineares Verhalten, Hysteresen,
5. Frequenzanalyse,
6. Partielle Differentialgleichungen/Ortsabhängigkeiten.

1. Darstellung von arithmetischen zeitunabhängigen Zusammenhängen

Diese Beschreibungsform eignet sich für alle Komponenten ohne speicherndes Verhalten wie z. B. ohmsche Widerstände oder idealisierte Dioden. Es dominieren hier lineare Zusammenhänge wie $U = I \cdot R$, aber auch e-Funktionen und Potenzfunktionen sind verbreitete Vertreter dieser Klasse. Bei räumlichen oder flächigen Ausdehnungen sind die zugehörigen Größen meist als Matrizen oder Vektoren zu berücksichtigen. Die mit zeitunabhängigen Zusammenhängen verbundenen Anforderungen an den Simulator sind vergleichweise gering, da nur der aktuelle Simulationszeitpunkt betrachtet werden muss.

2. Darstellung von direkten Zeitabhängigkeiten

Insbesondere Beschreibungen von aus der Umgebung einwirkenden sich ändernden Parametern erfordern die Möglichkeit der Angabe zeitlicher Abfolgen von Signalwerten. So können Sprungfunktionen der Form $\{x = 0 \text{ für } t < T_0; x = 1 \text{ für } t \geq T_0\}$ genutzt werden, um das dynamische Verhalten einer Anordnung zu untersuchen. Weitere übliche Testfunktionen sind periodische Anregungen (z. B. $x = sin(t)$) oder stückweise lineare Anregungen (PWL-Signale). Darüber hinaus kommen direkte zeitliche Abfolgen in Designs beispielsweise in Form von Taktgeneratornachbildungen mit periodischen Rechtecksignalen zum Einsatz.

3. Darstellung von Zeitabhängigkeiten bis zur zweiten Ableitung

Die Beschreibung von Systemteilen mit Hilfe von Differentialgleichungen beschränkt sich auf hoher Abstraktionsebene üblicherweise auf Terme bis zur zweiten zeitlichen Ableitung, wie in Gleichung 3.3 angegeben. Dies erfordert spezielle Lösungsmechanismen in den Simulatoren, die meist auf numerischen Methoden basieren. Enthält ein Simulator Konstrukte zur Modellierung dieses Gleichungstyps, können durch Analogiebeziehungen, wie in Tabelle 3.1 dargestellt, sowohl elektrische als auch nichtelektrische Zusammenhänge bearbeitet werden.

4. Nichtlineares Verhalten, Hysteresen

Einige Bereiche weisen nichtlineares Verhalten auf. Dies betrifft beispielsweise Reibungsvorgänge mit Übergängen zwischen Haft-, Gleit- und Rollreibung oder magnetische Hystereseeffekte. Eine Abbildung erfordert die Speicherung der vorhergehenden Zustände, um festzustellen, auf welchem Teil der Kennlinie sich das Systemteil befindet. Die Beschreibung dieser Vorgänge kann mit Hilfe digitaler Automaten erfolgen [57].

5. Frequenzanalyse

Neben der zeitlichen Betrachtung von Abläufen spielen Frequenzbetrachtungen eine große Rolle. So können durch Variation der Parameter aus Gleichung 3.3 Systemteile mit Tiefpass- oder Hochpasscharakter entstehen. Eine Analyse dieses Verhaltens im Frequenzbereich führt zu einer schnelleren und ressourcenschonenderen Charakterisierung des Systems, als es im Zeitbereich möglich wäre [58]. Insbesondere bei Systemteilen, die in Resonanz betrieben werden, empfiehlt sich die Berücksichtigung der Frequenzanalyse.

6. Partielle Differentialgleichungen/Ortsabhängigkeiten

Die höchsten Anforderungen an ein Modellierungstool stellen partielle Differentialgleichungen wie beispielsweise die Maxwell-Gleichungen 3.13 dar. Eine analytische Lösung ist nur für bestimmte Anordnungen schnell errechenbar, numerische Löser stoßen an ihre Grenzen. Derartige Zusammenhänge werden in der Regel mit Methoden aus dem Bereich der Finiten Elemente beschrieben und gelöst. Auf Systemebene versucht man, die Ortsabhängigkeiten bei feststehenden Geometrien aufzulösen und beispielsweise durch Polynomialfunktionen zu approximieren.

3.5.2. Weitere Anforderungen

Zur vollständigen Abbildung des Gesamtsystems und zur Unterstützung des Designprozesses sind außerdem notwendig:

1. Layoutinformationen,
2. Darstellung von speziellen mathematischen Algorithmen und Software,
3. Bewertungen von Realisierungsvarianten,
4. Verifikations- und Testverfahren.

1. Layoutinformationen

Neben dem Systemverhalten ist auch der physikalische Systemaufbau für den Erfolg eines Entwurfs entscheidend. Hier können durch ungünstige Anordnungen Störeffekte sowohl im elektromagnetischen als auch im klimatischen Bereich auftreten. Daher ist es notwendig, bereits auf Systemebene, besonders aber im Verlauf der Implementierung, Überlegungen zum Packaging und zum geometrischen Aufbau (z. B. Leiterplattendesign) anzustellen. Es empfiehlt sich, Modelle dieses Aufbaus in das Systemmodell zu integrieren um parasitäre Effekte und Wechselwirkungen frühzeitig zu erkennen und zu beheben.

2. Darstellung von mathematischen Algorithmen und Software

Die Nutzung der in diesem Kapitel dargestellten physikalischen Effekte erfordert in einigen Fällen komplexe Auswerteschaltungen. Die dazu nötigen Algorithmen (wie z. B. schnelle Fouriertransformationen - FFT) sowie deren Umsetzung in Hard- und Software müssen in die Systembeschreibung integriert werden. Dabei ist zu berücksichtigen, dass Hardware in der Regel wie physikalische Effekte parallel arbeitet, wogegen Software sequentiell ausgeführt wird.

3. Bewertung von Realisierungsvarianten

Für eine Spezifikation existieren meist mehrere Realisierungsvarianten. So kann ein Algorithmus in Hard- oder Software umgesetzt werden, oder für eine Messung sind verschiedene Messverfahren (beispielsweise kapazitiv oder magnetisch) bekannt. In diesen Fällen ist eine Abwägung der Varianten erforderlich, um mit minimiertem Aufwand ein möglichst leistungsfähiges und robustes System zu erhalten. Dies stellt wiederum Anforderungen an die Vergleichbarkeit der Implementierungen und damit an die Aufstellung geeigneter Vergleichsmetriken.

4. Verifikations- und Testverfahren

Die steigende Systemkomplexität erschwert Systemverifikation und Test erheblich. Ein vom Test losgelöster Entwurf führt zu hohen Testkosten bzw. verhindert sogar die Testbarkeit des Systems. Daher muss der Systemtest von Anfang an in den Designprozess einbezogen werden. Darüber hinaus erfordert jeder Entwurfsschritt die Verifikation des neuen Systemverhaltens gegenüber der Spezifikation bzw. dem Ausgangsverhalten des Designschritts. Je nach Abstraktionsebene kommen hier unterschiedliche Verifikationsverfahren zum Einsatz. Die Ergebnisse von Verifikation und Test sind nachvollziehbar zu dokumentieren, um beispielsweise bestimmten Zertifizierungen (ISO 9001) zu genügen.

4. Aktueller Stand der Wissenschaft

4.1. Abstrakte analoge Modelle

Die steigende Systemkomplexität stellt die Entwerfer und Testingenieure vor das Problem, in derselben Zeit immer mehr Funktionen implementieren bzw. verifizieren zu müssen. Das Ausweichen auf hohe Abstraktionsebenen ermöglicht einen Ausweg aus dieser Problematik. Im digitalen Bereich stehen auf endlichen Zustandsautomaten (Finite State Machine - FSM) basierende Modelchecker zur Prüfung der Funktionalität auf Systemebene zur Verfügung. Die Übertragung dieser Ansätze auf analoge Problemstellungen ist durch die Nutzung hybrider Automaten möglich.

Hybride Automaten basieren auf endlichen Zustandsautomaten, die man in mehrere Varianten unterteilt, am weitesten verbreitet sind dabei Moore- und Mealy-Automaten. Diese ereignisbasierten Automatentypen ermöglichen jedoch keine Beschreibung zeitkontinuierlichen Verhaltens. Das führte zur Entwicklung von „Timed Automata“ [59], welche die FSM um Uhren, also von Null startende Zähler, erweitern. In der Urversion unterstützen diese Modelle keine Hierachieebenen, dies implementierte Beyer in den sogenannten „Cottbus Timed Automata“ [60].

Nur wenige mikroelektromechanische Systeme arbeiten rein analog, die meisten MEMS wandeln die analogen Messwerte in digitale Signale um und verarbeiten diese weiter. Diese Heterogenität wird von reinen „Timed Automata“ nicht unterstützt, daher kombinierten Alur und Henzinger [61] die zeitkontinuierlichen Automaten mit ihren diskreten Ursprüngen zu „Hybriden Automaten“.

Ein hybrider Automat H besteht aus mehreren Komponenten [61]:

- einer Menge $X = \{x_1, ..., x_n\}$ von Gleitkomma-Variablen,
  - n wird als Dimension von H bezeichnet.
  - $\dot{X}$ stellt die ersten Ableitungen der Mengenelemente von X dar,
  - X' repräsentiert die Variablenwerte am Ende eines Zustandsübergangs;
- einem Graphen $(V = (v, s), E)$, bestehend aus:
  - den Orten v,
  - der Belegung s und
  - den Übergängen E;

- den Invarianten inv,
- den Initialzuständen $init$,
- den ortsinternen Transferfunktionen $flow$,
- den Sprungbedingungen $jump$ für die Zustandsübergänge E und
- einer Menge von Ereignissen $events$ für die Zustandsübergänge.

Bei der Abarbeitung des hybriden Automaten kommen sowohl kontinuierliche ($flow$) als auch diskrete ($jump$) Variablenänderungen zur Ausführung. Die Korrektheit des hybriden Automaten kann unter anderem durch Angabe von Invarianten (inv) geprüft werden, die im jeweiligen Zustand zu jedem Zeitpunkt erfüllt sein müssen.

Das folgende Beispiel einer Heizungssteuerung aus [62] soll die o. g. Begriffe verdeutlichen. Die Temperatur T des Raumes wird durch einen Sensor gemessen und die Heizung je nach Bedarf zu- oder abgeschaltet. Bei ausgeschalteter Heizung folge T der Gleichung

$$T = T_0 \cdot e^{-at}, \tag{4.1}$$

wobei T_0 die Anfangstemperatur, t die Zeit sowie a eine raumspezifische Konstante darstellen. Arbeitet die Heizung, berechnet sich die Raumtemperatur nach der Gleichung 4.2

$$T = T_0 \cdot e^{-at} + h \cdot \left(1 - e^{-at}\right). \tag{4.2}$$

In dieser Gleichung gibt h den Wert der Heizleistung an. Die Raumtemperatur soll sich im Bereich $T_{min} < T < T_{max}$ bewegen. Bild 4.1 zeigt den resultierenden hybriden Automat des Heizungssystems. Dabei ist im Ort l_0 die Heizung ausgeschaltet, während sie sich im Ort l_1 in Betrieb befindet.

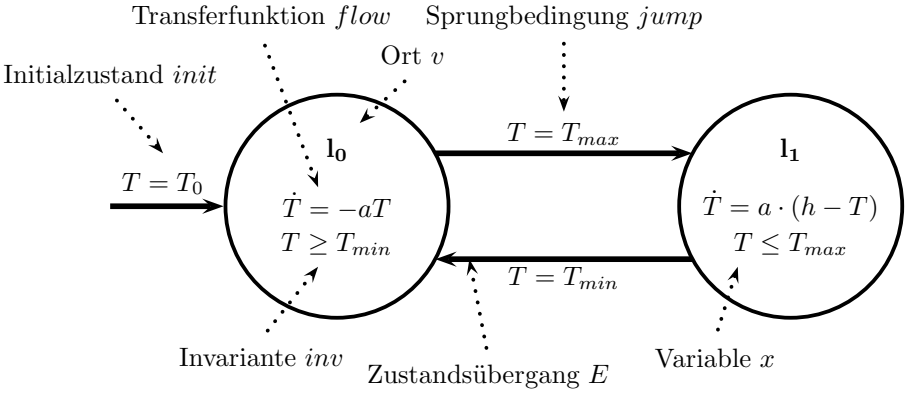


Bild 4.1.: *Hybrider Automat der Heizungssteuerung [62]*

Die Modellierung von Mikrosystemen mit dieser Automatenart ist Gegenstand des Kapitels 9.3.1.

Die Analyse von hybriden Automaten ist Ziel verschiedener Tools. Einen der ersten Modelchecker für hybride Automaten stellte Henzinger mit HyTech [63] zur Verfügung. Dieser beschränkt sich auf lineare hybride Automaten, kann also nur einfache Formen mit konstanter Ableitung bearbeiten. Als Nachfolger nutzt HyperTech [64] die Methoden der Intervallarithmetik und kann damit auch komplexere Systeme behandeln. Einen anderen Ansatz nutzt Tiwari [65] mit dem Tool HybridSAL, welcher hybride Automaten in diskrete FSM überführt und mit Hilfe des Digitalmodelcheckers SAL prüft. Beide Werkzeuge sind Gegenstand des Kapitels 6.

Des weiteren befassen sich die Werkzeuge HybridCC (NASA, [66]), SHIFT (Berkeley, [67]) und STeP (Stanford, [68]) mit hybriden Automaten. Das Werkzeug HyCharts [69] nutzt einen ähnlichen, graphenbasierten Ansatz, der theoretisch nach SystemC-AMS exportierbar ist [70], dessen Umsetzung aber noch nicht publiziert wurde. Darüber hinaus existieren viele weitere Ansätze wie z. B. zur Stabilitätsanalyse [71] im Rahmen des AVACS-Projekts.

Allen Werkzeugen ist gemein, dass die derzeit verfügbaren Rechner für eine umfassende Analyse komplexer heterogener Systeme nicht genug Rechenleistung bereitstellen, da der Zustandsraum durch die Kontinuität des Wertebereichs stark anwächst. Aus diesem Grund müssen diese Systeme zur Analyse partitioniert und eine Gesamtsystemprüfung nach wie vor manuell durchgeführt werden.

Neben den hybriden Automaten existieren weitere Beschreibungsformen für abstrakte heterogene Systeme. Diese sollen hier nur genannt werden, für eine nähere Erläuterung sei auf die Literaturangaben verwiesen. So definieren Steinhorst et al. [72] eine Sprache für analoge Spezifikationen. Diese bietet noch keine Verbindung zu gängigen Digitalmethoden. Die temporale Sprache CTL erhielt eine analoge Erweiterung CTL-AT [28], die auf einfache analoge Konstrukte beschränkt ist. Ebenso wurde die Sprache PSL (Property Specification Language) im Rahmen des PROSYD-Projekts um analoge Konstrukte erweitert [73]. Diese Erweiterung erlaubt nur signalfluss-basierte Properties.

4.2. Beschreibungsmöglichkeiten heterogener Systeme

Heterogene Systeme können mehrere analoge Domänen (siehe Kapitel 3) sowie digitale Hardware und Software umfassen. Für jede dieser Domänen und Bereiche existieren eigene Spezialsimulatoren, wie z. B. Spice für elektrische analoge Systeme mit diskreten Elementen oder VHDL-/Verilog-Simulatoren für digitale Sachverhalte. Im Bereich der mikroelektro-mechanischen Systeme werden beim Entwurf der Mikrostrukturen hauptsächlich FEM-Simulatoren wie z. B. ANSYS verwendet. Die Methode der Finiten Elemente (FEM) diskretisiert die Oberflächen der Strukturen und führt für die entstandenen Punkte Berechnungen durch. Bei entsprechend hoher Punktdichte kann das Gesamtverhalten einer Struktur hinreichend genau abgebildet werden, was jedoch zu langen Rechenzeiten führt, die sich durchaus im Bereich von Tagen bewegen können. Mittels Ordnungsreduktion kann der Zeitbedarf reduziert werden [74]. Die entstehenden Makromodelle lassen sich z. B. in VHDL-AMS [75] simulieren.

Zur Simulation eines mehrere Domänen umfassenden Systems unter Nutzung der Spezialsimulatoren sind aufwändige Simulatorkopplungen nötig. Besonders beim Übergang vom zeitkontinuierlichen analogen Bereich zum zeitdiskreten Digitalbereich stellt sich das Problem der Synchronisation der Zeitachsen. Daher befinden sich mehrere domänenübergreifende Beschreibungssprachen in der Entwicklung. Die sogenannten AMS-Sprachen (Analog and Mixed-Signal) vereinen digitale und analoge Beschreibungsformen auf einer gemeinsamen Syntax- und Simulationsgrundlage. Zu dieser Sprachgruppe zählen z. B. Verilog-AMS [76], VHDL-AMS [77], SystemC-AMS [78] und in begrenztem Umfang Modelica [79].

Verilog-AMS erweitert die digitale Beschreibungssprache Verilog [5] um analoge und Mixed-Signal Funktionen. Die Sprache bietet Modellierungsmöglichkeiten im Zeit- und Frequenzbereich. Ihre Verbreitung in Europa war bisher verglichen mit VHDL geringer, sie gewinnt jedoch durch die Globalisierung der Unternehmen und den weltweiten Modellaustausch zunehmend an Einfluss auch im nichtamerikanischen Raum. Die Sprache soll hier nicht weiter erläutert werden, eine gute Einführung bieten aber Kundert und Zinke in [80].

Eine weitere verbreitete Entwurfssprache ist Matlab mit seiner Erweiterung Simulink. Zu diesem Werkzeug existieren unzählige Zusatzbibliotheken, welche auf algorithmischer Ebene eine effiziente Untersuchung von Lösungsmöglichkeiten bieten. Jedoch unterstützt Matlab/Simulink lediglich gerichtete Datenflussnetze ohne direkte Umsetzung von konservativen Netzen. Damit schränkt sich die Möglichkeit der exakten Nachbildung physikalischer Effekte auf Verhaltensebene ein, eine stark abstrahierte Abbildung des Systemverhaltens von MEMS ist aber möglich [81].

Die folgenden Abschnitte geben einen Überblick über die drei AMS-Beschreibungssprachen Modelica, VHDL-AMS und SystemC-AMS auf dem Stand von 2008. Als gemeinsames Beispiel soll die Modellierung eines fallengelassenen Balles („bouncing ball example“) die Unterschiede der Sprachen verdeutlichen. Ein Vergleich von Modelica und VHDL-AMS (Stand 2001) anhand weiterer Beispiele kann in [82] nachgelesen werden.

4.2.1. Modelica

Die Modelica Association entwickelt seit Ende der neunziger Jahre des vergangenen Jahrhunderts die Sprache Modelica. Federführend in diesem Konsortium sind die Deutsche Luft- und Raumfahrtgesellschaft in Oberpfaffenhofen, die Universität Linköpping, Schweden und die schwedische Firma Dynasim aus Lund. Neben dem frei verfügbaren Tool OpenModelica [83] der Universität Linköpping sind mehrere kommerzielle Tools (Dynasim, MathModelica) verfügbar. Von einem Verbund der Fraunhofergesellschaften wird mit Mosilab [84] ein weiteres kommerzielles Tool entwickelt.

Modelica ist eine objektorientierte Sprache zur Modellierung physikalischer Zusammenhänge. Sie ist im Gegensatz zu den drei anderen angesprochenen Sprachen nicht als

Erweiterung einer digitalen Beschreibungssprache (Verilog, VHDL, SystemC) entstanden, sondern basiert auf C++ und Mathematica. Eine Einführung in Modelica gibt Fritzson [85].

Ein Modelica-Modell besteht aus einer Beschreibung der Schnittstellen und Deklaration der Variablen sowie einem Teil zur Verhaltensbeschreibung. Das Schlüsselwort *model* zeigt den Beginn einer Modelldefinition an, auf das die Ein- und Ausgangsdefinitionen sowie die Parameterangaben folgen. In diesem Teil des Modells werden auch Unterkomponenten deklariert. Modelica unterscheidet zwischen konservativen (*Pin*) und nichtkonservativen (*Input, Output*) Anschlüssen. Mit dem Schlüsselwort *equation* wird der Beginn des Bereichs der Verhaltensbeschreibung gekennzeichnet, hier erfolgt entweder die Angabe von Gleichungssystemen zur Modellierung des Klemmenverhaltens oder die Verbindung der Unterkomponenten.

Als objektorientierte Sprache bietet Modelica die Vorzüge der Vererbung auch für den analogen Bereich. Ähnlich wie in C++ existieren Zugriffsbeschränkungen wie *protected* oder *public*. Parameter und Variablen können durch *replacable* und *redeclare* bei der Instanziierung überschrieben werden. Für häufig genutzte Elemente bietet die Sprache vordefinierte Elemente im elektrischen und nichtelektrischen Bereich wie Spulen und Widerstände oder träge Massen. Die Modellerstellung kann entweder manuell durch Eingabe des Quellcodes oder durch Nutzung von grafischen Editoren, sogenannten Schematics, erfolgen. Die Sprache erlaubt im Ursprung nur Zeitbereichssimulationen, einen Ansatz zur Frequenzbereichssimulation zeigen Abel und Nähring [86].

Zur Evaluierung wurde das kostenfrei verfügbare OpenModelica der Version 1.4.1 genutzt. Dies verwendet zur Eingabe von Problemstellungen die aus Mathematica bekannten „Notebooks“. Einen kurzen praktischen Einblick in Modelica gibt das folgende Beispiel des aufspringenden Balles.

Beispiel Bouncing Ball

Im Beispiel „Bouncing Ball“ wird ein elastischer Ball in 10 m Höhe fallengelassen. Die einwirkende Gravitationskraft führt zur Beschleunigung des Objekts zum Erdboden hin, gleichzeitig wirkt die Reibungskraft dieser Bewegung entgegen. Nach mehreren elastischen Stößen auf dem Erdboden kommt diese gedämpfte Schwingung nach ca. 20 s zur Ruhe. Listing 4.1 zeigt den Quellcode des Beispiels „fallengelassener Ball“ für Modelica, Bild 4.2 das Ergebnis der Simulation als Funktionsdarstellung der Ballhöhe über der Zeit.

Die Sprache soll im folgenden nicht weiter berücksichtigt werden, da den Möglichkeiten der Modellierung von digitaler Hardware und Software in der genutzten Version enge Grenzen gesetzt sind. Für Mikrosysteme im Sinne dieser Arbeit bieten sich die beiden Sprachen VHDL-AMS und SystemC-AMS eher für die ganzheitliche Modellierung an.

```
model BouncingBall
  parameter Real airres=0.1 "air resistance";
  parameter Real g=9.81 "gravity acceleration";
  Real h(start=10) "height of ball";
  Real v "velocity of ball";
  Boolean flying(start=true) "true, if ball is flying";
  Boolean impact;
  Real v_new;

equation
  impact = h <= 0.0;
  der(v) = if flying then (if v>0 then -g - v*v*airres
           else -g + v*v*airres)  else 0;
  der(h) = v;

  when {h <= 0.0 and v <= 0.0,impact} then
    v_new = if edge(impact) then -pre(v) else 0;
    flying = v_new > 0;
    reinit(v, v_new);
  end when;
end BouncingBall;
```

Listing 4.1: *Modelica-Code für „bouncing ball“*

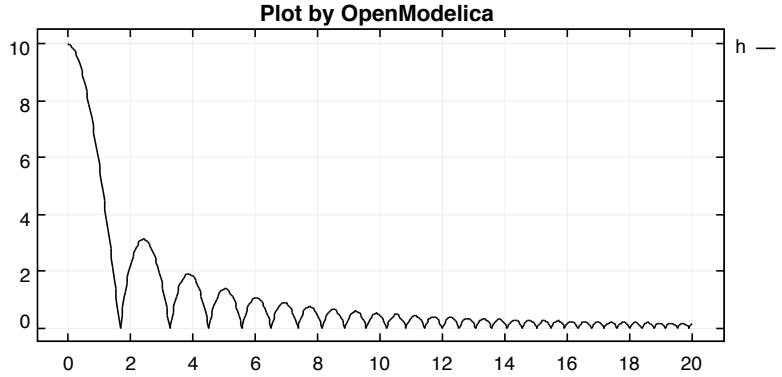


Bild 4.2.: *Modelica-Simulationsergebnis des aufspringenden Balls*

4.2.2. VHDL-AMS

Die Sprache VHDL-AMS basiert auf der für den Digitalbereich entwickelten Hardwarebeschreibungssprache VHDL [4]. Seit etwa 1993 entstand mit VHDL-AMS bzw. dem Vorläufer HDL-A die Möglichkeit der Beschreibung auch nichtdigitaler Sachverhalte mit der im Mikroelektronikentwurf weit verbreiteten Syntax von VHDL. Zwei Sprachvarianten („Jade“ und „Opal“) dienten als Grundlage für die im Jahr 1999 erfolgte Standardisierung durch die IEEE als Standard Nummer 1076.1 [77].

VHDL-AMS basiert auf Differentialgleichungssystemen, wobei jeder analytisch beschreibbare Zusammenhang zwischen Fluss- und Differenzgrößen erlaubt ist. Es sind sowohl Zeitbereichs- als auch Frequenzbereichssimulationen im Sprachstandard verankert. Eine ausführliche und praxisnahe Einführung in die Sprache gibt Ashenden [87], daher werden hier nur kurz die wichtigsten Modellierungsgrundlagen angesprochen.

Ein VHDL-AMS-Modell besteht aus einer Schnittstellenbeschreibung (Entity) und einer Beschreibung des Modellverhaltens bzw. seines inneren Aufbaus (Architecture). Zu einer Entity können mehrere Architectures, also Implementierungsvarianten, existieren. Die Auswahl der zu nutzenden Variante erfolgt durch eine Configuration. Vereinfacht ausgedrückt entspricht eine Entity einem Gehäuse, die Configuration ordnet dem Gehäuse einen konkreten Chip (Architecture) zu und fügt diesen in einen Sockel auf der Leiterplatte (Component) ein. Für mehrfach genutzte Designelemente wie z. B. Typdefinitionen, Prozeduren und Funktionen empfiehlt sich eine Auslagerung in Packages, was den Zugriff von mehreren Komponentenbeschreibungen aus ermöglicht.

Für die Schnittstellen zwischen Modulen stellt VHDL-AMS mehrere Verbindungsmöglichkeiten bereit. Auf Verhaltensebene bietet VHDL-AMS Terminals an, die als konservative Knoten im Netzwerk fungieren. In dieser Beschreibungsform gelten die Kirchhoffschen Sätze, zwischen Terminals sind Fluss- und Differenzgrößen definierbar. Verbindungen auf funktionaler Ebene basieren in VHDL-AMS auf sogenannten Interface Quantities. Hier kommunizieren die Teilsysteme über gerichtete Verbindungen miteinander, die Kirchhoffschen Sätze gelten auf dieser Ebene nicht.

Die Syntax von VHDL-AMS orientiert sich stark am digitalen Ursprung VHDL. So leitet das Schlüsselwort *ENTITY* eine Modulbeschreibung ein, die Parameter (*GENERIC*) und Schnittstellen (*PORT*) enthalten kann. VHDL-AMS unterscheidet drei Schnittstellentypen: *SIGNAL* als digitale Verbindung, *QUANTITY* als gerichtete Analogverbindung und *TERMINAL* als konservativen Knoten. Das eigentliche Verhalten einer Komponente beschreibt der Abschnitt *ARCHITECTURE*, der neben strukturalen Zusammenschaltungen von Teilsystemen auch Verhaltensbeschreibungen enthalten darf. Implizit angebbare Differentialgleichungen bilden das Verhalten der Komponente auf der jeweiligen Abstraktionsebene in Form von *simultaneous statements* mit dem Verbindungsoperator == nach. Rechte und linke Seite der Gleichung sind dabei gleichberechtigt, es erfolgt also keine Zuweisung wie bei einer Programmiersprache. Die Synchronisation der digitalen und analogen Zeitachse erfordert besondere Beachtung. VHDL-AMS erzwingt die Synchronisation durch Angabe des *BREAK*-statements bzw. generiert digitale Ereignisse im Analogbereich durch Abtastung oder Schwellwertschalter (*'ABOVE*).

Beispiel Bouncing Ball

Listing 4.2 zeigt die Verhaltensbeschreibung des Beispiels „fallengelassener Ball“ für VHDL-AMS, in Bild 4.3 ist das Simulationsergebnis dargestellt.

```
ARCHITECTURE behav OF bouncer IS
  quantity acc : real := 0.0;
  quantity v : real := 0.0;
  quantity s : real :=10.0;
  constant G : real := 9.81;
  constant luftwid : real := 0.1;
  SIGNAL vsave : real;

BEGIN
  v == s'dot; acc == v'dot;

  break WHEN not s'above(0.0);

  if now<=0.0 use s == 10.0;
  elsif NOT s'above(0.0) USE
    v == - vchange;
  elsif v> 0.0 use
    acc == -G -v**2 * luftwid;
  else
    acc == -G + v**2 * luftwid;
  end use;

  -- save velocity
  process(s'above(0.0))
  begin
    if NOT s'above(0.0) then
      vsave<=v;
    end if;
  end process;
END;
```

Listing 4.2: *VHDL-AMS-Verhaltensbeschreibung für „bouncing ball“*

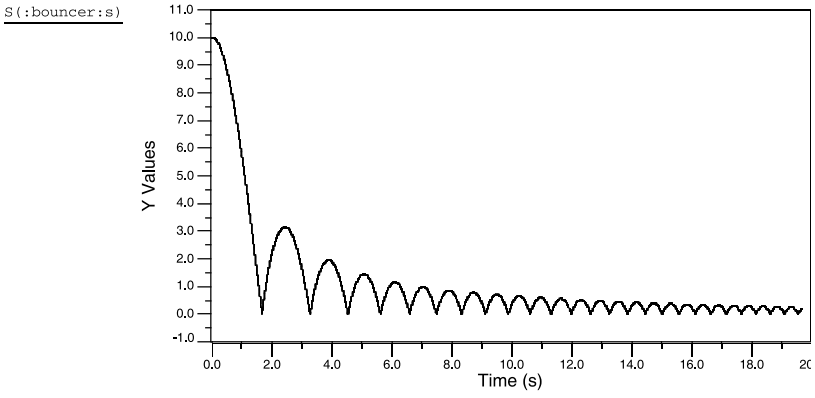


Bild 4.3.: *Ergebnis der VHDL-AMS-Simulation des aufspringenden Balls*

4.2.3. SystemC-AMS

Im Gegensatz zu VHDL-AMS befindet sich SystemC-AMS noch am Anfang der Standardisierungsphase durch die IEEE. Es existieren im Vergleich zu VHDL-AMS erst wenige Veröffentlichungen zur Sprache, daher soll ihr hier mehr Raum eingeräumt werden, um die speziellen Konstrukten der Sprache zu erläutern.

Die C++-Bibliothek SystemC [88] und ihre Erweiterung SystemC-AMS [78] ermöglichen die Beschreibung von Hardware mit Standard-C++-Compilern. Spezielle Konstrukte erlauben die Modellierung von Parallelität, Reaktivität und Zeitverhalten in der sequentiellen Abarbeitung von C++-Softwarecode. Jegliche Hardware wird in SystemC(-AMS)-Module gekapselt. Ein Scheduler überwacht die parallele Ausführung der Modulfunktionen. Die Zeitweiterschaltung erfolgt im digitalen SystemC ereignisgesteuert. Im analogen SystemC-AMS wird hingegen mit einer möglichst kleinen Zeitschrittlänge ein kontinuierlicher Zeitverlauf nachgebildet. Diese Zeitschrittlänge hat jedoch signifikanten Einfluss auf die Simulationsgeschwindigkeit. Die folgenden Ausführungen beziehen sich auf die SystemC-AMS-Version 0.15 RC1, welche in dieser Arbeit verwendet wird. Anfang 2009 erschien ein erstes Language Reference Manual (LRM) für die IEEE-Standardisierung. Die darin vorgeschlagenen Konstrukte für die Version 1.0 unterscheiden sich syntaktisch von denen der Version 0.15 RC1, jedoch stand für diese Arbeit noch kein Simulator der Version 1.0 zur Verfügung.

Modellaufbau in SystemC-AMS

Eine SystemC-AMS-Beschreibung einer Komponente besteht aus einem oder mehreren Modulen. In einem Modul können Ein- und Ausgänge, Parameter, Membervariablen und -funktionen sowie Enumerationen definiert sein. Außerdem sind hierarchische Anordnungen erlaubt, d. h. innerhalb eines Moduls werden Submodule instanziiert und mittels Signalen oder elektrischen Leitungen miteinander sowie zur Umgebung hin verbunden.

In **digitalen Modulen** wird die Funktionalität mit Methoden und Threads abgebildet. Der digitale Simulationskern arbeitet ereignisgesteuert, d. h. die Methoden werden nur bei Signaländerungen der als sensitiv gekennzeichneten Eingänge abgearbeitet. Auf RT-Ebene versteht man darunter im sequentiellen Fall Reset- und Taktsignale, bei kombinatorischen Sachverhalten sind die Methoden auf alle Eingänge zu sensitivieren.

Bei der Abarbeitung **analoger Module** wird durch eine möglichst kleine Simulatorschrittweite ein zeitkontinuierliches Verhalten nachgebildet. Beschreibungen im Frequenzbereich sind ebenfalls möglich. Zwischen Modulen existieren gerichtete (Datenflussnetze) und ungerichtete (lineare konservative Netze) Verbindungen. Für verschiedene Phasen der Simulation von Datenflusskomponenten bietet SystemC-AMS vordeklarierte Funktionen, die der Simulator automatisch aufruft. Tabelle 4.1 listet diese kurz auf.

Funktionsname	Inhalt
void attributes()	Setzen von Attributen wie z. B. Porteigenschaften
void init()	Initialisieren von Ports und Variablen
void sig_proc()	Beschreibung des Zeitverhaltens (Abarbeitung zu jedem Zeitschritt)
void ac_sig_proc()	Beschreibung des Frequenzverhaltens
void post_proc()	Simulationsabschluss (Schließen von Dateien, Auswertefunktionen)

Tabelle 4.1.: *vordefinierte Funktionen in SystemC-AMS v0.15*

Neben SystemC-Funktionen umfasst SystemC-AMS v0.15 eine Reihe eigener, vordefinierter Bauelemente für lineare Netze:

- konstante und gesteuerte Widerstände, Spulen und Kapazitäten,
- konstante, gesteuerte und sinusförmige Strom- und Spannungsquellen,
- Gyrator, idealer Transformator und Nullor/Norator,
- Wandlerkomponenten von und zu Datenflusssignalen.

Weiterhin steht im Datenflussbereich eine Laplace-Transferfunktion für Zeit- und Frequenzverhalten zur Verfügung, Entsprechungen für Pol-Nullstellen- und State-Space-Darstellungen sind geplant. Die FhG IIS/EAS Dresden entwickelte einen nichtlinearen Löser [89].

Für SystemC-AMS der genutzten Version ergibt sich ein Schichtenaufbau wie in Bild 4.4 dargestellt. Die AMS-Erweiterung setzt auf dem digitalen SystemC-Kernel auf. Eine Synchronisationsschicht sorgt dabei für die notwendige Kopplung der Zeitachsen. Sie dient als Ansatzpunkt für die verschiedenen Löser der Sprache wie z. B. Laplace-Löser oder Matrixlöser. Diese Löser stellen die entsprechenden Modellierungsfunktionen für den Nutzer bereit.

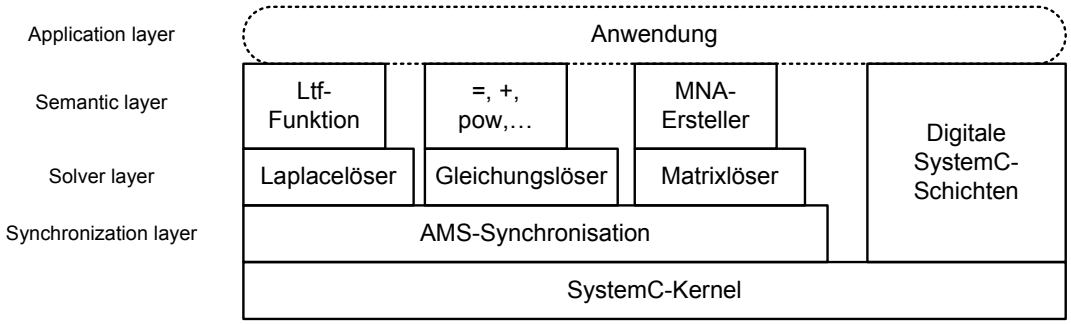


Bild 4.4.: *Schichtenaufbau von SystemC-AMS v0.15 (nach [78])*

Ablauf einer Simulation

Nach der Compilierung startet die Simulation mit einer Elaborationsphase. Diese dient der Erzeugung interner Datenstrukturen und zum Setzen der in attributes() angegebenen Parameter. Darüber hinaus sind die nötigen Zeitschrittweiten festzulegen sowie eine Berechenbarkeitsanalyse durchzuführen. An die Elaboration schließt sich die eigentliche Simulation an. Den ersten Simulationsschritt bildet die Initialisierung durch Aufruf von init(). Anschließend sind die sig_proc() bzw. ac_sig_proc()-Funktionen aller beteiligten Module aufzurufen, sobald diese genug Eingangswerte gesammelt haben. Der Ruf von sig_proc() bzw. ac_sig_proc() ist zu wiederholen, bis die vorgegebene Simulationsdauer erreicht ist. Den Abschluss bildet der Ruf von post_proc(), der zur Durchführung von Operationen auf die gesammelten Simulationsdaten genutzt werden kann.

Einführung in die Syntax

Die Komponentenbeschreibung in SystemC-AMS erfolgt auf Basis von C++-Klassen. Vordefinierte Makros unterstützen dabei den Entwerfer. Wie in C/C++ üblich, ist auch SystemC-AMS case-sensitiv, es unterscheidet also im Gegensatz zu VHDL/VHDL-AMS zwischen Groß- und Kleinschreibung.

Eine Komponente wird üblicherweise als *SC_MODULE* für digitale und hierarchische sowie als *SCA_SDF_MODULE* für analoge Sachverhalte beschrieben. Daneben ist die direkte Definition der Komponente in Form einer eigenen Klasse als Ableitung der Basisklasse *sc_module* möglich. Ein Analogmodul muss in diesem Falle zusätzlich von der Klasse *sca_sdf_view* abgeleitet werden. Als Schnittstellen stehen gerichtete und ungerichtete Anschlüsse zur Verfügung:

- gerichtete Digitalanschlüsse *sc_in* $<Typ>$, *sc_inout* $<Typ>$ und *sc_out* $<Typ>$
- gerichtete Analoganschlüsse *sca_sdf_in* $<Typ>$ und *sca_sdf_out* $<Typ>$
- ungerichtete Verbindungen (konservative Klemmen) *sca_elec_port*

Die Klassendefinition enthält darüber hinaus noch die Deklaration der Komponentenfunktionen und strukturaler Elemente. Für Analogmodule auf Datenflussebene existieren zusätzlich die fünf in Tabelle 4.1 vordefinierten Funktionen. Der Modulkonstruktor (in Form der Makros *SC_CTOR* bzw. *SCA_CTOR* oder als Direktdefinition mit der Möglichkeit der Parameterübergabe) ermöglicht die Initialisierung von Variablen und die Anforderung von Simulationsspeicher.

Die Umsetzung der bisher angesprochenen Sprachelemente für ein konkretes Modell werden üblicherweise in ein Headerfile (.h) ausgelagert, da sich die Modulschnittstellen selten ändern. Die Implementierung der bereits angesprochenen Komponentenfunktionen befindet sich hingegen in einer .cpp-Datei. Diese entspricht dem Verhalten der Komponente im Zeit- bzw. Frequenzbereich.

Das digitale SystemC bietet darüber hinaus noch zahlreiche zusätzliche Konstrukte zur Modellierung auf hohen Abstraktionsebenen, die im Language Reference Manual beschrieben sind. Da sie im Rahmen dieser Arbeit nicht benutzt werden, entfällt eine nähere Erläuterung dieser Syntaxelemente. Die folgenden Abschnitte geben Beispiele für die Anwendung der Sprachelemente von SystemC-AMS.

Beispiel Bouncing Ball

Listing 4.3 zeigt den Quellcode der Verhaltensbeschreibung des Beispiels „fallengelassener Ball“ für SystemC-AMS; in Listing 4.4 ist das zugehörige Hauptprogramm mit der Einstellung der Simulationsparameter sowie der zu beobachtenden Signale zu sehen. Bild 4.5 stellt das Simulationsergebnis dar.

```
SCA_SDF_MODULE(bouncer){

  // outputs for tracing
  sca_sdf_out<double> acceleration, velocity, position;

  // internal variables
  double height, acc_act, vel_act;
  static const double acc_g = 9.81;
  static const double air_res = 0.1;

  // initialize variables
  void init(){
    height = 10.0; // start at 10m above ground
    vel_act = 0.0; // no starting velocity
    acc_act = 0.0;}

  // describe behaviour
  void sig_proc(){
    // conservation of linear momentum
    if (height <= 0.0)
      vel_act = -vel_act;
    else // integrate velocity
      vel_act = vel_act+ acc_act*velocity.get_T().to_seconds();

    height = height + vel_act*velocity.get_T().to_seconds();

    // calculate acceleration
    if (vel_act < 0)
        acc_act = - acc_g + vel_act*vel_act* air_res;
    else
        acc_act = - acc_g - vel_act*vel_act* air_res;
```

```
    // write outputs
    acceleration.write(acc_act);
    velocity.write(vel_act);
    position.write(height);}

  SCA_CTOR(bouncer){;}
};
```

Listing 4.3: *SystemC-AMS-Code für „bouncing ball“ - Verhaltensbeschreibung*

```
#include "bounce.h"

int sc_main(int argc, char* argv[])
{
  sc_set_default_time_unit(10.0, SC_US);

  // connections for tracing
  sca_sdf_signal<double> acceleration, velocity, position;

  // instanciate bouncing ball
  bouncer bounce1("Bounce1");
  bounce1.position(position);
  bounce1.velocity(velocity);
  bounce1.velocity.set_T(100.0, SC_US);
  bounce1.acceleration(acceleration);

  // signal termination
  terminator3 t3("Terminator3");
  t3.in1(position);
  t3.in2(velocity);
  t3.in3(acceleration);

  // trace section
  sca_trace_file *trp = sca_create_matlab_trace_file("bounce.
     dat");
  trp->add(acceleration, "acceleration");
  trp->add(velocity, "velocity");
  trp->add(position, "position");

  // start simulation
  sc_start(35,SC_SEC);
  sca_terminate();
  return 0;
}
```

Listing 4.4: *SystemC-AMS-Code Hauptprogramm für „bouncing ball“*

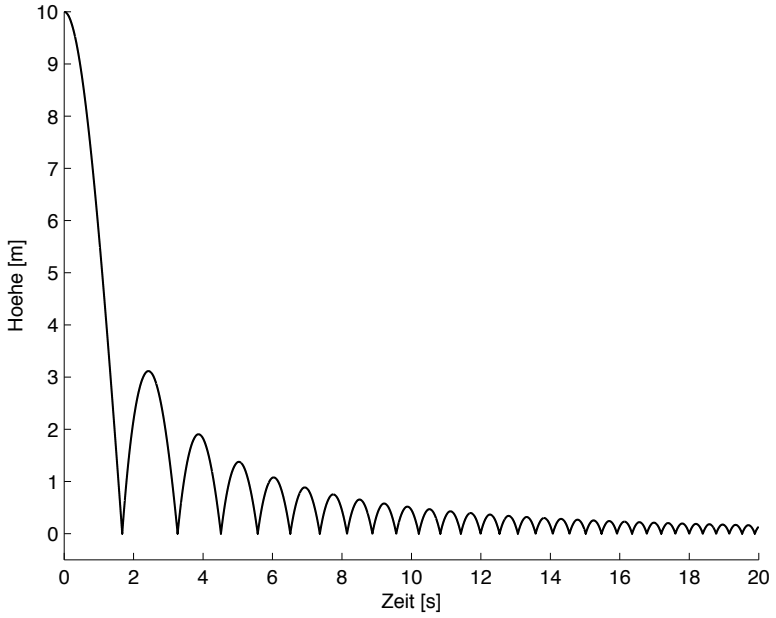


Bild 4.5.: *Ergebnis der SystemC-AMS-Simulation des aufspringenden Balls*

Weitere Beispiele

SystemC-AMS v0.15 zielt mit seinen Sprachkonstrukten und dem internen Aufbau vor allem auf Systeme im Telekommunikationsbereich [90]. Dies zeigt beispielsweise die Referenzimplementierung eines Telefons durch Einwich [91]. Hierbei handelt es sich um ein System mit umfangreicherem Digitalteil und relativ kleinen Analogteil. Die analogen Komponenten arbeiten dabei in sehr niedrigen Frequenzbereichen, so dass eine starke Überabtastung des analogen Teilsystems die Zeitkontinuität abbilden kann. Ebenfalls aus der Telekommunikationsbranche stammt die Implementierung eines Gigabit-Übertragers durch Grimm [31], die mit Hilfe polymorpher Signale verschiedene Abstraktionsebenen in einem Modell vereint. Die Modellierung eines Mikrorelais als Teil eines Messsystems für Mikrosysteme beschreibt Schneider [92]. Im Relais-Modell werden lineare und nichtlineare Anteile getrennt voneinander berechnet um die Beschränkungen der dort genutzten proprietären SystemC-AMS-Version zu umgehen.

Im Rahmen der Arbeiten zu dieser Dissertation entstanden mehrere Modelle von Mikrosystemen in SystemC-AMS, um die Eignung dieser Sprache zur MEMS-Modellierung zu prüfen. Die Basis dieser Modelle bildeten vorhandene verifizierte VHDL-AMS-Modelle und soweit möglich Messungen an den im Rahmen des SFB 379 bereits gefertigten Prototypen. Im folgenden sollen zwei dieser Modelle kurz erläutert werden. Darüber hinaus entstanden weitere Modelle, einen Überblick gibt [93].

Mikrospiegelarray

Mit Hilfe der ersten verfügbaren SystemC-AMS-Bibliothek v0.12 sollte das im Rahmen des SFB 379 entwickelte, elektrostatisch angesteuerte Mikrospiegelarray beschrieben und die Simulationsergebnisse mit den vorhandenen VHDL-AMS-Ergebnissen verglichen werden. Der Artikel [94] enthält neben einer ausführlicheren Beschreibung auch Bilder zum Simulationsverlauf.

Das Spiegel-Array besteht aus 25 quadratisch angeordneten Mikrospiegeln. Jeder dieser Spiegel besitzt durch seine Aufhängung an vier Federbändern zwei nutzbare Auslenkungsmöglichkeiten (x- und y-Richtung). Die Rotationsmomente führen jedoch zusätzlich zu einer Verschiebung in z-Richtung, was zusammen mit der Auslenkung eine Änderung der Ansteuerkapazität zur Folge hat. Im elektrischen Teil (elektrostatische Ansteuerung) werden die Eingangsspannungen sowie die momentanen Auslenkungen direkt in Gleichungen im Zeitbereich eingesetzt und aus den elektrostatischen Kräften bzw. sich ändernden Kapazitäten die neuen Drehmomente bzw. Kräfte in z-Richtung berechnet. Im mechanischen Teil erfolgt die Implementierung der Bewegungsgleichung im Bildbereich bzw. durch eine elektrische Analogie.

Bei der rückkopplungsfreien Modellierung des Spiegels erfolgt die Drehmomentberechnung ausschließlich auf Basis der angelegten Spannungen ohne Berücksichtigung der Position des Spiegels zu den Elektroden. Daher besitzt das Signalflussmodell keine Schleifen, sondern kann linear abgearbeitet werden. Bei dieser einfachen Beschreibungsform sind keine Unterschiede zwischen der Simulation mit SystemC-AMS und VHDL-AMS feststellbar. Jedoch nimmt die Simulation mit VHDL-AMS auf dem selben Rechner mit 3 min 20 s deutlich mehr Rechenzeit in Anspruch als mit SystemC-AMS (3 s) bei gleichen Parametern (Schrittweite 0,1 ms, Trapez-Integrationsalgorithmus).

Im Realbetrieb soll der Mikrospiegel für Bildprojektionen genutzt werden. Um die dabei notwendigen schnellen Auslenkungsänderungen zu erreichen wird der Spiegel im Resonanzbereich betrieben. Als Eingangsspannungen fungieren modulierte Sinusschwingungen. Die Modellierung des mechanischen Teils erfolgt im dynamischen Fall als lineares elektrisches Netz mit widerstandstreuer Analogietransformation. Dabei tritt jedoch im engeren Resonanzbereich eine geringfügig größere Überhöhung als bei der VHDL-AMS-Vergleichssimulation auf. Im Bereich außerhalb der Resonanz wird Deckungsgleichheit zwischen den SystemC-AMS-Simulationsergebnissen und den Referenzdaten erreicht. Auch im dynamischen Fall nimmt die Rechenzeit bei gleicher minimaler Auflösung (10 ns) mit SystemC-AMS (1 min 36 s) gegenüber VHDL-AMS (24 min) deutlich ab.

Vibrationssensor

Einen weiteren Testfall für die SystemC-AMS-Bibliothek v0.12 stellt ein System zur Vibrationsanalyse [95] dar. Es umfasst einen kapazitiven Sensor sowie die analoge und digitale Signalverarbeitung. Damit eignet es sich besonders gut für die Untersuchung der Beschreibung heterogener Systeme. Bild 4.6 zeigt den grundlegenden Systemaufbau als Blockschaltbild.

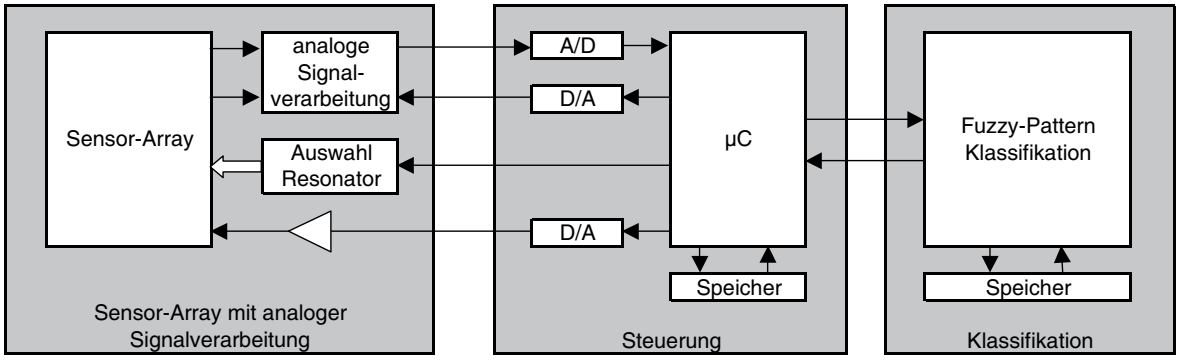


Bild 4.6.: *System zur Vibrationsanalyse*

Das Sensor-Array besteht aus einer Vielzahl lateral beweglicher Feder-Masse-Systeme, welche frequenzselektiv in Resonanz arbeiten, die Auswahl eines einzelnen Sensors obliegt dabei dem Steuerteil des Gesamtsystems. Die am Eingang aufgenommenen Werte werden dem Sensor-Array zugeführt und an dessen Ausgang einer analogen Signalverarbeitung bereitgestellt. Diese Daten dienen als Grundlage für die digitale Steuerung. In der Klassifikationskomponente erfolgt die Einteilung der im Steuerteil gewandelten Signale mittels Fuzzy-Pattern-Algorithmen in Güteklassen [96]. Im Bereich der Maschinenüberwachung können durch das Aufbringen der Sensoren auf verschleißkritische Maschinenteile Vibrationen klassifiziert werden und somit eine frühzeitige Erkennung der Abnutzung erfolgen.

Das Demonstratorsystem ist in die drei Funktionsbereiche Sensor-Array, Steuerung und Klassifikation eingeteilt. Ein Mikrocontroller steuert die Abläufe innerhalb des Systems und führt die notwendigen Digital-Analog- und Analog-Digital-Wandlungen durch. Er ist mit der rein digitalen Komponente Fuzzy-Pattern-Klassifikator verbunden, welche den Klassifizierungsprozess umfasst. Der Selektor wählt aus dem Array von acht Sensoren die Subkomponente mit passender Resonanzfrequenz aus. Zur Feinabstimmung der Sensoren dient die vom Mikrocontroller bereitgestellte Tuning-Spannung. Die analoge Signalverarbeitung umfasst einen Lock-In-Verstärker sowie Komponenten zur Tiefpassfilterung, Summation und Verstärkung.

In der Simulation zeigten sich beim Sensormodell geringe Abweichungen zum VHDL-AMS-Modell von etwa 2 %. Die Ursache hierfür ist in der notwendigen Entkopplung der Rückführungen in SystemC-AMS v0.12 zu suchen. Der Fehler beim Gesamtsystem verminderte sich durch die Klassifikation auf ca. 0,07 %, so dass auf Systemebene eine Nutzung des SystemC-AMS-Modells im akzeptablen Bereich liegt. Die Simulationszeit reduzierte sich etwa um den Faktor 8. Für weitere Ausführungen sei auf [95] verwiesen.

Schon mit dieser ersten SystemC-AMS-Version 0.12 zeigte sich, dass MEMS prinzipiell mit SystemC-AMS beschreibbar sind. Die Folgeversionen verbesserten die Modellierbarkeit weiter, die Modelle wurden entsprechend portiert.

Ausblick auf Version 1.0

Das sich derzeit in der Standardisierung befindliche SystemC-AMS wird einigen Änderungen unterworfen; im neuen LRM [97] finden sich neue Konstrukte und Schlüsselworte wieder. So werden die bisherigen zwei Modellierungsarten SDF und lineare Netze erweitert zu den drei Arten ELN (electrical linear networks), LSF (linear signal flow) und TDF (timed data flow, entspricht dem bisherigen SDF). Der im neuen Whitepaper [98] vorgestellte Schichtenaufbau unterscheidet sich jedoch kaum von dem in Bild 4.4 angegebenen Aufbau der Version 0.15. Der Schwerpunkt der Entwicklung liegt weiterhin auf Kommunikationssystemen, jedoch wird mit dem Automotive-Sektor auch ein klassisches Anwendungsgebiet von MEMS adressiert. Die in Tabelle 4.1 angegebenen vordefinierten Rufe im SDF-Bereich erhalten die in Tabelle 4.2 angegebenen neuen Bezeichner im TDF-Bereich. Im LSF-Bereich stehen vordefinierte Module wie z. B. Addierer/Subtrahierer und Ableitungsfunktionen zur Verfügung, um diese als gerichtete Netzwerkelemente benutzen zu können.

Funktionsname v0.15	**Funktionsname v1.0**
void attributes()	void set_attributes()
void init()	void initialize()
void sig_proc()	void processing() bzw. void register_processing (Alternativfunktion)
void ac_sig_proc()	void ac_processing() bzw. void register_ac_processing (Alternativfunktion)
void post_proc()	keine Entsprechung definiert

Tabelle 4.2.: *geplante vordefinierte Funktionen in SystemC-AMS 1.0*

5. Spezifikationserfassung und Entwurfsunterstützung durch SpecScribe

Die Spezifikation bildet den Ausgangspunkt des Entwurfsprozesses. Einen Überblick über die mit der Spezifikationserstellung verbundenen Abläufe gibt Wasson [99]. Die Erfassung von Spezifikationsdaten in formaler maschinenlesbarer Form ist die Grundvoraussetzung für eine rechentechnische Unterstützung des Designers während des Entwurfs. Die Spezifikation liegt zu Beginn üblicherweise in nichtformaler textueller Form, ggf. unterstützt durch Grafiken, vor. Werkzeuge wie DOORS [100] bieten eine Verwaltung dieser Dokumente an, können inhaltlich jedoch mangels Formalisierung keine weitere Unterstützung bieten.

Das an der Professur Schaltkreis- und Systementwurf der TU Chemnitz entstehende Werkzeug „SpecScribe“ soll hier Abhilfe schaffen. Die Gemeinschaftsarbeit mehrerer Entwickler [101,102] ist Gegenstand dieses Kapitels, wobei der Fokus auf den speziellen Konstrukten für heterogene Systeme liegt. Dabei besteht das Ziel nicht darin, vorhandene Werkzeuge zu ersetzen, sondern diese zu verbinden und so einen durchgängigen Ablauf von der Spezifikationserfassung über die Implementierung und Verifikation bis hin zur Inbetriebnahme und zum Lifecycle-Management zu bieten. Dieses Kapitel stellt die Grundlagen des Werkzeugs und seine Erweiterungen für heterogene Systeme dar, ein ausführliches Beispiel zeigt Kapitel 9.

SpecScribe hat seinen Ursprung in der Software SpecEdit von LucentTechnologies [103], die im ersten Unterkapitel kurz erläutert wird.

5.1. SpecEdit und die ADeVA-Semantik

Die komplexen Spezifikationen von Telekommunikationssystemen sowie deren schnelle Entwicklungszyklen führten bei Lucent Technologies Nürnberg zur Entwicklung einer Software „SpecEdit“ zur Formalisierung von rein digitalen Automaten sowie zur Speicherung der zugehörigen textuellen Anforderungen. Die Automatenbeschreibung erfolgte in der im folgenden Abschnitt beschriebenen ADeVA-Semantik. Diese Automatenbeschreibungen sind nach VHDL und C exportierbar und können somit als Ausgangspunkt für VHDL-basierte Modelchecker dienen.

5.1.1. ADeVA-Semantik

Die formale Spezifikationssprache ADeVA (Advanced Design and Verification of ASICs) [104] ist eine zustands- und übergangsbasierte Sprache für digitale Automaten (FSM), die im abstrakten Bereich asynchrones Verhalten modellierbar macht. Eine Automatenbeschreibung besteht aus zwei Arten von Tabellen, den MTT (Mode Transition Table) für Zustandsübergänge sowie den DTT (Data Transformation Table) zur Veränderung von Variablen und Ausgangsgrößen. Dabei entsprechen die MTTs dem Steuerpfad und die DTTs dem Datenpfad des Systems. Das genaue Aussehen der MTT/DTT wurde im Laufe der Entwicklung verändert, um eine einheitliche Bedienbarkeit zu gewährleisten. Das hier gezeigte Aussehen bezieht sich auf [103] aus dem Jahr 2006, wo syntaktisch kein Unterschied mehr zwischen MTT und DTT besteht, jedoch nach wie vor zwischen den Zugehörigkeiten zu Steuer- und Datenpfad unterschieden wird.

Mode Transition Table MTT

Die MTT als Zustandsübergangstabelle besteht aus drei Hauptbereichen. In der linken, ersten Spalte ist der derzeitige Zustand angegeben. Die folgende Spalte enthält dazu die jeweils erreichbaren Folgezustände. Die übrigen Spalten geben die Übergangsbedingungen an, wobei zwischen flankensensitiven (@T, @F) und zustandssensitiven (T, F) Übergängen unterschieden wird. Ein don't care-Feld (-) ermöglicht die Nichtberücksichtigung einzelner Bedingungen im konkreten Zustandsübergang.

Data Transformation Table DTT

Die Beschreibung der Änderung von Variablen und Ausgängen erfolgt mit Hilfe von DTTs. Diese Tabellen unterscheiden sich von den MTT durch den Ersatz der ersten beiden Spalten mit dem Variablennamen und dem zuzuweisenden Wert. Die Übergangsbedingungen folgen in den weiteren Spalten und nutzen dieselbe Syntax wie die MTT.

Beispiel

Die Funktionalität der MTT und DTT soll anhang eines kurzen fiktiven Beispiels gezeigt werden. Der Beispielautomat liest alle sechs Takte einen 32 bit breiten Wert vom Eingang „sample32“ in das interne Register word2byte. In den folgenden Takten wird dies byteseriell mit dem niederwertigsten Byte zuerst an den Ausgang „byteout“ angelegt.

Die MTT (Tabelle 5.1) besteht aus Zeilen für die drei Zustände receive, send und wait. Die Taktzählung des Eingangs clk erfolgt über eine Zählvariable count_clk. Die DTT in Tabelle 5.2 stellt die Zuweisung des Ausgangs „byteout“ dar. Durch das Zufügen der Bedingung @T(clk == true) erfolgt hier die Abbildung synchronen Verhaltens (steigende Taktflanke) in der prinzipiell asynchronen DTT.

currentMode	nextMode	count_clk == 0	count_clk == 3	count_clk == 5
receive	send	T	-	-
send	wait	-	T	-
wait	receive	-	-	T

Tabelle 5.1.: *Beispiel-MTT zur Wort-Byte-Wandlung*

Variable	Value	state == send	clk == true
byteout	word2byte[count_clk+1]	T	@T
byteout	0	F	-

Tabelle 5.2.: *Beispiel-DTT zur Wort-Byte-Wandlung*

5.1.2. Grenzen von SpecEdit

Das Tool SpecEdit verwendet für die Speicherung der Designdaten Dateien in einem XML-Format. Bei großen Designs führt dies zu entsprechend langen Zugriffsdauern und stellt ein Problem für Suchvorgänge dar. Darüber hinaus sind die Fähigkeiten, Hierarchieen abzubilden, eingeschränkt. Die Verwaltung der Anforderungen erfolgt strukturiert, wogegen MTTs und DTTs als eine gemeinsame Hierarchieebene betrachtet werden. Dies vereinfacht die Anwendung von Modelcheckern, jedoch ist ein modulares Entwurfskonzept bestehend aus Komponenten mit mehreren Hierarchieebenen nicht abbildbar.

Die Behebung dieser Grenzen im bisherigen Konzept erschien wenig erfolgversprechend. Daher wurde mit dem Nachfolgetool SpecScribe ein veränderter, datenbankbasierter Ansatz genutzt, welcher Gegenstand der folgenden Abschnitte sein soll.

5.2. Konzepte des Werkzeugs SpecScribe

Das Werkzeug SpecScribe hat zum Ziel, ein Produkt von der ersten textuellen Spezifikation bis zur Fertigung und darüber hinaus auch während seines Lebenszyklus zu begleiten. Die dabei entstehende Datenmenge erfordert eine flexible, schnell durchsuchbare Struktur. Dabei muss beachtet werden, dass eine Entwurfsautomatisierung nur durch eine Formalisierung erreicht werden kann. Eine reine informale Dokumentenverwaltung, wie beispielsweise in DOORS [100], genügt nicht allen Anforderungen der Designer. Die Formalisierung ermöglicht eine Prüfung der Spezifikationsdaten auf

- Vollständigkeit,
- Widerspruchsfreiheit und
- Eindeutigkeit.

Neben diesen Fragestellungen auf Spezifikationsebene soll SpecScribe in den weiteren Designschritten folgende Domänen unterstützen:

- Anforderungsmanagement (wer hat was und wann verändert),
- Implementierung,
- Design Space Exploration (Suche nach alternativen Implementierungen),
- Modellierung und Simulation auf verschiedenen Abstraktionsebenen,
- Verifikation und Test,
- Qualitätssicherung und
- Dokumentation.

Um dabei eine größtmögliche Breite von Rechnerplattformen zu unterstützen, wurde auf die für viele Betriebssysteme verfügbaren Sprachen Python [105] und Qt [106] zurückgegriffen. Dies stellt sicher, dass die grafische Oberfläche sowohl in der Windows-Welt als auch bei Nutzung von MacOS oder Unix-basierten Systemen benutzbar bleibt.

Der folgende Abschnitt erläutert kurz das als Ausgangspunkt der Arbeit in Theorieform [107] vorhandene allgemeine Konzept von SpecScribe. Die Umsetzung der Struktur in Python-Code als Basis der Erweiterung für Mikrosysteme erfolgte im Rahmen dieser Arbeit. Die darüber hinaus geschaffenen speziellen Konzepte und Konstrukte zur Modellierung heterogener Systeme stehen im Mittelpunkt des Abschnitts 5.3. Klassennamen sind in den folgenden Abschnitten *kursiv* gesetzt.

5.2.1. Allgemeines Konzept

Für die konzeptionelle Ausrichtung von SpecScribe musste ein Kompromiss zwischen größtmöglicher Unterstützung von Spezialfällen bei gleichzeitiger Wahrung der Generalität für eine beherrschbare Umsetzung des Tools in Software gefunden werden. Das Grundkonzept von SpecScribe lässt sich in drei Bereiche unterteilen:

1. Anforderungen und Strukturinformation auf Modulebene,
2. Beschreibungen des Modulinneren und
3. Schnittstellen zu weiteren Sprachen und Tools.

In diese drei Bereiche ordnen sich die speziellen Konzepte der Domänen Anforderungsmanagement, Implementierung, Verifikation/Test, Qualitätssicherung und Dokumentation wie in Tabelle 5.3 dargestellt ein.

Der Bereich 1 bildet den Kern von SpecScribe. Die hier angelegten Klassen und Konzepte sind so generell wie möglich gehalten, um in diesem Bereich spätere Änderungserfordernisse zu vermeiden. Die zu Bereich 1 gehörenden Daten haben einen allgemeingültigen Charakter, der in jedem Designprozess zu berücksichtigen ist.

Domäne	Bereich 1	Bereich 2	Bereich 3
Anforderungen	informale Requirements (Texte, Bilder), Anforderungsmanagement	formalisierte Requirements (z. B. abstrakte Automatenbeschreibungen)	Modelchecker, Textverarbeitung
Verifikation und Test	Struktur	Properties, Testskripte	Modelchecker
Implementierung, Modellierung, Simulation		FSM, hybride Automaten, Register	Simulatoren
Design Space Exploration		Kosten	Optimierer
Qualitätssicherung		Entwurfsbegleitende Maßnahmen (DFMEA [108]), Lifetime Monitoring	Identifikatoren für potentielle Fehlerquellen, Werkstatt-Interface
Dokumentation		Implementierungsdetails, Simulationsergebnisse	Textverarbeitung

Tabelle 5.3.: *Einordnung der Domänen in die Bereiche*

In Bereich 2 findet sich unter anderem das aus dem Vorgängerwerkzeug SpecEdit bekannte Konzept der MTT/ DTT wieder. Die Aufteilung in MTT und DTT besteht hier nicht mehr, vielmehr erlauben die Zustandsübergangstabelle und die Zustandsliste nun die Angabe von Handlungen (*Actions*) auf den Variablen. In Bereich 2 erfolgt die Diversifizierung des Designprozesses. Es treten je nach Designschritt und Systemtyp unterschiedliche Klassen auf, so z. B. spezielle Klassen zur Automatendarstellung, für analoge Problemstellungen sowie für Registerbeschreibungen. Der Bereich 2 stellt zusammen mit der grafischen Benutzeroberfläche den Hauptarbeitsbereich der Entwickler von SpecScribe dar.

Die Schnittstellen nach außen sind Gegenstand des Bereichs 3. Dabei ist geplant, für externe Erweiterungen standardisierte Schnittstellen, sogenannte Plug-In-Möglichkeiten, zu bieten. Im Speziellen handelt es sich bei den Bereich-3-Programmen um Funktionen zum Im- und Export zu verschiedenen Sprachen (SystemC, VHDL) und Tools.

SpecScribe verfolgt die Logik eines anforderungsbasierten Entwurfs, d. h. jeder Entwurfsschritt muss mit einer Anforderung begründbar sein. Daher startet der Entwicklungsprozess mit der Eingabe der Anforderungen in das Tool. Diese Anforderungen

können durch Detailbeschreibungen konkretisiert oder durch die Nutzung von Konstrukten des zweiten Bereichs aus der Anforderungsdomäne in die jeweiligen Spezialdomänen wie Implementierung oder Verifikation überführt werden. Dabei werden alle Ebenen des V-Modells (Bild 1.3) von den Anforderungen über die Implementierung bis zur Inbetriebnahme durchlaufen.

5.2.2. Datenstruktur

In der Datenstruktur von SpecScribe finden sich die drei Bereiche des Grundkonzeptes wieder. Auf Basis dieser Struktur erfolgt das Ablegen der Designinformationen in der Datenbank. Im folgenden sollen kurz die wichtigsten Klassen von SpecScribe dargestellt werden, soweit dies zum Verständnis der Erweiterungen für heterogene Systeme notwendig ist.

SpecItem

Die Klasse *SpecItem* bildet die Basisklasse für alle Elemente von SpecScribe. Sie stellt die grundlegenden Informationen wie Name oder Änderungsdatum bereit und speichert Informationen zur Darstellung des jeweiligen Elements in der GUI. Tabelle 5.4 zeigt die Mitglieder der Klasse *SpecItem*.

Member	Typ	Bedeutung
ID	String	Spezifikationsweit eindeutige Zeichenkette zur Elementidentifikation (wird automatisch erzeugt)
name	String	Name des Elements
description	String	Beschreibung des Elements
section	int []	Abschnittsnummer in der Dokumentation
owner	User	Ersteller des Elements
created	DateTime	Erstellungsdatum und -uhrzeit
whoChanged	User	zuletzt geändert von
whenChanged	DateTime	Datum und Uhrzeit der letzten Änderung
historyElement	SpecItem	Inhalt des Elements vor der letzten Änderung
newerElement	SpecItem	Inhalt des Elements nach der nächsten Änderung
attributes	Attribut []	Liste von zugehörigen Attributen (Erläuterungen)
parentItem	SpecItem	in der GUI-Darstellung nächsthöheres Element
childItems	SpecItem []	in der GUI-Darstellung nächstniedrigere Elemente
upperTraces	Trace []	Verbindungen zu anderen SpecItems mit hinweisendem Charakter

Tabelle 5.4.: *Mitglieder der Klasse SpecItem*

Requirement

Die Klasse *Requirement* nimmt die Anforderungen eines Systems auf. Sie fungiert dabei als Basisklasse für spezialisierte Requirements und wird selbst nicht direkt verwendet. Als Kind von SpecItem erbt sie alle Konstrukte dieser Universalklasse. Folgende Klassen sind als Spezialfälle direkt von ihr abgeleitet:

- *TextualRequirement* für Anforderungen in unformatiertem Text,
- *FileRequirement* für Anforderungen in Dateiform (z. B. Bilder, formatierte Texte, Referenzmodelle),
- *ComponentRequirement* für Anforderungen, die eine konkrete Implementierung fordern,
- *MemoryRequirement* für Registerbeschreibungen,
- *VerificationRequirement* für Anforderungen an die Verifikation des Systems und
- *AnalogRequirement* für analoge Anforderungen.

Die Mitglieder der Klasse (siehe Tabelle 5.5) umfassen sowohl Konstrukte zur hierarchischen Verknüpfung von Requirements als auch zur Behandlung der Erfüllung der Anforderungen, die Elemente sind dabei aus der Theorie des Requirements-Management abgeleitet. Die Anforderungshierarchie bildet eine Baumstruktur, die der Dokumentenstruktur der Spezifikation entspricht.

Member	Typ	Bedeutung
satisfactionState	SatisfactionState	Status der Erfüllung der Anforderung (approved, qualified, implemented, verified, tested, accepted)
parentrequirement	Requirement	hierarchisch übergeordnetes Requirement
subrequirements	Requirement []	hierarchisch untergeordnete Anforderungen
lowerTrace	Trace	Verbindung zu anderen SpecItems mit hinweisendem Charakter

Tabelle 5.5.: *Mitglieder der Klasse Requirement*

Component und FSM

Die Klasse *Component* bildet die Basisklasse für alle strukturalen Informationen und damit auch für alle Implementierungen. Sie umfasst Informationen zu Schnittstellen (*Ports*) und Verbindungen (*Connection, Bus*) und fungiert als Elternklasse für hierarchische (*StructuredComponent*) und Verhaltensmodelle (*BehavioralComponent*).

Die Beschreibung von Abläufen und Algorithmen im digitalen Bereich erfolgt mit Hilfe der von *BehavioralComponent* abgeleiteten Klasse *FSM*. Diese Verhaltensbeschreibung

einer Komponente basiert auf der Logik der sequentiellen Automaten. Innerhalb einer digitalen FSM laufen die Signalzuweisungen sequentiell nach der Reihenfolge der Eingabe bzw. nach der angegebenen Priorität ab. Mehrere FSM werden zueinander parallel abgearbeitet, um die Nebenläufigkeit von Hardware und Softwarethreads zu modellieren. Die Klasse *FSM* ermöglicht außerdem die Abbildung des MTT-/DTT-Konzepts von SpecEdit in SpecScribe und damit die Wiederverwendung der bereits in SpecEdit erstellten Spezifikationen. Dazu sind jedoch Umordnungen der Tabellen erforderlich, um die in SpecEdit vorhandene Trennung von Daten- und Steuerpfad auf die gemeinsame Klassenstruktur abzubilden. Eine MTT entspricht einer Instanz von *FSM*, der die anhängenden DTT als Signalzuweisungen zugeordnet werden. Den Aufbau der Klasse *FSM* als vereinfachtes Klassen-Diagramm zeigt Bild 5.1.

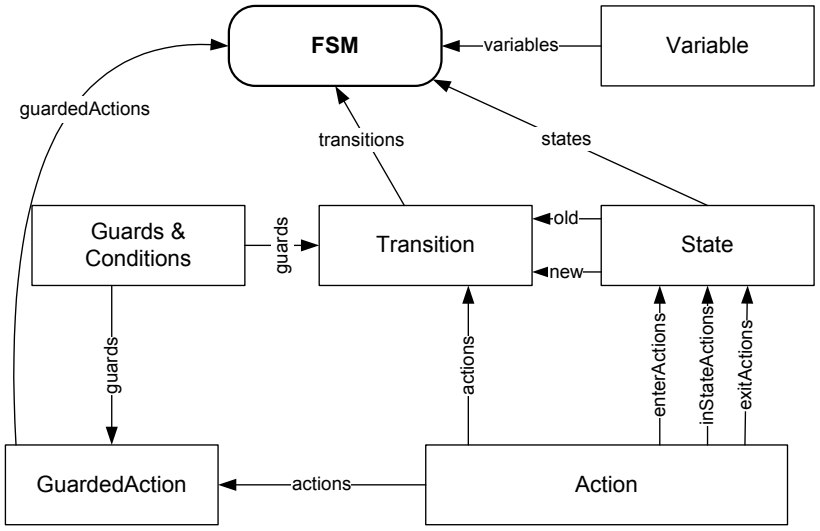


Bild 5.1.: *vereinfachtes Klassendiagramm der Klasse FSM*

Eine FSM verfügt über Variablen, die als Hilfsgrößen die internen Abläufe unterstützen. Einer Variable kann ein Port zugeordnet sein, um das Ausgangsverhalten des Automaten weiterzureichen. Die Klasse *Transition* beschreibt Zustandsübergänge zwischen *States*. Diese Transitionen modellieren anhand der Bedingungen (*Guards* und *Conditions*) die sequentielle Abfolge der einzelnen Algorithmenabarbeitungsstufen. An die Zustandsübergänge sowie an die Zustände selbst können durch *Actions* Wertzuweisungen an Variablen unter Zuhilfenahme von Termen der Klasse *Expressions* erfolgen.

5.2.3. Grafische Bedienoberfläche (GUI)

Die Interaktion mit dem Nutzer erfolgt über eine in Bild 5.2 dargestellte zweigeteilte grafische Oberfläche. Auf der linken Seite visualisiert ein Baum die Struktur der Anforderungen und Implementierungen, die rechte Seite erlaubt die Kontrolle und Eingabe von Details zu den Einzelpositionen des linken Baums. Aktionen im Baum wie das Hinzufügen und Löschen von Elementen sind mittels des per Rechtsklick erreichbaren

Kontextmenüs möglich (siehe Bild 5.3). Die Implementierung der GUI ist nicht Teil dieser Arbeit.

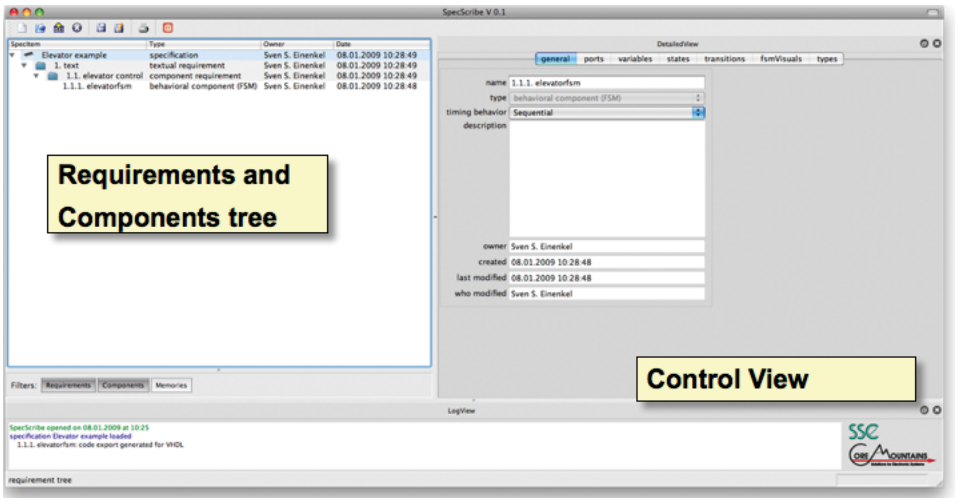


Bild 5.2.: *Screenshot der SpecScribe-GUI*

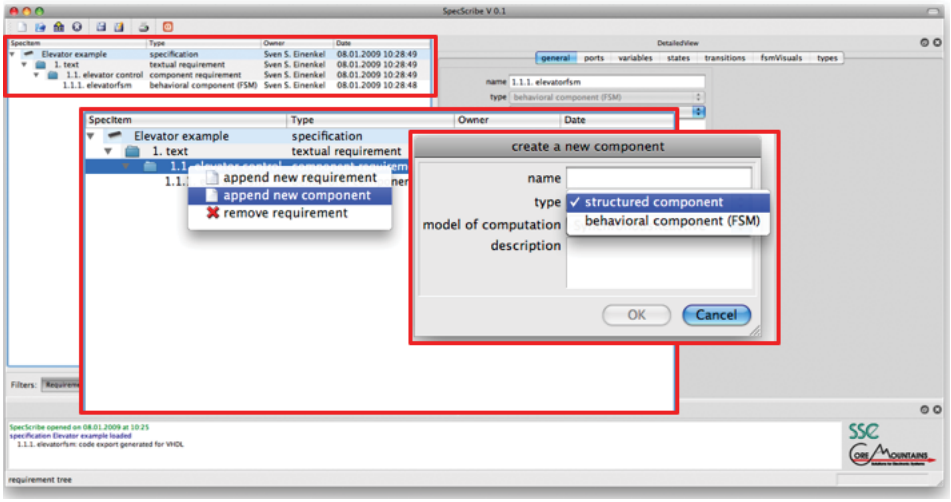


Bild 5.3.: *Screenshot des Kontext-Menüs*

5.2.4. Beispiel Ampelsteuerung

Die Funktionsweise des vorgestellten Konzepts im Digitalbereich soll kurz am Beispiel einer Ampelsteuerung demonstriert werden [102]. Aus Platz- und Komplexitätsgründen beschränken sich die Ausführungen auf eine Variante mit vier Ampeln wie in Bild 5.4 dargestellt. Eine Variante mit 16 Ampeln konnte erfolgreich beschrieben werden, überstieg jedoch die Komplexitätsgrenze des genutzten SAL-Modelcheckers [109].

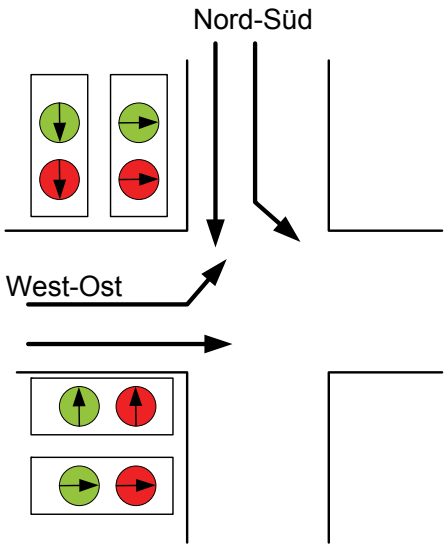


Bild 5.4.: *Skizze des Beispiels Ampelsteuerung*

Im anforderungsgetriebenen Entwurf steht am Ausgangspunkt das Requirement „Absicherung einer Kreuzung zweier Straßen“. An diese Anforderung schließen sich Konkretisierungen wie „Setze für jede ankommende Straße zwei Ampeln ein – eine für Geradeausfahrer sowie eine für Linksabbieger“. Der Übergang zur Implementierung erfolgt durch die Definition von Komponenten wie der einzelnen Ampel sowie dem Modell der Kreuzung als Verschaltung von vier Ampeln. Dies zeigt sich im Screenshot (Bild 5.5) anhand der Kombination der vier Instanzen der Komponente „redgreenfsm_4tl“ zum hierarchischen Modul „crossing_4tl“ und der dabei notwendigen Signale zur Kommunikation der vier Ampeln.

Das Innenleben einer einzelnen Ampel zeigt das Bild 5.6 mit der Zustandsübergangstabelle als Kombination aus MTT und DTT. Die Ampeln synchronisieren und arbitrieren sich dabei über die Prioritätssignale „request“ und „conflict“ selbst, es existiert keine globale Steuerungskomponente. Um keine Ampel zu benachteiligen, sperrt sich eine Ampel nach ihrer Grünphase solange selbst, bis alle anderen „request“-Signale bedient wurden.

Das Beispiel der Ampelkreuzung wurde erfolgreich beschrieben und mit Hilfe der im Rahmen dieser Arbeit geschaffenen Codegeneratoren (siehe auch Abschnitte 5.3.4 und 6.2) automatisiert nach SAL, SystemC und VHDL exportiert. Im Modelchecker SAL waren zur Reduzierung der Komplexität manuelle Anpassungen der Datentypen nötig (Reduzierung des Wertebereichs bei *Integer* auf den real genutzten Bereich von z. B. 0 bis 7), wogegen die erzeugten Quellcodes für SystemC und VHDL ohne Änderungen simuliert werden konnten.

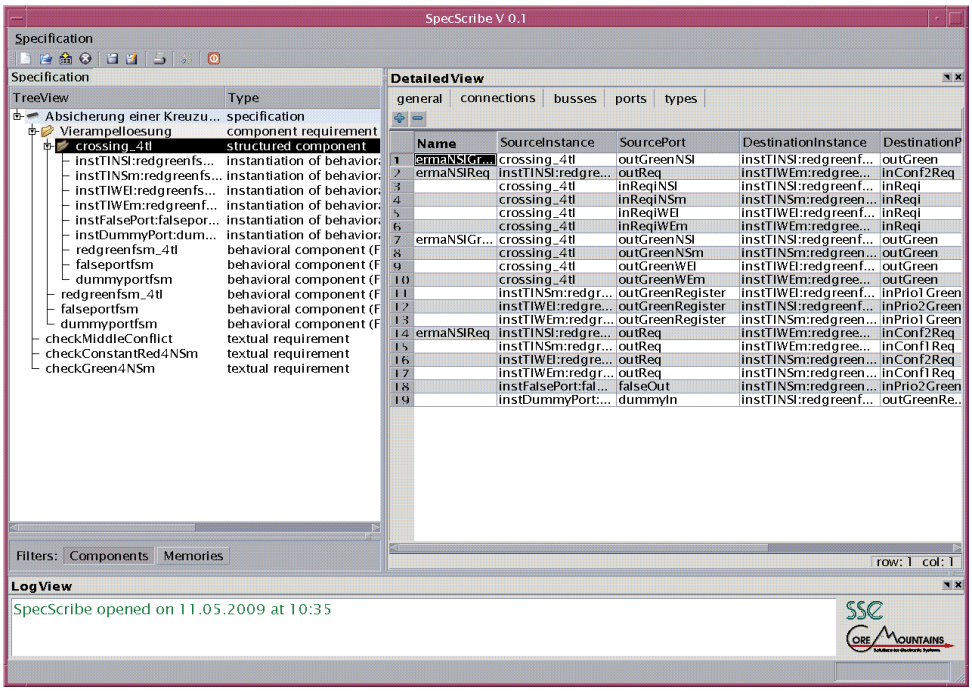


Bild 5.5.: *Screenshot des hierarchischen Moduls der Ampelkreuzung*

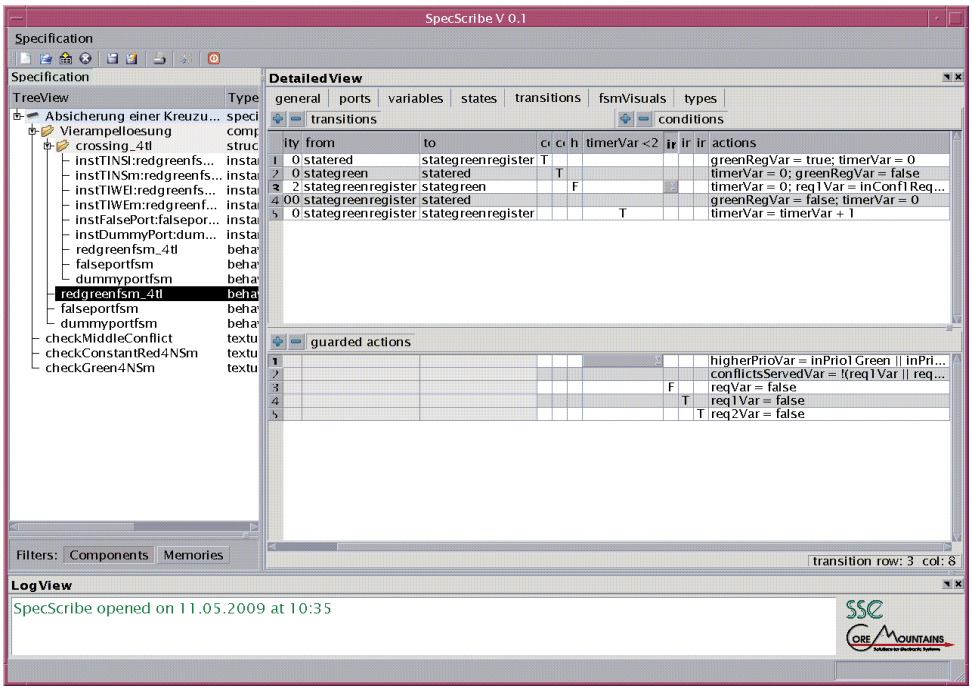


Bild 5.6.: *Screenshot der Komponente Ampel*

5.3. Erweiterungskonstrukte in SpecScribe für heterogene Systeme

Die bisher dargestellten allgemeinen Konzepte berücksichtigen nicht alle der für den Entwurf von Mikrosystemen notwendigen Parameter und Konstrukte, die in Kapitel 3.5 aufgezeigt wurden. Die im Rahmen dieser Arbeit geschaffenen Ergänzungen umfassen im Wesentlichen die Punkte:

1. Analoge Anforderungen,
2. Abbildung von analogen Abläufen in Form hybrider Automaten,
3. Schaltplandarstellung und
4. Code-Generatoren.

Eine Auswahl dieser Punkte wurde in [110] vorgestellt. Mit Hilfe der Erweiterungen gelingt es, das Ziel dieser Arbeit im Bereich der Spezifikation zu erreichen.

5.3.1. Analoge Anforderungen

AnalogRequirement

Zur Spezialisierung der allgemeinen textuellen oder dateibasierten Anforderungen entstand eine Erweiterung der Basisklasse *Requirement* namens *AnalogRequirement*. Die im Rahmen dieser Klasse angegebenen formalen und nichtformalen Anforderungen sind damit explizit als dem wert- und zeitkontinuierlichen Bereich zugehörig gekennzeichnet. Gleichzeitig fungiert diese Klasse als Basis für speziellere Anforderungsbeschreibungen in formalisierter Form wie z. B. die im folgenden Abschnitt dargestellte Klasse *ReferenceSignalForm*.

ReferenceSignalForm

ReferenceSignalForm erlaubt als Kind der Klasse *AnalogRequirement* die Angabe von Signalverläufen. Beim Design der Klasse wurde besonderer Wert auf eine möglichst weit reichende Generalität der Beschreibungsform gelegt. Um den Charakter als analoges Requirement zu verdeutlichen, sind als Achsen der Signalverläufe nur sogenannte *PhysicalSignals* zugelassen, die eine Spezialisierung der aus dem Konzept der FSM stammenden Variablen darstellen.

Die Klasse *PhysicalSignal* umfasst neben den von der Elternklasse *Variable* geerbten Mitgliedern wie Initialwert und Datentyp die Erweiterungen *unit* als physikalische Einheit (z. B. Zentimeter) und *physicalType* als pyhsikalischen Typ (z. B. Länge). Dies ermöglicht es, in einem späteren Konsistenzcheck sicherzustellen, dass nur Signale mit

entsprechender Typen- und Einheitenübereinstimmung miteinander ins Verhältnis gesetzt werden. Eine Unterscheidung rein auf dem Datentyp basierend ist hier nicht ausreichend, da alle wertkontinuierlichen Vorgänge einen Gleitkommatyp (float oder double) erfordern.

Die Werte der Signalverläufe sind als *Expression* angebbar. Diese bereits bei den sequentiellen Automaten genutzte Klasse repräsentiert Terme einer Gleichung. Dabei sind folgende Unterklassen definiert:

- *Literal* für Konstanten,
- *Aggregation* für Listen und Wertesammlungen,
- *BinaryOperation* für Ausdrücke mit zwei Operanden (bspw. '+', Potenzen, Vergleichsoperatoren),
- *UnaryOperation* für Ausdrücke mit einem Operanden (bspw. Negation) und
- Referenzen auf Ports und Variablen.

Im rein digitalen Fall enthalten diese Terme nur lineares Zeitverhalten für Zuweisungen. Für analoge Sachverhalte reicht dies, wie in Kapitel 3.5 gezeigt, nicht aus. Daher wurde die Klasse Expression um Operatoren für Differentialgleichungen und häufig vorkommende kontinuierliche Signalverläufe erweitert. Dies umfasst insbesondere die Operatoren:

- *Derivative* als Ableitungsoperator,
- *Integration* als Integral-Operator,
- die Winkelfunktionen und zugehörigen Arcus-Funktionen.

Eine weitere Möglichkeit neben Differentialgleichungen und algebraischen Zusammenhängen bietet der Expression-Subtyp *Aggregation* als Sammlung von Wertepaaren, welche beispielsweise aus Labormessreihen bestimmt wurden. Die angegebenen Signalverläufe können als periodisch gekennzeichnet werden, so dass z. B. stückweise lineare Funktionen mit wiederkehrenden Werten einfach in der Datenstuktur abbildbar sind. Darüber hinaus ermöglicht die Angabe eines Toleranzbandes die unvermeidbaren Wertschwankungen in der Praxis. Daraus ergibt sich die Klassenstruktur gemäß Tabelle 5.6.

Member	Typ	Bedeutung
horizontalAxis	PhysicalSignal	X-Achse des Signalverlaufs
verticalAxis	PhysicalSignal	Y-Achse des Signalverlaufs
form	Expression	Signalverlauf
periodical	Expression	Periodizität des Signalverlaufs
toleranceUp	Expression	rel. Toleranzband oberhalb des Signalverlaufs
toleranceLow	Expression	rel. Toleranzband unterhalb des Signalverlaufs

Tabelle 5.6.: *Mitglieder der Klasse ReferenceSignalForm*

Der in Bild 5.7 dargestellte Signalverlauf der Geschwindigkeit eines Beschleunigungssensors entspricht beispielsweise den in Tabelle 5.7 angegebenen Werten der Datenstruktur und kann, wie in Bild 5.8 dargestellt, in der SpecScribe-GUI eingegeben werden.

Member	Wert
horizontalAxis	Zeit t in s
verticalAxis	Geschwindigkeit v in $\frac{m}{s}$
form	$\frac{dv}{dt} == -cos(t)$
periodical	0 (durch Cosinus implizit gegeben)
toleranceUp	0,2
toleranceLow	0,1

Tabelle 5.7.: *Datenstruktur für Beispiel Beschleunigungssensor*

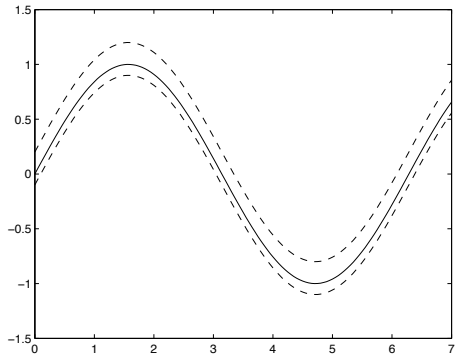


Bild 5.7.: *Signalverlauf zur ReferenceSignalForm in Tabelle 5.7*

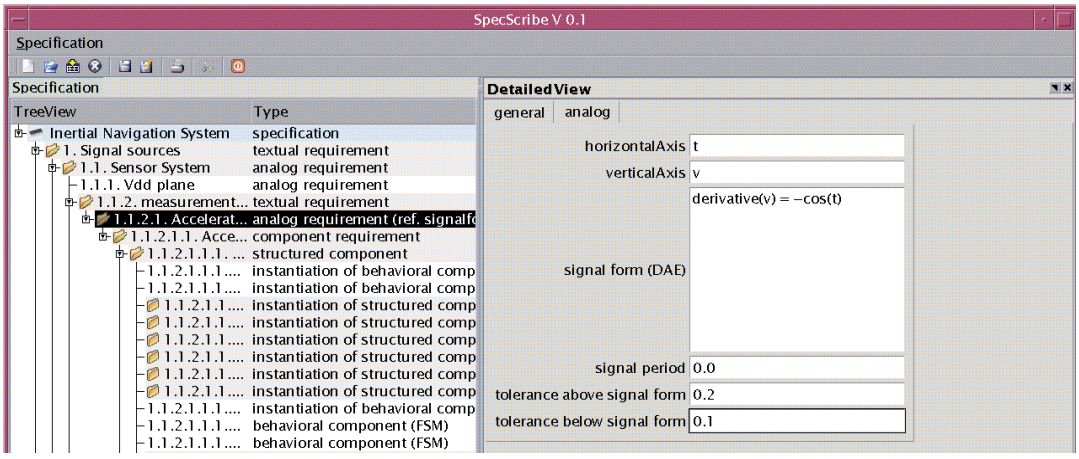


Bild 5.8.: *GUI-Eingabe der ReferenceSignalForm*

5.3.2. Abbildung von analogen Abläufen

Digitale diskrete Abläufe lassen sich mit Hilfe von Zustandsübergangsbeschreibungen (FSM) auf verschiedenen Abstraktionsebenen ereignisbasiert einfach darstellen. Für zeit- und wertkontinuierliche Abläufe im elektrischen und nichtelektrischen Bereich ergibt sich die in Kapitel 4.1 aufgezeigte Möglichkeit, hybride Automaten zu nutzen. Dies erfordert gegenüber den in Abschnitt 5.2.2 dargestellten rein digitalen FSM sowohl konzeptionelle Erweiterungen als auch neue Elemente in der Datenstruktur. Beispiele für die Umsetzung hybrider Automaten mit SpecScribe anhand eines Beschleunigungssensors zeigt das Kapitel 9.

Hybride Automaten umfassen nach [61] neben den klassischen Bestandteilen der FSM (Zustände, Zustandsübergangsverhalten, Ausgangsverhalten) in jedem Zustand Invarianten zur Systemprüfung. Diese als „wahr“ oder „falsch“ bestimmbaren Ausdrücke (Gleichungen oder Ungleichungen) müssen während der gesamten Zeit erfüllt sein, in der der Automat in einem Zustand verharrt. Ist eine Invariante nicht mehr erfüllt, muss zwingend ein Zustandsübergang durchgeführt werden, anderenfalls liegt ein Fehler im Automaten vor. Die Invarianten bieten sich für eine Korrektheitsprüfung des Automaten und damit auch für die spätere Nutzung als analoge Testpattern an.

In klassischen digitalen FSM erfolgt die Änderung von Ausgangswerten bzw. Variablen per Zuweisung. Bei analogen Zusammenhängen, insbesondere wenn es sich um Differentialgleichungen handelt, sind Gleichungen nur selten in aufgelöster Form angebbar. Hier überwiegt die Angabe „echter“ Gleichungen, die im Kontext des Gesamtsystems gelöst werden müssen. Für die Angabe von Zuweisungen ist in der Datenstruktur die Klasse *Action* vorgesehen. Diese besteht aus zwei Feldern für die linke Seite (Variable) und die rechte Seite (Zuweisungsvorschrift), welche per „=“ miteinander verbunden sind. Die Zuweisungsvorschrift kann dabei ein beliebiger Ausdruck der Klasse *Expression* sein, wogegen die linke Seite einen Verweis auf eine Variable enthalten muss. Für analoge Zusammenhänge ist eine solche strikte Zuweisungsform nicht mehr ausreichend. Vielmehr muss hier sowohl die linke als auch die rechte Seite beliebige Terme der Klasse*Expression* aufnehmen können. Die Terme umfassen, wie im Abschnitt zur ReferenceSignalForm erläutert, zusätzlich algebraischen Operatoren auch Ableitungsoperatoren und Integratoren.

Neben den algorithmischen Problemstellungen der analogen Erweiterung müssen auch im strukturellen Bereich neue Konstrukte eingeführt werden. Im Digitalbereich findet immer ein gerichteter Datentransport statt. In der Analogwelt existieren dazu je nach Abstraktionsebene verschiedene Möglichkeiten, Module miteinander zu verbinden. Auf der einen Seite spricht man von sogenannten „nichtkonservativen Knoten“, in denen wie im Digitalbereich ein gerichteter Informationsfluss erfolgt. Die Anschlüsse besitzen demzufolge eine Richtungsinformation (Ein- oder Ausgang). Auf der anderen Seite verwendet man die sogenannten „konservativen Knoten“ für die Darstellung vermaschter Netze von Analogmodulen. In einem Netz konservativer Knoten gelten die Kirchhoffschen Gesetze wie in Kapitel 3.5 dargestellt.

Beide Knotentypen haben gegenüber dem Digitalbereich gemein, dass sie rein zeitschrittgesteuert arbeiten. Im Gegensatz zu ereignisgesteuerten digitalen Signalen erfolgt hier eine Berechnung kontinuierlich in jedem Simulationszeitschritt, da im Analogbereich durch die Wertkontinuität keine Ereignisse existieren. Diese neuen Knoten erfordern eine Erweiterung der Datenstruktur um die Klasse *AnalogSignalInterface*, welche die Basisklasse *SignalInterface* um konservative und nichtkonservative analoge Verbindungen erweitert. Weitere abgeleitete Klassen von *SignalInterface* sind *DigitalSignalInterface* für kombinatorische Verbindungen und *ClockedSignalInterface* für Verbindungen über Register.

5.3.3. Schaltpläne und Layoutinformationen

Neben den bisher genannten algorithmischen Problemstellungen besteht auch Optimierungsbedarf bei der Bearbeitung von Schaltplänen. So benötigte auch das als Anwendungsbeispiel fungierende Inertialnavigationssystem (siehe Kapitel 9) mehrere Leiterplatten zur Aufnahme der Sensoren und der Auswerteschaltungen. Bisher erfolgt die Schaltplaneingabe meist mit Hilfe von grafischen Editoren in Schematics. Gerade bei häufig wiederkehrenden Schaltungsteilen und bei komplexen Einzelschaltkreisen mit vielen Anschlusspins gerät der Schematics-Entwurf an seine Grenze. Hier würde eine tabellenbasierte Eingabemöglichkeit durch Nutzung von Kopierbefehlen eine Steigerung der Entwurfseffektivität bedeuten. Daher soll in der Datenstruktur die Aufnahme derartiger Daten vorbereitet werden.

Die bisherige Datenstruktur enthält bereits die grundlegenden Elemente zur Verbindung von Modulen in Form von *Ports* und *Connections*. Diese müssen für die Nutzung in Schaltplänen und für Layouts erweitert werden:

- Gruppierung von Verbindungen zu Bussen in der neuen Klasse *Bus*,
- Einfügen von linearen Elementen in den Signalpfad (z. B. Serien-/Parallelwiderstände),
- Parametrisierung der Ports (Treiberstärke, Eingangslast),
- interne Zusatzelemente (Pull-up/Pull-down Widerstände) und
- Gehäuseparameter (Pins, Abmessungen).

Ausgehend von diesen Anforderungen werden die Klassen *Port* und *Component* mit physikalischen Parametern erweitert:

Port zu CircuitPort

Ein Port eines Schaltkreises verfügt im Gegensatz zu einem internen *Port* zur Verbindung von Modulen über die in Tabelle 5.8 aufgeführten Zusatzinformationen zu physikalischen Parametern und angeschlossenen Kleinkomponenten.

Member	Typ	Bedeutung
internalAssociatedComponents	PhysicalSignal []	interne lineare Elemente (Pullups/ Pulldowns)
driverStrength	PhysicalSignal	Treiberstärke
inputLoad	PhysicalSignal	Last
setupTime	PhysicalSignal	Setupzeit
holdTime	PhysicalSignal	Holdzeit

Tabelle 5.8.: *Erweiterungen von Port für CircuitPort*

Component zu CircuitPackage

Eine Komponente auf Leiterplattenebene verfügt in der Regel über ein Gehäuse. Die bekannten Ausnahmen der direkten Implementierung auf oder in die Leiterplattenstruktur haben bisher noch keinen relevanten Marktanteil erreicht. Das Gehäuse als schützende Hülle des ICs bestimmt den Platzbedarf des Bauteils und hat Einfluss auf das thermische Verhalten. Dem Gehäuse sind die Schaltkreispins zugeordnet, welche üblicherweise durch Bonddrähte mit den *CircuitPorts* verbunden sind. Die Angabe eines Symbols ermöglicht die Nutzung des Packages in einem grafischen Editor.

Tabelle 5.9 zeigt die Mitglieder der Klasse.

Member	Typ	Bedeutung
type	Zeichenkette	Gehäusetyp (BGA, DIL, TSOP, ...)
pins	Pin []	vorhandene Pins
symbol	Zeichenkette	Pfad zu einer Grafikdatei
length	PhysicalSignal	Gehäuselänge
width	PhysicalSignal	Gehäusebreite
height	PhysicalSignal	Gehäusehöhe

Tabelle 5.9.: *Erweiterungen von Component für CircuitPackage*

Component zu IntegratedCircuit

Das Gehäuse und die auf dem Siliziumchip implementierte Funktionalität bilden für den Nutzer eine Einheit. Diesem wird mit der Verwendung der Klasse *IntegratedCircuit* Rechnung getragen. Die Klasse erweitert die Elternklasse *Component* um die Angaben zum Package (*CircuitPackage*) sowie zum Hersteller (Membervariable „manufacturer“). Alle auf einer Leiterplatte eingesetzten Module sollten von diesem Typ gebildet werden. Für einfache, oft benötigte Module existieren vorgefertigte Klassen, so z. B. für Zweitore (*TwoPort*), welche neben Gehäuseinformationen nur noch die Angabe einer Transferfunktion erwarten sowie deren Spezialisierung *LinearTwoPort* für lineare Bauelemente wie Spulen, Widerstände und Kondensatoren.

Beispiel

Die Nutzung der vorgestellten Klassen zur Schaltplanbearbeitung soll kurz anhand eines sehr einfachen Beispiels gezeigt werden. Bild 5.9 zeigt ein aus zwei ICs bestehendes System, in welchem zwei Signale den Datenaustausch ermöglichen. Dabei besitzt IC 1 ein BGA-Gehäuse während IC 2 in einem DIL-Gehäuse angeordnet ist. Am Signal 1 befindet sich ein Pull-up-Widerstand, gehäuseintern umfassen die Signalpfade noch einen Widerstand (resultierend aus dem Eingangspad) und eine beim Signalwechsel umzuladende parasitäre Eingangskapazität.

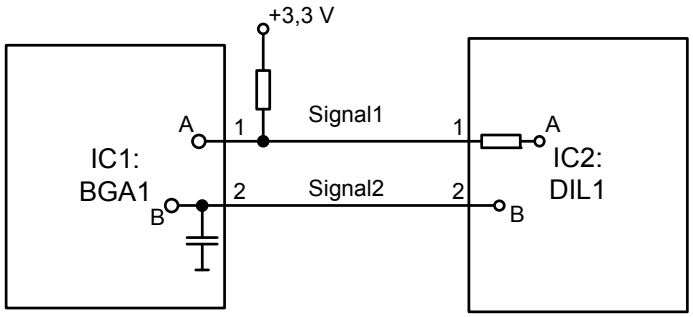


Bild 5.9.: *Beispielschaltplan für den Schaltplaneditor*

Die Umsetzung des Beispiels erfordert die Nutzung aller in diesem Abschnitt vorgestellten neuen Klassen. Die Gehäuse sind als *CircuitPackage* (BGA, DIL) umzusetzen und enthalten jeweils zwei *Pins* (1 und 2). Zu jedem Gehäuse existiert ein *IntegratedCircuit* mit je zwei *CircuitPorts* (A und B). Dabei enthält CircuitPort A bei IC 2 den Widerstand und CircuitPort B bei IC 1 die parasitäre Kapazität jeweils als *internalAssociatedComponent*. Der Pull-up-Widerstand am Signal 1 ist als *LinearTwoPort* zu realisieren, die Signale selbst sind *Connections* des Systemmodells.

5.3.4. Code-Generatoren

Die Prüfung des spezifizierten Verhaltens und der Implementierung der heterogenen Systeme erfordern die Nutzung externer Tools. Deren Eingabesyntax und -semantik unterscheidet sich meist von der internen Datenstruktur des SpecScribe, so dass hier eine Aufbereitung der Daten erforderlich ist. Die dazu entwickelten generischen Klassen zum Code-Export werden durch Vererbung an die jeweilige konkrete Sprache angepasst. Im Rahmen dieser Arbeit entstanden, neben der generischen Klasse, Exporter nach VHDL, SystemC(-AMS) und Hybrid-SAL. Die beiden letztgenannten Klassen sind ein Gegenstand des nächsten Kapitels, an dieser Stelle soll die generische Klasse kurz erläutert werden.

Die Basisklasse *CodeExporter* stellt sprachunabhängige Funktionen zur Erzeugung von simulierbarem Code bereit. Dies betrifft insbesondere Funktionen zur formatierten Textausgabe und zur Behandlung von Dateien. Des Weiteren deklariert die Klasse virtuelle Funktionen, welche von den konkreten Sprachexporteuren implementiert werden müssen.

Über die Basisklasse *CodeExporter* hinaus erstellt die Klasse *StructDictGeneration* ein Python-Dictionary mit häufig genutzten Strukturinformationen. Dies umfasst Daten zu Modultyp, Ports, Signalen, Datentypen, Variablen und Unterkomponenten.

Die Wiederverwendung dieser zielunabhängigen Strukturdatensammlung bei verschiedenen Sprachen beschleunigt den Exportprozess, da sie pro Quellkomponente nur einmal generiert werden muss. Darüber hinaus erzeugt ein Teil der Exporter Testbench-Gerüste, was die Erstellung von Simulationsläufen unterstützt und beschleunigt. Als Rückgabewert übergibt der Exporter ein *Information*-Objekt mit Übersetzungsnachrichten (*Messages*) sowie eine Erfolgs-/Misserfolgsmeldung an die aufrufende Funktion.

Für den Export der Daten in eine Modellierungssprache ergibt sich ein schematischer Ablauf wie in Bild 5.10 dargestellt.

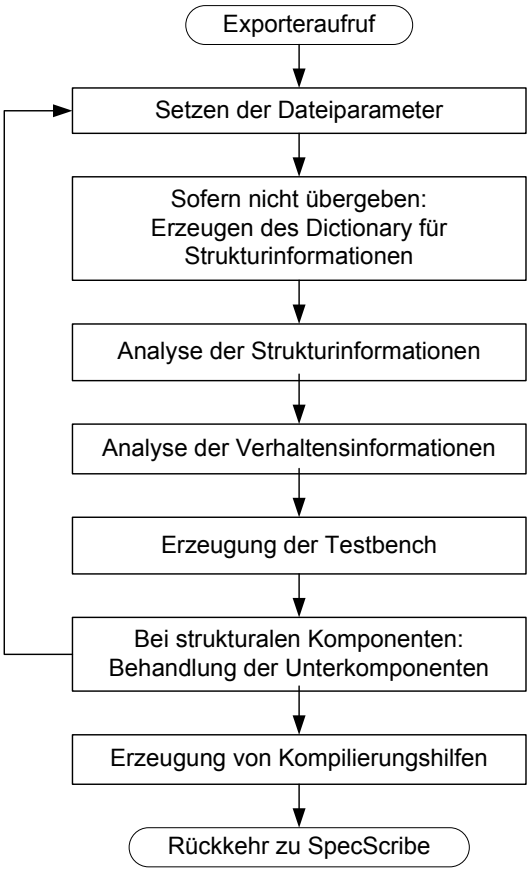


Bild 5.10.: *Ablauf des Exports der Daten in eine Modellierungssprache*

Der mit dieser Struktur erstellte Exporter nach VHDL fungiert als Hilfsmittel für das Modelchecking rein digitaler Systeme. Der automatisiert erzeugte Code bildet die Grundlage für eine Netzlistenerstellung, auf deren Basis verschiedene Modelchecker wie beispielsweise OneSpin 360MV [111] arbeiten. Der erzeugte VHDL-Code verwendet synthesefähige Konstrukte und kann damit auch zur Implementierungserstellung genutzt werden. Der interne Aufbau des VHDL-Exporters ähnelt dem in Kapitel 6.2 erläuterten SystemC(-AMS)-Exporter.

5.4. Ausblick auf weitere Konzepte

Die bisher geschilderten Konzepte bilden die Grundlage zur Spezifikationserfassung und -verarbeitung digitaler und heterogener Systeme in SpecScribe. Neben diesen befinden sich weitere Konzepte in der Umsetzungsphase. So werden die Problematiken der Kostenabbildung und Design Space Exploration auf Spezifikationsebene derzeit untersucht und anhand eines komplexen Beispiels umgesetzt [112]. Darüber hinaus laufen Arbeiten zum Anforderungsmanagement (Tracking und Tracing) und zur Spezifikation von Registereigenschaften. Die Verifikationsunterstützung auf Basis von PSL-Statements befindet sich in der Implementierungsphase, die daraus generierten Hardware-Monitorautomaten [113] sind durch die SpecScribe-Codegeneratoren flexibel in VHDL und SystemC umsetzbar.

6. Übergänge zwischen den Beschreibungsformen

Das in Kapitel 5 geschilderte Werkzeug „SpecScribe“ dient als Grundlage für den Entwurfsprozess heterogener Systeme. Zur Verifikation, Simulation und Implementierung bietet sich die Nutzung verschiedener Tools und Sprachen an. Diese auf ihr spezielles Anwendungsgebiet zugeschnittetenen Werkzeuge erlauben eine effektive Betrachtung und Lösung von Teilproblemen, welche durch das Spezifikationstool miteinander verbunden werden. Die folgenden zwei Abschnitte stellen exemplarisch zwei im Rahmen dieser Arbeit entstandene Codeerzeuger basierend auf der Spezifikation vor. Im dritten Abschnitt steht ein ebenfalls in dieser Arbeit geschaffener Codewandler von SystemC-AMS nach VHDL-AMS im Mittelpunkt, der die Verfeinerung der Modellierung unterstützt. Die Sprachwandlung ist nötig, da VHDL-AMS deutlich mächtigere Beschreibungsmöglichkeiten auf niederer Abstraktionsebene bietet als dies SystemC-AMS zum Ziel hat.

6.1. Generierung von HyTech/HybridSAL-Code

Die Entwickler der beiden Werkzeuge HyTech [63] und HybridSAL [65] haben sich zum Ziel gesetzt, Modelchecking für heterogene Systeme zu unterstützen. Während HyTech dabei direkt den angegebenen Quellcode verwendet, wandelt HybridSAL diesen zunächst in klassische Automaten um und nutzt zur eigentlichen Berechnung den digitalen Modelchecker SAL (Symbolic Analysis Laboratory) [109]. HyTech beschränkt sich auf die Prüfung linearer hybrider Automaten der Form

$$\frac{dx}{dt} = Konstante \tag{6.1}$$

wogegen HybridSAL diese Einschränkung nicht explizit setzt, jedoch durch die indirekte Nutzung eines Digitalcheckers implizit ebenfalls eine starke Einschränkung des Parameterraums erfordert. Beide Werkzeuge sind frei im Netz verfügbar.

HyTech

Der Hybridchecker Hytech beschränkt sich auf lineare hybride Automaten und arbeitet ohne Hierarchieebenen jeweils genau einen Automaten ab. Daher kann eine hierarchische Struktur nicht mit HyTech behandelt werden, der Code-Exporter muss hier entsprechende Prüfungen vornehmen und die Automaten listenartig sequentiell an HyTech übergeben. Das Zusammenspiel der Automaten ist vom Entwerfer manuell zu prüfen.

HybridSAL

Im Gegensatz zu HyTech erlaubt HybridSAL die Nutzung von Hierarchieebenen. Dabei stehen zwei Arten der Verbindung zur Verfügung: Die Operatoren || und [] ermöglichen eine synchrone bzw. asynchrone Abarbeitung der Einzelkomponenten. Die Verbindungen zwischen Komponenten erfolgen durch ein Renaming-Konstrukt, da in der Analysephase ein Design ohne Hierarchieebenen erzeugt wird.

Da eine generelle Reduktionsvorschrift für den Parameterraum nicht realisierbar ist, muss dieser vom Nutzer per Hand angepasst werden. Es empfiehlt sich, den Wertebereich der Datentypen auf ein Minimum zu begrenzen, da dessen Größe mit dem dualen Logarithmus in den zu prüfenden Parameterraum eingeht. Bei Tests des zugehörigen Digitalcheckers SAL wurde als mit aktueller Einzelplatz-Rechentechnik sinnvoll behandelbare Obergenze 150 boolesche Variablen ermittelt, was etwa fünf 32-Bit Integer-Variablen entspricht.

6.2. Export von Spezifikationsdaten nach SystemC-AMS

Der Export nach SystemC-AMS schafft ein simulierbares Modell der Spezifikation. Wie in Kapitel 4.2.3 gezeigt, unterstützt diese Sprache viele im Entwurf heterogener Systeme notwendige Konstrukte auf hoher Abstraktionsebene. Sie kann daher als umfassendes Modellierungswerkzeug auf System- und algorithmischer Ebene genutzt werden. Selbst eine Simulation auf Register-Transfer-Ebene bzw. Gleichungssystemebene ist möglich, hier wird jedoch üblicherweise ein Weg über VHDL(-AMS), wie in Abschnitt 6.3 kurz dargestellt, vorgezogen. Bild 6.1 zeigt die Spezialisierungen des Exporters gegenüber der allgemeinen Darstellung in Bild 5.10, bezogen auf eine Komponente. Dieser Ablauf wiederholt sich bei hierarchischen Designs für jede Unterkomponente.

SystemC und damit auch deren Erweiterung SystemC-AMS weisen eine strikte Trennung von Struktur- und Verhaltensinformationen auf. Die Strukturinformationen befinden sich in einer Header-Datei, während die Verhaltensinformationen in einer cpp-Datei enthalten sind. Die Headerdatei kann in mehrere Module eingebunden werden und stellt so die Strukturinformationen auch hierarchisch höheren Komponenten bereit, so dass dort eine Deklaration der Schnittstellen der Unterkomponenten wie in VHDL nicht nötig ist. Dies vereinfacht den Code-Export, da nicht hierarchieübergreifend gearbeitet werden muss.

Die Analyse des Verhaltens bei hybriden bzw. digitalen Automaten beginnt mit einer Umordnung der internen Signalzuweisungen. Dies stellt sicher, dass die höchstpriorisierte Zuweisung am Schluss ausgeführt wird und damit die niederpriorisierten Zuweisungen überdeckt. Die Modellierung der Zustandsübergänge erfolgt mittels case-Anweisung, deren Zweigen die jeweiligen Variablenzuweisungen zugeordnet sind. Für die Behandlung von *Expressions* als Zuweisungswerte kommen rekursive Funktionsaufrufe zum Einsatz.

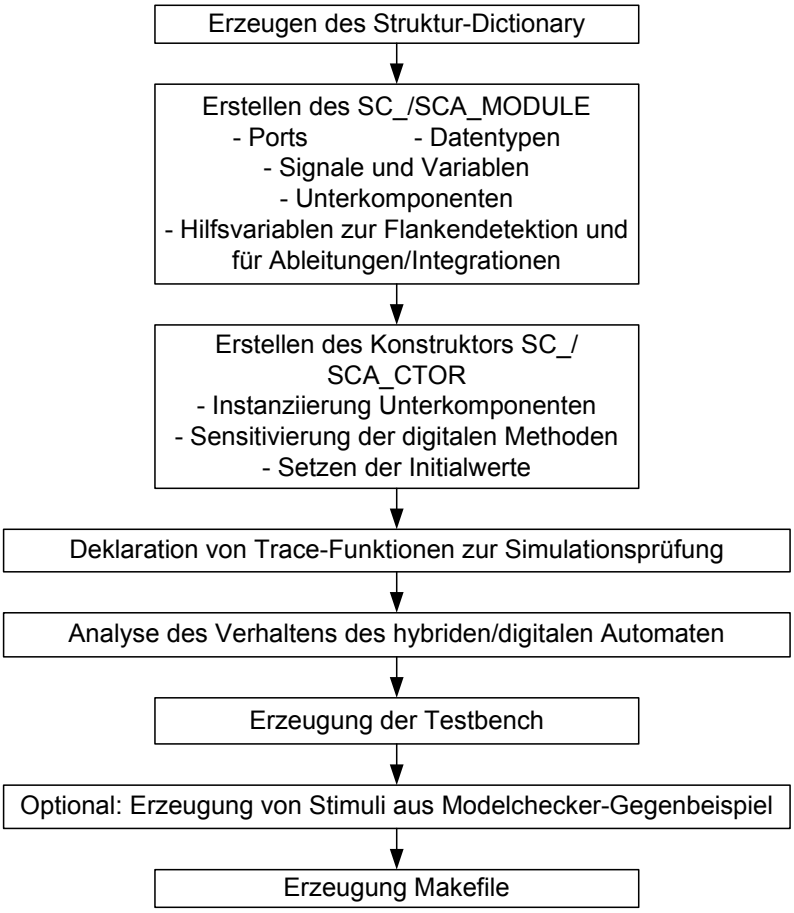


Bild 6.1.: *Ablauf des Exports der Daten aus SpecScribe nach SystemC-AMS*

Die genutzte SystemC-AMS-Version 0.15 erlaubt keine direkte Nutzung von Gleichungen, sondern basiert auf Zuweisungen. Gleichungssysteme sind in der frei verfügbaren Version nur indirekt durch Netzstrukturen abbildbar. Damit ist eine volle Unterstützung aller möglichen SpecScribe-Expressions noch nicht möglich; eine derartige Funktionalität ist in späteren Versionen von SystemC-AMS geplant.

Im Unterschied zum digitalen SystemC sind bei SystemC-AMS die Funktionsnamen für die Verhaltensbeschreibung vordefiniert. Eine Schlüsselfunktion übernimmt dabei sig_proc() als Abbildung des analogen Verhaltens zu jedem Zeitschritt. Hybride Automaten erfordern die Nutzung dieser Funktion um einen zeitkontinuierlichen Ablauf nachzubilden. Entsprechend muss die Schrittweite der Simulation möglichst gering gewählt werden - sie sollte typischerweise im Bereich von Pikosekunden liegen. Dies ist jedoch abhängig von den entstehenden Signalverläufen, bei extrem steilen Anstiegen ist die zeitliche Auflösung entsprechend zu verfeinern. Der Algorithmus des Code-Exports hybrider Automaten stellt eine Erweiterung des Algorithmus der digitalen Automaten um die notwendigen Hilfsvariablen und analogen Zuweisungen dar.

Der erfolgreiche Export nach SystemC(-AMS) von mehreren verschiedenen Beispielen zeigt die Zuverlässigkeit dieses Codegenerators. In naher Zukunft sind jedoch Anpassungsarbeiten nötig, da sich SystemC-AMS gerade in der Standardisierungsphase durch IEEE befindet und in diesem Zusammenhang größere Syntaxänderungen bevorstehen [97]. Zur Systemverifikation stellten Lämmermann et al. [29] eine auf Behauptungen (Assertions) basierende Methode für SystemC-AMS vor, die Anwendbarkeit auf MEMS wird derzeit untersucht.

6.3. Export SystemC-AMS nach VHDL-AMS

Zur Synthese von SystemC-Beschreibungen existieren einige Ansätze [114], jedoch werden diese von den Toolherstellern derzeit nicht weiter verfolgt. Daher ergibt sich die Notwendigkeit, die digitalen Bestandteile von SystemC in das synthesefähige VHDL zu konvertieren. Dies wurde von Kühn [115] im Rahmen einer Studienarbeit umgesetzt, jedoch traten bei der Konvertierung noch Fehler auf. Diese Arbeit bildete die Grundlage für eine korrigierte Fassung, die gleichzeitig um die SystemC-AMS- und VHDL-AMS-Syntax erweitert wurde [116]. Damit lässt sich die SystemC-AMS-Beschreibung von Komponenten automatisiert nach VHDL-AMS übersetzen und so die manuelle Übertragungsarbeit beim Sprachwechsel vermeiden. VHDL-AMS bietet durch implizit angebbare Differentialgleichungen die Möglichkeit, einfacher detailliertere Modelle zu erstellen. Dies wird aber durch eine gegenüber SystemC-AMS deutlich verlängerte Simulationszeit (bis zu Faktor 60 beim Modell eines einzelnen Beschleunigungssensors nach [117]) erkauft.

6.3.1. VHDL-AMS-Konstrukte für SystemC-AMS-Elemente

Der Modellaufbau von SystemC-AMS-Modulen wurde in Abschnitt 4.2.3 erläutert. Daraus lassen sich die folgenden Konzepte für den Konverter ableiten:

Grundkonzept

Wie SystemC-AMS ist auch VHDL-AMS bzw. sein digitaler Ursprung VHDL eine sehr strukturierte Sprache. Die Trennung zwischen den Schnittstellen einer Komponente und seiner Beschreibung erfolgt durch das Entity/Architecture-Konzept noch strenger als in SystemC-AMS. Alle Schnittstellen des Moduls nach außen (wie Ports und Parameter) sind in die Entity aufzunehmen, wogegen die innere Struktur und das Verhalten in der Architecture abzulegen sind. Verbindungen und Variablen dürfen nur in abgetrennten Deklarationsteilen eingeführt werden. Im Gegensatz dazu erlaubt SystemC, Variablen an fast allen Stellen im Sourcecode zu definieren. Vor dem Export nach VHDL-AMS muss also eine Sammlung der verteilten C++-Deklarationen erfolgen. SystemC-Modulvariablen sind von allen Methoden des Moduls aus schreib- und lesbar. VHDL'93 führt eine derartige Funktionalität unter dem Schlüsselwort *shared variable* in den VHDL-Standard ein.

Für die Abbildung digitalen Verhaltens kommen in SystemC *Methoden* zum Einsatz. In Analogie dazu bietet VHDL das Konzept der *Prozesse* an. Die im SystemC-Modulkonstruktor erfolgte Sensitivierung auf Eingangssignale muss dazu in die sog. sensitivity-list des Prozesses transformiert werden. Ein Problem stellen strukturelle Module dar: In SystemC enthält die Einbindung einer Subkomponente keine Informationen bezüglich der Porttypen, sondern lediglich die Verbindung mit lokalen Signalen. Die Typinformationen müssen aus dem Ursprungsmodul gewonnen werden. Damit ist innerhalb einer Modulkonvertierung die zugrunde liegende Komponente zu wechseln.

Analoge Modellierung

Der Analogbereich von SystemC-AMS ist gekennzeichnet durch die vordeklarierten Funktionsaufrufe aus Tabelle 4.1 und vordefinierte Bauelemente. Im VHDL-AMS-Standard existieren jedoch keine vorgefertigten Bauelemente. Die in SystemC-AMS vordefinierten linearen Elemente sind also als VHDL-AMS-Bibliothekselemente zu hinterlegen und entsprechend hierarchisch einzubinden. Dazu kann derselbe Algorithmus wie im Digitalbereich genutzt werden.

Im Gegensatz zum zuweisungsorientierten SystemC-AMS basiert VHDL-AMS auf sog. *simultaneous statements*. Diese Gleichungen werden vom Simulator zu einem Gleichungssystem zusammengefasst und gemeinsam gelöst. Dabei existiert keine Unterscheidung zwischen linker und rechter Gleichungsseite, es handelt sich also nicht um eine Zuweisung. Der Inhalt der in Tabelle 4.1 angegebenen Funktionen kann demzufolge nicht direkt in das VHDL-AMS-Gleichungssystem integriert werden. Als Hilfsmittel stellt der VHDL-AMS-Sprachstandard *simultaneous procedurals* zur Verfügung, deren Inhalt sequentiell abgearbeitet wird, die aber auch Zuweisungen für analoge Ausgänge zulassen. Digitale Signalzuweisungen sind allerdings nicht erlaubt.

Die Ablaufsteuerung kann durch Nutzung der VHDL-AMS-Information *domain* erfolgen. Hier werden drei Werte unterschieden:

- **quiescent_domain** für die Elaborationsphase (entspricht attributes() und init() aus Tabelle 4.1),
- **time_domain** für Transientensimulationen (entspricht sig_proc()) und
- **frequency_domain** für Frequenzbereichssimulationen (entspricht ac_sig_proc()).

Durch Integration der procedurals in simultaneous-if-use-statements kann damit je nach Simulationszustand die zugehörige Funktionalität genutzt werden.

6.3.2. Aufbau des Konverters

Der Konverter hat die Aufgabe, den SystemC-AMS Quelltext auf Syntaxfehler zu prüfen, in eine Metastruktur umzusetzen, die Elemente umzuordnen und anschließend

in VHDL-AMS Syntax zu exportieren. Um im Konverter größtmögliche Flexibilität zu gewährleisten, wurde eine zweistufige Umsetzung implementiert. Ähnlich wie in [32] erstellt der Konverter in der ersten Stufe *SVTOOLS* ein XML-Zwischenformat. Dieses dient der zweiten Stufe *SC2VHDL* als Ausgangsmaterial zur Umordnung des Codes und zur Wandlung nach VHDL-AMS. Zum Schreiben des VHDLAMS- Codes wird auf einen bereits vorhandenen VHDL-Parser *VHDL reverse analyzer* zurückgegriffen, dieser erweitert und in Umkehrrichtung betrieben. Vorteil dieser zweistufigen Lösung ist das flexible Umschalten auf andere Exportsprachen. Die XML-Repräsentation bleibt gleich, lediglich die zweite Stufe muss entsprechend angepasst werden. Geplant sind hier Exporte nach Spice und zu FEM-Tools. Bild 6.2 zeigt den Aufbau als Blockschaltbild. Im Folgenden stehen die beiden Konverterstufen SVTOOLS und SC2VHDL im Mittelpunkt.

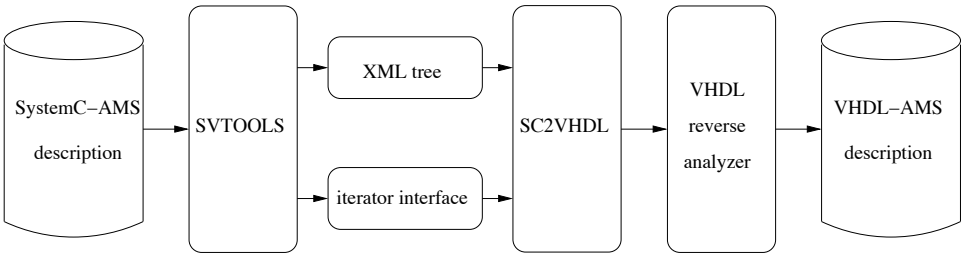


Bild 6.2.: *Toolchain für die Konvertierung SystemC-AMS – VHDL-AMS (nach [115])*

SystemC-AMS nach XML Konverter

Die Grammatik von SystemC-AMS folgt der Extended-Backus-Naur-Form (EBNF). Damit bietet es sich an, einen Parser-Generator zu verwenden, der diese Grammatikart direkt unterstützt. Im Projekt kommt PCCTS [118] bzw. dessen Nachfolger ANTLR [119] zum Einsatz. Im Generator wird eine Anzahl von Schlüsselwörtern/-zeichen (Token) definiert. Bei Auffinden dieser kommen entsprechende Regeln zur Anwendung. Hier wird eine XML-Notation des Ursprungscodes erzeugt, prinzipiell ist aber auch eine direkte Verarbeitung, z. B. zu arithmetischen Zwecken, möglich. Die Definition des Parsers ist zweigeteilt. ANTLR umfasst Regeln für das Einlesen des Ursprungscodes und definiert Schlüsselbegriffe. Es entsteht eine Baumstruktur des Codes, welche der sog. Sorcerer verarbeitet und in eine XML-Notation übersetzt. Die Notation enthält außer Kommentaren alle Elemente des Ursprungscodes.

XML nach VHDL-AMS Konverter

Ungleich schwieriger als Teil 1 ist der zweite Teil des Konverters zu implementieren. Da die Schlüsselworte von SystemC-AMS nicht direkt nach VHDL-AMS übersetzbar sind, erfolgt eine Code-Umordnung. Signale, externe Schnittstellen, Variablen und hierarchische Anordnungen müssen extrahiert und teilweise in andere Codebereiche umgesetzt

werden. Dazu dienen definierte Strukturen, die mit Hilfe vom STL-Maps und Smart Pointern verwaltet werden.

Bild 6.3 zeigt den schematischen Aufbau der zweiten Konverterstufe. Die XML-Beschreibung wird vom *converter reader* ausgelesen. Als zentrale Kontrollinstanzen fungieren Regeln, die für jeden zu übersetzenden Teilbereich (wie z. B. Modulkopf oder Signaldeklaration) aus einer Basisklasse rule_base abgeleitet werden. Zu jeder Regel existiert ein Datensammler (*collector*) und ein Übersetzer (*translator*). Die zu konvertierenden Daten werden von den Sammlern in Strukturen abgelegt, welche im *converter info manager* verwaltet werden. Diese Strukturen orientieren sich am VHDL-AMS Ausgabeformat. Die Übersetzer laden die Strukturen und bringen sie per *converter writer* in ein für den *VHDL reverse analyzer* lesbares Baumformat. Der Sorcerer-Teil (*Treewalker*) des erweiterten *VHDL reverse analyzers* verarbeitet diese Baumbeschreibung und exportiert den VHDL-AMS Quelltext. Die Instanzen der Regeln, Sammler und Übersetzer werden vom *converter object manager* mit Hilfe von Vektoren verwaltet. Zusätzlich existiert ein globales statisches Modul für Hilfsfunktionen.

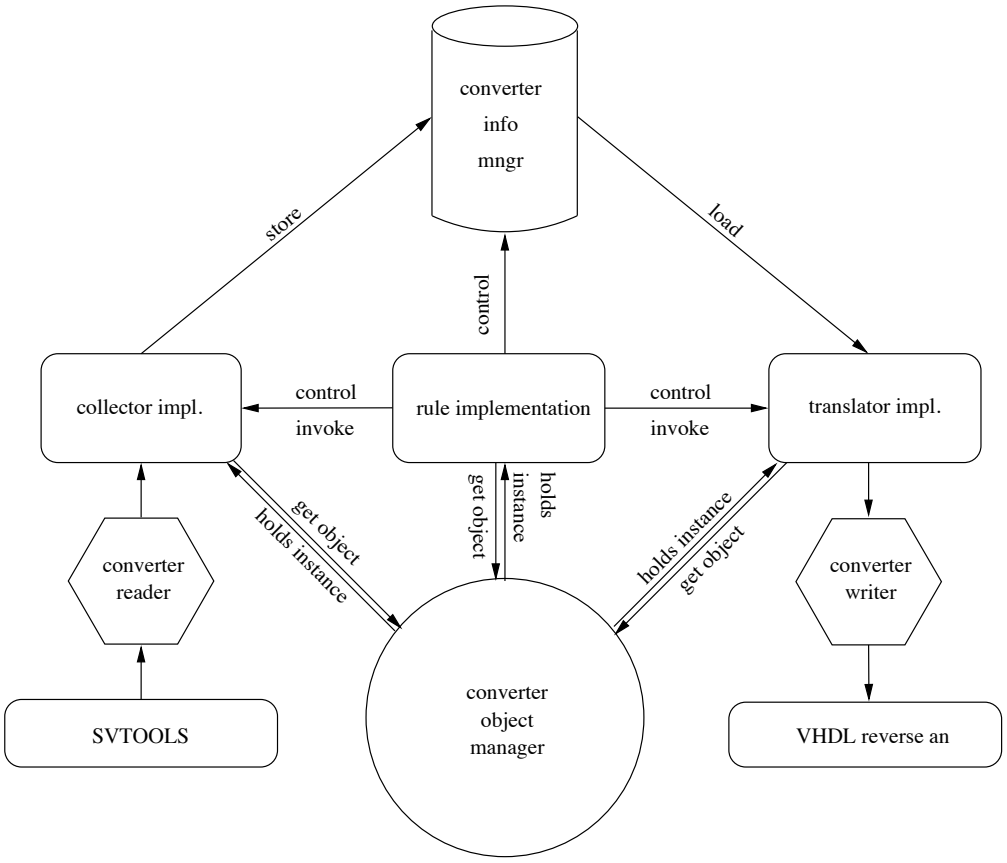


Bild 6.3.: *Aufbau des SC2VHDL-Konverter [115]*

Neben der Umordnung der Quelltexte und dem Ersetzen von Schlüsselworten sind die Datentypen umzuwandeln. Nicht jeder SystemC-AMS-Datentyp besitzt eine direkte VHDL-AMS-Entsprechung, so dass indirekte Typumwandlungen notwendig sind

(z. B. Festkommatypen wie *sc_fixed* zum Gleitkommatyp *real*). Bei Datentypen, die besondere Bibliotheken erfordern, werden automatisch die entsprechenden Header eingefügt. Spezielle Strukturen nehmen die Inhalte von Methoden und Subkomponenten auf. Da in SystemC-AMS Komponentenaufrufe per Instanziierung erfolgen, muss für das Component/Instance-Paar in VHDL-AMS die Definition der Subkomponente berücksichtigt werden. Dazu wird der Fokus des Übersetzers temporär vom aktuellen Modul auf die XML-Beschreibung der Subkomponente umgeleitet und die entsprechenden Informationen zu Ports und Parametern extrahiert.

6.3.3. Anpassungen an reale Simulatoren

Die verfügbaren VHDL-AMS Simulatoren folgen nicht in vollem Umfang dem IEEE-Standard. So werden die in 6.3.1 aufgeführtern *shared variables* und *simultaneous procedurals* nicht von allen Simulatoren unterstützt. Zur Gewährleistung der Universalität des Konvertierungsergebnisses sind diese Konstrukte zu vermeiden. Die folgenden Abschnitte zeigen kurz die verwendeten Ersatzkonstrukte.

Simultaneous procedurals

Simultaneous procedurals sind zeitkontinuierliche Äquivalente zu VHDL-Prozessen. Der Inhalt wird zu jedem Zeitschritt sequentiell abgearbeitet. Es dürfen Variablen und Quantities (analoge Verbindungen) zugewiesen werden, jedoch keine digitalen Signale. Um eine solche Funktionalität mit Alternativkonstrukten nachzubilden, ist die Nutzung eines VHDL-Prozesses mit zyklischem Aufruf (*wait for simulation_stepwidth*) notwendig. Hier sind jedoch nur Signalzuweisungen erlaubt, Quantities müssen separat geschrieben werden. Die Synchronisation der analogen und digitalen Zeitachse erfordert entsprechende BREAK-Statements. Die SystemC-AMS Basisfunktionen (siehe Tabelle 4.1) sind zur besseren Segmentierung des Codes als Prozeduren ausgeführt. Die Zykluszeit des Prozesses bestimmt sich aus dem Port-Parameter set_T() im SystemC-AMS-Quelltext.

Shared variables

Die ebenfalls nicht allgemein unterstützten globalen Variablen passen sich bei analogen Modulen in das Schema des vorherigen Abschnittes ein. Für rein digitale Module ist die Nutzung eines VHDL-Simulators vorzuziehen, der *shared variables* direkt unterstützt (z. B. Modelsim). Im Analogen sind die Globalvariablen als Signale zu deklarieren. Sie werden in jeder Prozedur zu Variablen untersetzt, d. h. zu Prozedurbeginn gelesen und am Abschluss geschrieben. Dies ist möglich, da im analogen Bereich immer nur eine der Prozeduren aus Tabelle 4.1 aktiv sein kann.

Listing 6.1 zeigt den aus diesen Einschränkungen resultierenden VHDL-AMS Pseudocode für SDF-Analogmodule, in dem je nach aktuellem Simulationsstatus (Elaboration, Frequenz- oder Zeitbereichssimulation) die entsprechend umgewandelten SystemC-AMS-Funktionen gerufen werden.

```
ARCHITECTURE
 -- Deklariere Signale zu allen Quantities und globalen
    Variablen
BEGIN
  scams_handler:PROCESS
  BEGIN
    LOOP
    -- Lese alle In-Quantities zu Signalen
    IF domain=quiescent_domain THEN
      attributes(Signale);
      init(Signale);
      BREAK;
    ELSIF domain=frequency_domain THEN
      ac_sig_proc(Signale);
      BREAK;
    END IF;
    IF domain /= frequency_domain THEN --time_domain & op-point
      WAIT FOR 0.0 ns; -- Signal update
      sig_proc(Signale); BREAK;
      WAIT FOR simulation_stepwidth;
    END IF;
    END LOOP;
  END PROCESS;
  -- Schreibe alle Out-Quantities aus Signalen
END;
```

Listing 6.1: *Pseudocode für konvertierte SDF-Module*

Damit ist es möglich, mit dem Konverter

- digitale Beschreibungen auf Register-Transfer-Ebene,
- analoge hierachische Beschreibungen,
- Datenflussmodule inkl. anwenderspezifischer arithmetischer Funktionen im Zeitbereich (außer Laplace-Transferfunktionen) und
- in SystemC-AMS vordefinierte lineare Bauelemente

syntaktisch von SystemC(-AMS) nach VHDL(-AMS) zu übertragen.

Nicht unterstützte Konstrukte

Bei einigen, in Tabelle 6.1 dargestellten SystemC-AMS Konstrukten mangelt es an Entsprechungen in VHDL-AMS. Dies betrifft insbesondere aus abstrakt modellierten Schnittstellen (integrierte FIFO-Speicher) resultierende Sprachelemente, die nur durch Integration expliziter FIFOs in VHDL-AMS nutzbar wären. Im Rahmen der Mikrosystembeschreibung in dieser Arbeit wurden derartige Konstrukte nicht benötigt.

SystemC-AMS Element	**Erläuterung**
sca_ltf_nd	Nicht in Prozessen verwendbar
set_T(sc_time)	set_T(double, sc_time_unit) nutzen
set_rate(), get_rate(), get_sample_cnt()	keine Default-FIFOs in VHDL-AMS
get_t0()	keine sinnvolle Entsprechung in VHDL-AMS

Tabelle 6.1.: *Noch nicht unterstützte Sprachelemente von SystemC-AMS in VHDL-AMS*

Die Syntaxumstellung von SystemC-AMS bei der Version 1.0 wird einen Teil dieser Problemfälle umgehen. Die damit einhergehende Umstellung des Konverters wird für eine weitere Überarbeitung genutzt und soll die Zahl der nicht unterstützten Konstrukte verringern.

7. Systembewertung auf Basis von Kostenparametern

Mit der fortschreitenden Entwicklung von Entwurfswerkzeugen und Bibliotheken für den Mikrosystementwurf entsteht zunehmend die Notwendigkeit, Mikrosysteme nicht nur hinsichtlich funktioneller Parameter, sondern auch hinsichtlich von Kostenfaktoren im Systementwurf zu optimieren [120, 121], da diese Parameter neben den funktionalen Daten wichtige Kenngrößen eines Systems darstellen. Dabei sind unter dem Begriff Kosten nicht nur fiskalische Kosten, sondern im Sinne der Terminologie der mathematischen Optimierung eine Repräsentation einer zu optimierenden Zielfunktion zu verstehen. Mit dieser Zielfunktion sollen Größen wie z. B. Latenz, Datenrate, Leistungsaufnahme, Chipfläche, Fehlertoleranz, Testbarkeit, Speicherverbrauch, Herstellungskosten u. a. gewichtet bewertet werden. Bei der Systemkomposition sind diese Werte einerseits auf das Einhalten einer vorgegebenen Grenze zu prüfen (z. B. maximal zur Verfügung stehende Chipfläche) und andererseits zu einem globalen Gütemaß zu verknüpfen. Eine Optimierung dieses globalen Gütemaßes führt zu einer Verbesserung der Systemimplementierung.

Die Bewertung von Realisierungsvarianten findet meist in Zusammenhang mit der Partitionierung des abstrakt beschriebenen Systems in Hardware- und Softwareteile statt. Zur Verknüpfung von Kostenparametern entwickelte Ragan das Werkzeug „Ghost“ [122], dass die Daten von kommerziellen Kostenabschätzungswerkzeugen einlesen kann und eine Partitionierung in Hardware- und Softwareanteile vorschlägt. Einen in C implementierten Ansatz stellte Sangeetha [123] vor, der sich ausschließlich mit Hardware befasst und auf Register-Transfer-Ebene arbeitet. Einen ähnlichen Ansatz erarbeitete Kim [124], der basierend auf einer Systembeschreibung in C als Verschaltungsmodell von „Functional Units“ den Aufwand für die Umsetzung in Hardware bestimmt. Im Gegensatz dazu ermittelt Zhao [125] die Softwarekosten eines Systems basierend auf Komplexitätsabbildungen.

Um die Datenkonsistenz zwischen Modell und Kostenwerten sicherzustellen, bietet es sich im Gegensatz zu den bisher vorgestellten Ansätzen an, die Kosten in einem simulationsfähigen Werkzeug mit einzubeziehen. Dafür stehen zahlreiche Systembeschreibungssprachen wie z. B. SystemVerilog, VHDL/VHDL-AMS und SystemC/SystemC-AMS zur Verfügung. Eine auf additive Verknüpfungen beschränkte Implementierung von hierarchischen Kostenbeschreibungen mit VHDL präsentierte Schlegel [126]. Eine Nutzung dieser Lösung mit VHDL-AMS scheiterte an der mangelnden Unterstützung der IEEE-Sprachkonstrukte durch kommerzielle Simulatoren.

Da die Fähigkeiten von VHDL-AMS zur Systemsimulation eingeschränkt sind, bietet sich für die Abbildung der Kosten die schon für die Systemmodellierung genutzte höherabstrakte Sprache SystemC-AMS an. Der Ansatz aus [126] wurde daher im Rahmen dieser Arbeit nach SystemC-AMS überführt und um grundlegende Konstrukte erweitert [127, 128]. Nach einer kurzen allgemeinen Betrachtung der Kostenparameter in der Systembewertung zeigen die darauf folgenden Abschnitte die Funktionalität des neuen Ansatzes kurz auf. Ein abstraktes Beispiel erläutert die Implementierung, die Anwendung auf ein praktisches Beispiel erfolgt im Kapitel 9.

7.1. Auswahl der notwendigen Kostenparameter

In das Ergebnis des Designprozesses fließen unzählige Entscheidungen ein, damit existieren meist mehrere Implementierungsvarianten. Die Auswirkungen der Entscheidungen auf das zu erstellende System sind vor der Realisierung abzuschätzen um eine optimale Umsetzung der Spezifikation zu erhalten. Dabei hängen die zu berücksichtigenden Parameter stark vom Systemzweck ab. Dennoch lassen sich einige generelle Aussagen finden, welche für die meisten Systeme gelten und die sich so für eine Berücksichtigung im allgemeinen Entwurfsfluss anbieten.

Für alle Entwürfe, egal ob aus dem Digital- oder aus dem Analogbereich, haben unter anderem die folgenden Parameter Einfluss auf die Kosten bzw. den Erfolg der Implementierung:

- die für den Entwurf nötige Arbeitszeit,
- die Kosten der zu verwendenden Werkzeuge, Fertigungsprozesse und Bauteile,
- die abzuführende Verlustleistung und damit einhergehend der zulässige Temperaturbereich,
- der Silizium-Flächenbedarf als Chip bei integriertem Aufbau bzw.
- der Raumbedarf als Leiterplattensystem bei diskreten Realisierungen,
- den verwendeten Gehäusetypen der Bauteile und damit verbunden die Anzahl und Lage der Pins sowie der Bestückungsaufwand und
- der Aufwand für den Test.

Der Digitalbereich mit seiner hohen Integrationsdichte erfordert die Betrachtung einiger besonderer Kostenparameter wie der Taktfrequenz, der Gesamtdauer der Verarbeitung der Daten, dem Speicherbedarf sowie der nötigen Anzahl an kombinatorischen Ressourcen.

Im Analogbereich trifft man zwar meist Strukturen geringerer Integrationsdichte an, dafür sind diese Schaltungen bzw. Strukturen deutlich empfindlicher gegenüber Abweichungen in der Realisierung. Insbesondere ergeben sich Kostenparameter aus den Bereichen der Störsicherheit, der notwendigen Schutzschaltungen, dem zu unterstützenden Frequenzbereich bzw. der notwendigen Bandbreite und dem Arbeitstemperaturbereich.

7.2. Abbildung der Kostenparameter im Simulator

Die Berücksichtigung von Kostenparametern im Designprozess erfordert Überlegungen zur Bestimmung dieser Parameter und zu Algorithmen der Verknüpfung der Komponentenparameter.

7.2.1. Bestimmung von Kostenparametern

Die Bestimmung von Kostenwerten auf hoher Abstraktionsebene, insbesondere mit Ziel der kritischsten Parameter Fläche und Leistungsverbrauch, war Gegenstand mehrerer Arbeiten in Deutschland [129], aber auch international [130, 131]. Generell lassen sich drei Arten als Quellen von Parametern für den Designer erkennen:

- Statische Bestimmung aus Erfahrungswerten,
- Nutzung von Bibliothekselementen mit vorhandenem Kostensatz,
- Simulative Bestimmung.

Die statische Bestimmung von Kostenparametern erfordert ein hohes Maß an Erfahrung und ist damit sehr fehleranfällig. Jedoch ist diese Methode der einzige Algorithmus, der bei von Grund auf neu zu entwerfenden Systemen zur Verfügung steht. Da die später verwendete Technologie in diesem Entwurfsschritt meist noch nicht feststeht, sind in der Regel nur qualitative, vergleichende Aussagen möglich.

Kann der Entwickler hingegen auf bereits vorhandene Systeme aufbauen, erhöht sich die Qualität dieser Abschätzungen erheblich. Im Idealfall können sogar Teilkomponenten wiederverwendet werden, so dass die notwendigen Parameter entweder bereits vorliegen oder anhand des vorhandenen Systems bestimmbar sind. Dieselben Aussagen gelten bei der Nutzung von IP-Komponenten von Drittanbietern.

Die Erstellung des simulierbaren Systemmodells ermöglicht die simulative Bestimmung von Kostenparametern. Mit der weiteren Verfeinerung der Teilsysteme lassen sich diese Parameter präzisieren und können in folgenden Entwurfsprojekten als Bibliothekselemente dienen. Voraussetzung für die Ermittelbarkeit ist die Nutzung eines geeigneten Simulators, wobei auch Spezialsimulatoren für einzelne Kostenbereiche existieren (z. B. PowerMill [132] für den Verlustleistungsbereich).

7.2.2. Verknüpfung der Kostenparameter

Die Realisierung der Kostenmodellierung in SystemC/SystemC-AMS, wie in [128] dargestellt, bietet gegenüber der Implementierung mit VHDL/VHDL-AMS mehrere Vorteile. In SystemC-AMS, der analogen Erweiterung von SystemC, besteht im Gegensatz zu VHDL-AMS keine Notwendigkeit, Ports als Verbindungselemente zu nutzen. Stattdessen kommt eine Listenstruktur zur Anwendung. Dies erlaubt die Nutzung des

C++-Pointerkonzepts unabhängig vom Modulinterface. Somit können die Kostenwerte auch während der Laufzeit noch geändert werden, was eine dynamische Bestimmung von Kostenparametern (z. B. Stromaufnahme) ermöglicht. Außerdem ist es mit vorhandenen C++-Compilern möglich, automatisiert mehrere Implementierungsvarianten zu untersuchen. Die nächsten Abschnitte erläutern kurz die verwendete Datenstruktur sowie die zur Verfügung stehenden Funktionen.

Datenstruktur

Die Kostenwerte jeder Komponente werden in einer eigenen, durch Membervariablen einer Instanz der neuen Klasse cost repräsentierten, Struktur abgelegt. Um von einem Element des Designs aus das nächste zu erreichen, bietet sich eine verzweigte Listenstruktur an. Dazu enthält jede Komponente einen Zeiger auf das nächste Element auf gleicher Hierarchieebene (pNext) und einen Zeiger auf das erste Element der nächsttieferene Hierarchieebene (vNext). Zusätzlich kann die Komponente mit einer ID versehen werden. Sie enthält außerdem einen Zeiger auf sich selbst, mit der beim Parsen des Kostenbaumes der systemweit eindeutige Instanzname ausgegeben werden kann. Ein optionaler Kostengrenzvektor dient zum Festlegen der maximal von der Komponente und deren Unterkomponenten verbrauchbaren Ressourcen. Ein Flag limit_exceeded weist auf eine Kostengrenzenüberschreitung hin. Damit ergibt sich eine Kostenstruktur wie in Bild 7.1 dargestellt.

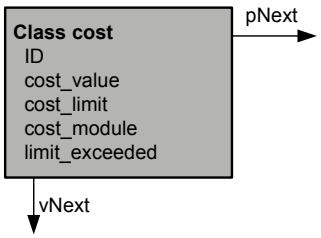


Bild 7.1.: *Kostenstruktur*

Die Kostenwerte einer Komponente werden in einer Struktur als Instanz der neuen Klasse cost abgelegt, wobei eine verkettete Listenstruktur das Erreichen benachbarter Module ermöglicht. Im Design entsteht damit ein sog. Kostenbaum (ein Ausschnitt ist in Bild 7.2 dargestellt). Von dessen oberster Hierarchieebene können alle Einzelelemente durch Zeigeroperationen angesprochen werden.

Ein Vektor umfasst alle Einzelkosten einer Komponente, ein weiterer Vektor gibt optional die maximal erlaubten Kosten der Komponente an, wobei der Wert -1 die Prüfung dieses Wertes in dieser Hierarchieebene abschaltet. Die Bedeutung des Werts erschließt sich aus seiner Position. Tabelle 7.1 enthält die Elemente der neuen Klasse cost.

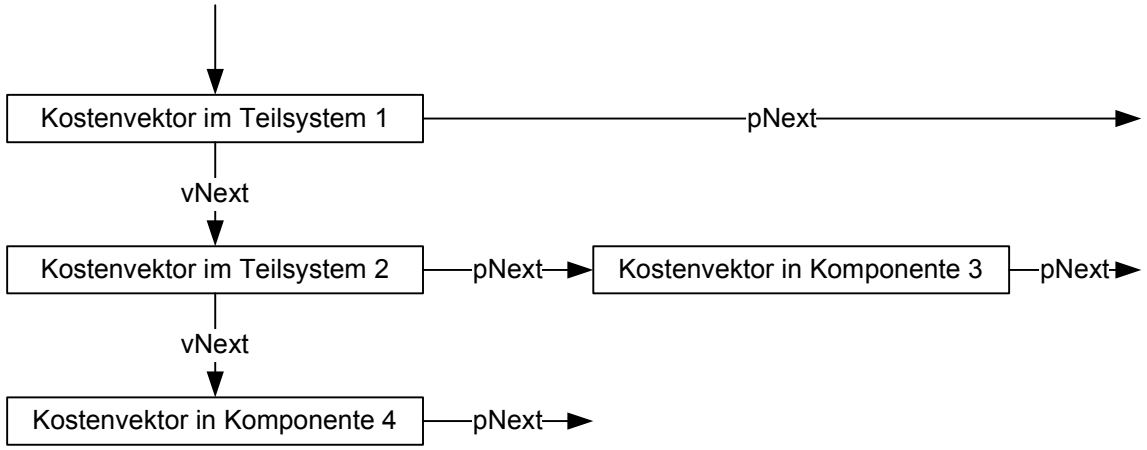


Bild 7.2.: *Teil eines Kostenbaums*

Element	Datentyp	Beschreibung
ID	char[128]	Name
cost_value	double[]	Kostenwerte
cost_limit	double[]	Kostengrenzen
cost_module	sc_module*	Zeiger zum zugehörigen SystemC(-AMS)-Modul
limit_exceeded	bool[]	Flags für Kostenüberschreitungen
pNext	cost*	Zeiger auf das nächste horizontale Element
vNext	cost*	Zeiger auf das nächste vertikale Element)

Tabelle 7.1.: *Elemente der Klasse cost*

Funktionen zur Kostenmodellierung

Zur Erstellung und Analyse des Kostenbaumes muss die Klasse cost um Funktionen erweitert werden. Die Aufrufe zum Abarbeiten des Kostenbaumes basieren soweit wie möglich auf Rekursion. Die Klasse cost besitzt die in Tabelle 7.2 angegebenen Funktionen zum Aufbau des Kostenbaumes und zur Berechnung der Systemkosten.

Funktion	Beschreibung
add(cost* a)	Fügt ein Modul auf nächstniedriger Hierarchieebene ein
set_val(double[])	Definition der Kostenwerte eines Moduls
set_limit(double[])	Definition der Kostengrenzen eines Moduls
del()	Löschen des Kostenbaumes
list_print()	Strukturierte Ausgabe der Kostenwerte aller Elemente
check_system()	Prüft das System auf Kostengrenzenüberschreitungen
sum()	Berechnet die Kostenwerte über Hierarchieebenen
scale(double[])	Berechnet das globale Gütemaß

Tabelle 7.2.: *Funktionen der Klasse cost*

Für die Berechnung der Kostenwerte wurden drei neue Operatoren eingeführt. Der schon in der VHDL-Version vorhandene Operator ADD summiert die Kostenwerte der Unterkomponenten. Dies ist in den meisten Fällen ausreichend, z. B. für Chipflächenverbrauch oder Leistungsaufnahme. Der MAX-Operator bestimmt das Maximum der Kosten (z. B. für die minimale Betriebstemperatur oder die minimale Taktzeit in einer Pipeline), als Gegenstück fungiert der MIN-Operator. Der vierte Operator MULT multipliziert die Kostenwerte, z. B. für die direkte Berechnung von Verstärkungen oder Zuverlässigkeitsberechnungen. Die Operatoren können für jeden Kostentyp (jedes Vektorelement) unabhängig festgelegt werden.

Die Überwachung der Kostengrenzen erfordert eine Fallunterscheidung: Einige Werte wie Chipflächenverbrauch müssen die Grenze unterschreiten, während andere die Grenze überschreiten sollen (z. B. minimale Taktfrequenz). Darum stehen mit COMP_MAX und COMP_MIN zwei neue Vergleichsoperatoren zur Verfügung. Das globale Gütemaß eines Systems wird mit dem Befehl scale() bestimmt. Dieser berechnet eine gewichtete Summe der Kostenwerte des Top-level-Moduls bezogen auf die Kostengrenzen. Das Vorzeichen der Gewichte bestimmt das Modul aufgrund des Grenzoperators.

Abstraktes Beispiel

Um die gesamte Funktionalität der erweiterten Kostenmodellierung zu zeigen, wird im folgenden das in Bild 7.3 dargestellte theoretische System mit dieser Methodik untersucht. Es umfasst 5 Hierarchiestufen, für die die in Tabelle 7.3 angegebenen zwei Realisierungsvarianten A und B möglich sind. Dabei stellen die Kostenvektoren für Subsysteme den zusätzlichen Aufwand beispielsweise für die Verdrahtung dar. Die gewählten Kostenparameter (Fläche, minimale und maximale Betriebstemperatur, Zuverlässigkeit) erfordern dabei die Nutzung verschiedener Verknüpfungsvarianten.

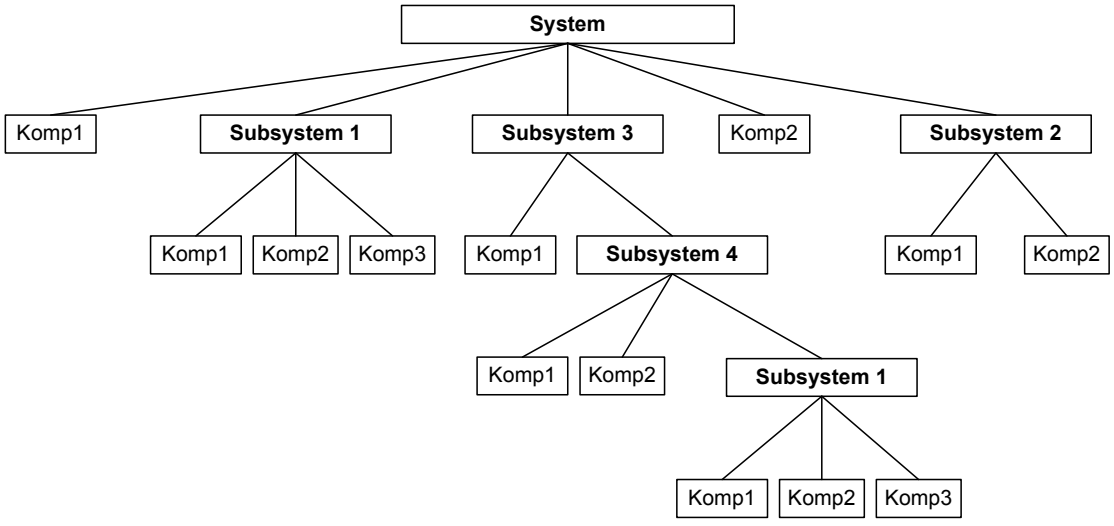


Bild 7.3.: *Überblick über das abstrakte Beispielsystem*

Listing 7.1 zeigt einen Teil der SystemC-Moduldefinition für das Subsystem 3, Listing 7.2 die Ausgabe eines Simulationslaufes für den Zweig des Subsystems 3. Der Gewichtsvektor (1, 0, 0, 1.5) verhindert den Einfluss der Temperaturangaben auf das globale Gütemaß und betont die Zuverlässigkeit im Verhältnis zum Flächenverbrauch. Die Gesamtkosten des Systems erhalten ein negatives Vorzeichen, resultierend aus der Nutzung des COMP_MIN-Vergleichsoperators.

```
SC_MODULE(teilsys3) {
  cost* cost_vector, *teilsys3_selbst;
    komponente1 * komp1_1;
    teilsys4 * teilsys4_1;

    SC_CTOR( teilsys3 ) {
      #ifdef COST_USE_A
        double cost_val[COST_WIDTH]={5,0,80,0.999}; // A
      #else
        double cost_val[COST_WIDTH]={3,0,80,0.995}; // B
      #endif
    double cost_limit[COST_WIDTH]={1000,-1,-1,-1};
    komp1_1 = new komponente1("comp1_1");
    teilsys4_1 = new teilsys4("subsys4_1");
    teilsys3_selbst = new cost("subs3_self");
      teilsys3_selbst->set_val(cost_val);
      teilsys3_selbst->cost_module=this;
    cost_vector = new cost("subsys3");
      cost_vector->cost_module=this;
      cost_vector->set_limit(cost_limit);
      cost_vector->add(komp1_1->cost_vector);
      cost_vector->add(teilsys4_1->cost_vector);
      cost_vector->add(teilsys3_selbst);
    }
};
```

Listing 7.1: *SystemC-Moduldefinition für Subsystem 3*

Die Realisierungsvariante B erfüllt nicht die gestellten Anforderungen zur maximalen Betriebstemperatur und zur Zuverlässigkeit. Die verfehlten Anforderungen werden dabei als INFO-Zeilen ausgegeben und erhalten in der Systemübersicht ein Doppelkreuz. In diesem Beispiel muss der Realisierungsvariante A der Vorzug gegeben werden, obwohl sie ein schlechteres globales Gütemaß besitzt.

Die Einrückungen der Modulinformationen entsprechen in der Bildschirmausgabe den Hierarchieebenen des Systems. Zur Information ist neben dem berechneten Kostenvektor auch der Kostengrenzvektor angegeben.

Element	Realisierungsvariante	Fläche pro Element	minimale Betriebstemperatur	maximale Betriebstemperatur	Zuverlässigkeit
System (top-level)	A	1	0	80	1.0
	B	1	0	80	1.0
	limit	2000	0	80	0.950
Subsysteme 1 und 4	A	3	0	80	0.999
	B	2	0	80	0.995
	limit	800	-1	80	-1
Subsysteme 2 und 3	A	5	0	80	0.999
	B	3	0	80	0.995
	limit	1000	-1	-1	-1
Komponente 1	A	150	-20	140	0.998
	B	100	0	80	0.995
Komponente 2	A	130	0	80	0.995
	B	80	0	75	0.999
Komponente 3	A	200	-20	100	0.998
	B	180	0	80	0.990

Tabelle 7.3.: *Liste der Kostenparameter der Systemteile*

7.3. Anbindung an die Entwurfsdatenbank auf Spezifikationsebene

Die mit dem vorgestellten Algorithmus bestimmten Kostenparameter sind auch auf Spezifikationsebene von Interesse. Daher bietet sich eine bidirektionale Kopplung mit dem in Kapitel 5 beschriebenen Entwurfstool an. Einerseits können bereits in der Spezifikation bekannte Kostenwerte übernommen werden, andererseits dienen die Ergebnisse der simulativen Kostenverknüpfung der Konkretisierung der Spezifikation. Die Kostengrenzen ergeben sich in der Regel aus konkreten Anforderungen in der Spezifikation und können somit automatisiert in die cost_limit-Vektoren übernommen werden. Als Ansatzpunkt im SpecScribe-Konzept bieten sich die Elemente *Parameter* als Kostenwerte und -grenzen sowie *ParameterRelationship* als Verknüpfungsmethoden. Die dazu notwendigen Algorithmen sind nicht Bestandteil dieser Arbeit und werden in [112] dargestellt.

```
Realisation A:
Global System Cost: -0.522839
Check_System: Design is realisable
system1 consists of (cost vector: 1970 0 80 0.9549
                      cost limit: 2000 0 80 0.95)
    subsys3 consists of (cost vector: 921 0 80 0.979185
                         cost limit: 1000 -1 -1 -1 )
        Component1 cost vector: 150 -20 140 0.998
                   cost limit: -1 -1 -1 -1
        subsys4 consists of (cost vector: 766 0 80 0.982
                             cost limit: 800 -1 80 -1)
        subsys1 consists of [...]
            Component1 cost vector: 150 -20 140 0.998
                       cost limit: -1 -1 -1 -1
            Component2 cost vector: 130 0 80 0.995
                       cost limit: -1 -1 -1 -1
            subs4_self cost vector: 3 0 80 0.999
                       cost limit: -1 -1 -1 -1
        subs3_self cost vector: 5 0 80 0.999
                   cost limit: -1 -1 -1 -1
    [...]

Realisation B:
INFO: In system1_1.subsys3_1.subsys4_1 3. value exceeds cost
   limit
INFO: In system1_1 3. value exceeds cost limit
INFO: In system1_1 4. value exceeds cost limit
Global System Cost: -0.727313
Check_System: Design is NOT realisable
system1 consists of (cost vector: 1373 0 75 0.895
                      cost limit: 2000 0 #80 #0.95)
    subsys3 consists of: (cost vector: 647 0 75 0.944367
                          cost limit: 1000 -1 -1 -1 )
        Component1 cost vector: 100 0 80 0.99
                          cost limit: -1 -1 -1 -1
        subsys4 consists of: (cost vector: 544 0 75 0.9587
                          cost limit: 800 -1 #80 -1 )
            subsys1 consists of: [...]
            Component1 cost vector: 100 0 80 0.99
                       cost limit: -1 -1 -1 -1
            Component2 cost vector: 80 0 75 0.999
                       cost limit: -1 -1 -1 -1
            subs4_self cost vector: 2 0 80 0.995
                       cost limit: -1 -1 -1 -1
        subs3_self cost vector: 3 0 80 0.995
                   cost limit: -1 -1 -1 -1
    [...]
```

Listing 7.2: *Simulationsergebnis der Kostenanalyse*

8. Vorstellen eines Entwurfsablaufs für MEMS

Der in Abschnitt 2.3 als Ziel der Arbeit vorgeschlagene prinzipielle Entwurfsablauf für heterogene Systeme soll in diesem Kapitel konkretisiert werden. Die dazu nötigen Werkzeuge und Verbindungsarbeiten bildeten den Gegenstand der vorangegangenen vier Kapitel. Diese fügen sich zu dem in Bild 8.1 dargestellten Designprozess zusammen. Ein Zwischenstand des Ablaufs noch ohne Spezifikationserfassungswerkzeug wurde in Paris [133] präsentiert. Der folgende Abschnitt erläutert die Designmethodik für Mikrosysteme, daran schließt sich eine Betrachtung der Vor- und Nachteile an.

8.1. Überblick über den Designflow

8.1.1. Spezifikationserfassung und Datenbank

Der Entwurfsprozess beginnt mit der Erfassung der textuellen Spezifikation, wofür das in Kapitel 5 beschriebene Werkzeug „SpecScribe“ zum Einsatz kommt. Neben der Erfassung nichtformaler Daten, wie Texte oder Bilder, erlaubt das Werkzeug erste Formalisierungen, beispielsweise durch die Klassen *ReferenceSignalForm* oder hybride Automaten. Darüber hinaus ist die Angabe von Referenzmodellen (sog. „Golden Reference“) in Form von Dateisammlungen möglich. Die verwendete Datenbank gestattet eine schnelle Suche nach Parametern im üblicherweise umfangreichen Anforderungsdokument. Die durch die Klassenvernetzung entstehende Struktur der Anforderungen hilft sowohl bei der Konsistenzprüfung als auch bei späteren Dokumentationen.

Für die Datenbank stehen verschiedene Realisierungsvarianten zur Auswahl; man unterscheidet die zwei Hauptprinzipien der objektorientierten und der relationalen Datenbank. Während die objektorientierte Datenbank den Zusatzaufwand für die Ansteuerung minimiert, da sie den Verlinkungen der Objektstruktur automatisch folgt, bedeutet die Umsetzung der Klassenstruktur in eine relationale Datenbank die Entwicklung von Zusatzfunktionen für die einzelnen Klassen. Letztgenannte Datenbanken können jedoch große Datenmengen redundanzärmer verarbeiten und sind robuster gegenüber Strukturänderungen [134]. Die derzeitige Umsetzung von SpecScribe nutzt eine objektorientierte Datenbank, da diese für die Erprobung der Entwurfsmethodik eine einfach zu implementierende Lösung darstellt. Dabei zeigte sich jedoch die mangelnde Robustheit gegenüber Strukturänderungen im Datenbankschema, so dass ein Übergang zu einer relationalen Datenbank angestrebt wird.

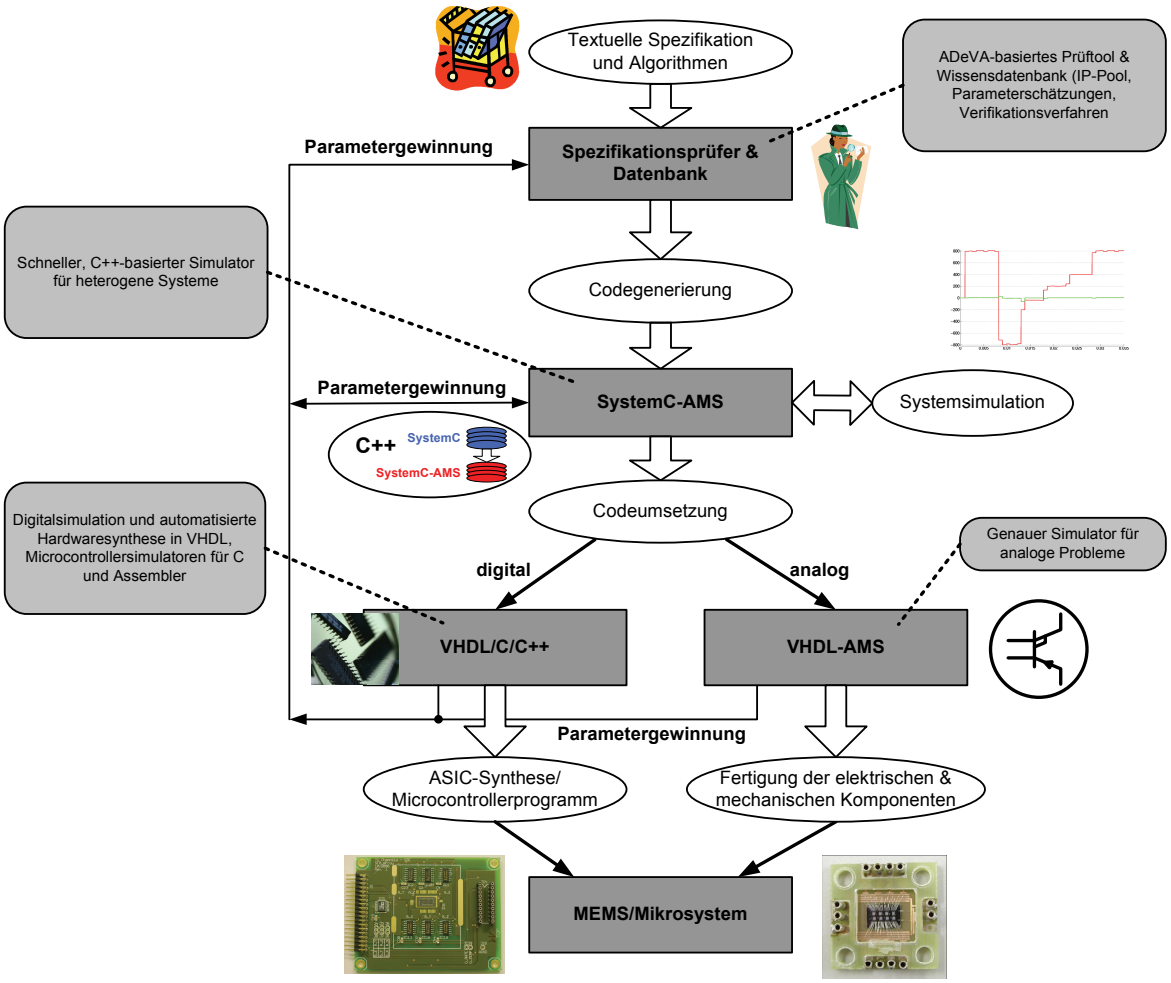


Bild 8.1.: *Im Rahmen der Arbeit entstandener Entwurfsablauf für heterogene Systeme*

Die Nutzung hybrider Modelchecker erlaubt die Prüfung der formalisierten Spezifikationsdaten auf Erreichbarkeit der Zustände, Eindeutigkeit (und damit Reproduzierbarkeit) des Verhaltens sowie auf einfache Aspekte der Konsistenz. Hier sind jedoch noch keine Werkzeuge mit ausreichendem Funktionsumfang verfügbar bzw. reicht die üblicherweise verfügbare Rechentechnik für umfangreiche Analysen unter dem Gesichtspunkt der Laufzeit und Speichergröße noch nicht aus. Mit dem Fortschreiten der Entwicklung der digitalen Modelchecker, die inzwischen in einigen Bereichen der Industrie zum Einsatz kommen, besteht die Aussicht, das auch die hybriden Modelchecker von den neuen Algorithmen profitieren und komplexere Systeme behandeln können.

Als Ergebnis dieses Entwurfsschritts steht eine in der Datenbank erfasste Spezifikation mit möglichst vollständig formaler und damit eindeutiger Beschreibung [135]. Diese enthält neben den funktionalen Anforderungen auch physikalische Parameter der Schnittstellen sowie ggf. Gehäuseformen und thermische Parametervorgaben.

8.1.2. Codegenerierung für SystemC-AMS

Die weitgehend formalisierte Spezifikation bildet die Grundlage für eine erste Systemsimulation. Diese recht grobe Abstraktion des Systems dient der Prüfung der geplanten Algorithmen in Zusammenhang mit den bereits bekannten Parametern des Systems und seiner Umgebung und kann als „ausführbare Spezifikation“ angesehen werden. Die Systembeschreibung ermöglicht unter anderem den Test der zwischen den Modulen definierten Schnittstellen und eine erste Abschätzung der Verarbeitungszeit. Bei der Fortsetzung des Entwurfsprozesses in Einzelkomponenten kann dieses abstrakte Modell als Umgebungsmodell der konkretisierten Teilsysteme fungieren.

Die Entwerfer verfeinern das abstrakte SystemC-AMS-Modell. So entsteht schrittweise ein Systemmodell auf funktionaler Ebene unter Verwendung gerichteter Datenflussgraphen. Die Möglichkeiten des digitalen SystemC zum Entwurf von Software sowie zur Modellierung von Kommunikation wie z. B. Transaction Level Modeling (TLM) [136] oder System Verification Environment (SVE) [137] erlauben schnelle Weiterentwicklungen der Spezifikation. Einzelne analoge Teile können bereits in Form linearer elektrischer Netze bzw. deren Analogie wie z. B. Feder-Masse-Dämpfer-Anordnungen modelliert werden.

8.1.3. Partitionierung und Codeumsetzung

Anhand des Systemmodells in SystemC-AMS erfolgt die Partitionierung in die drei Bereiche Software, digitale Hardware und analoge Module. Dabei hilft die Untersuchung der Systemrealisierungskosten mit dem in Kapitel 7 vorgestellten Verfahren.

Die sequentiell abzuarbeitende Software behält dabei ihre Beschreibungsform, da SystemC-AMS als Erweiterung zum C++-basierten SystemC bereits die weit verbreitete Programmiersprachenfamilie C/C++ direkt unterstützt. Nahezu alle Rechner- und Betriebssysteme vom 8-bit-Mikrocontroller bis hin zu leistungsstarken Mehrprozessorsystemen bieten Compiler, die C-Code in die jeweilige Maschinensprache umsetzen. Andere Sprachen wie Python oder Java erlauben die Integration von C-Code in ihre Quelltexte, sodass C/C++ für derzeitige Digitalsysteme einen hohen Universalitätsgrad besitzt.

Der Entwurf von digitaler Hardware konzentriert sich neben der Zusammenschaltung von Standardkomponenten wie Prozessoren und Speicher auf die beiden Sprachen VHDL und Verilog. Während letztere vor allem in Amerika Verbreitung findet, nutzt ein Großteil der außeramerikanischen Designteams VHDL. Im Rahmen dieser Arbeit erhielt die Umsetzung nach VHDL den Vorzug gegenüber Verilog. Zwar existiert auch die Möglichkeit, direkt aus SystemC Hardware zu synthetisieren, dies beschränkt sich jedoch auf ein Subset der Sprache und konnte sich auf dem Markt bisher nicht durchsetzen. Die Übertragung des Digitalverhaltens nach VHDL ermöglicht die Nutzung der in den Firmen vorhandenen Standardflows in Entwurf und Fertigung. Erfahrene De-

signer können die umgesetzte Hardwarebeschreibung weiter optimieren und so eine je nach Vorgabe flächen- oder laufzeitoptimale Lösung des Teilproblems realisieren. Dabei helfen ihnen Werkzeuge der Toolhersteller, wie z. B. Synopsys oder Mentor Graphics.

Im elektrischen und nichtelektrischen Analogbereich stellt sich den Entwerfern eine vollkommen andere Toollandschaft dar. Es existieren keine durchgängigen Werkzeuge, die wie im Digitalen eine automatisierte Umsetzung von funktionaler Ebene in die Schaltungs- oder Layoutebene zulassen. Ebenso ist eine Abschätzung der nach der Fertigung erreichbaren Parameter nur mit viel Erfahrung möglich, da mit weiterer Abnahme der Strukturbreiten die parasitären Effekte immer mehr an Einfluss gewinnen. Die im Entwurfsablauf integrierte Codeumsetzung nach VHDL-AMS kann daher nur ein Hilfsmittel auf dem Weg zum analogen Aufbau gleich welcher Domäne sein. Das für Zeit- und Wertkontinuität noch relativ abstrakte SystemC-AMS-Modell muss auf Basis von Differentialgleichungen weiter verfeinert werden. Um gerade im Bereich der Mikromechanik zuverlässige Parameter der Struktur zu erhalten, erfolgt meist der Rückgriff auf FEM-Simulationen, die zusammen mit Methoden der Ordnungsreduktion zur Modellgewinnung auf Verhaltens- und Schaltungsebene dienen.

8.1.4. Implementierung

Die Umsetzung der Komponentenbeschreibungen in reale Hardware stellt nach wie vor große Anforderungen an das Know-how der Entwurfsingenieure. Zwar bieten im Digitalbereich zahlreiche Tools eine gute Unterstützung durch Bereitstellen von Algorithmen zu Synthese, Mapping und Platzierung/Trassierung, jedoch ist eine umfangreiche Prüfung der Ergebnisse nach wie vor unerlässlich. Die Durchführung von Simulationen mit aus der Technologiesynthese ermittelten Verzögerungszeiten (sog. Backannotation) führt zu langen Simulationszeiten, die bei komplexen Entwürfen mehrere Stunden pro Sekunde Realzeit erfordern können. Die Generierung von Testpattern und Hardware-Monitoren, die Fehler im Rahmen der Emulation automatisiert finden, ist Ziel der Arbeiten von Tischendorf [113]. Die Emulation reduziert die Dauer der Funktionsprüfung erheblich, im Idealfall kann der Entwurf in Echtzeit geprüft werden.

Wie bereits im letzten Abschnitt geschildert, erfordert die Umsetzung des Analogbereichs viel Hintergrundwissen. Eine reine Top-Down-Lösung kommt auf den niedrigsten drei Entwurfsebenen nicht in Betracht, hier erfolgt ein iteratives Vorgehen, um die Vorgaben der Spezifikation und der Schnittstellenbeschreibungen einzuhalten. Dabei ist immer eine Balance zwischen Modellgenauigkeit und Modellerstellungsaufwand bzw. Rechenaufwand zu finden. Simulationszeiten von mehreren Tagen sind bei heutigen Entwurfszeiten im industriellen Umfeld nicht mehr tolerierbar, die Zeit bis zur Produkteinführung bietet dafür nicht genug Raum. Mit der Einführung von Nanotechnologien und dem damit verbundenen weiteren Anstieg der Systemkomplexität bei gleichzeitigem Hinzukommen neuer technologisch bedingter Effekte verschärft sich dieses Problem weiter. Die Extraktion der relevanten Modellanteile ist dabei essentiell um die Modellkomplexität zu reduzieren.

Die Inbetriebnahme und der ausführliche Test des Systems bilden den Abschluss des Entwurfs und stellen den Übergang zur Überwachung während der Produktlebensdauer („Lifetime-Monitoring“) dar. Hierbei werden die Produkte während der Nutzungsdauer beim Kunden weiter beobachtet, um bei Problemen schnell gegensteuern zu können. Bei Systemlaufzeiten im Automobil- und Medizinbereich von mehr als 10 Jahren muss die Zuverlässigkeit ständig gewährleistet sein. Eine Nachverfolgung von Ausfällen einzelner Systeme hilft Probleme bei verwandten Systemen zu erkennen und gegebenenfalls zu beheben.

8.1.5. Parametergewinnung

Die Verfeinerung der Komponentenmodelle und das Bereitstehen erster Prototypen ermöglicht die Bestimmung weiterer Parameter des Systems, die auf hohen Abstraktionsebenen gar nicht oder nicht exakt bestimmbar sind. So kann auf funktionaler Ebene zwar eine Abschätzung der Laufzeit der Algorithmen und Messungen erfolgen, definitive Aussagen zur Durchlaufzeit sind aber erst in den späten Phasen der Implementierung möglich. Die im Laufe des Entwurfsprozesses gewonnenen oder konkretisierten Parameter müssen sowohl in die Entwurfsdatenbank als auch in das Systemmodell eingepflegt werden. Sie dienen einerseits der Prüfung, ob sich das System noch im Rahmen der Spezifikation bewegt, und andererseits der Dokumentation. Nach Abschluss des Entwurfsprozesses enthält diese Datenbank im Idealfall alle Informationen zur Generierung von Datenblättern und Zertifizierungsunterlagen. Dies ist ein Ziel des Projekts „ParaObsol“, das zusammen mit Verbundpartnern an der Professur bearbeitet wird. Eine Dokumentengenerierung auf Basis einer VHDL-AMS-Beschreibung stellt das Werkzeug ASPECTOR [10] bereit.

8.2. Diskussion des vorgestellten Entwurfsablaufs

8.2.1. Vorteile

Der vorgestellte Ablauf bietet eine durchgehende Toolkette von der Spezifikationserfassung über die Gesamtsystemsimulation bis hin zur Verhaltensebene. Die sonst beim Übergang zwischen den Tools auftretenden Informationsverluste, Übertragungsfehler und Inkonsistenzen können so unterbunden werden. Die Speicherung aller Daten des Systems in der Entwurfsdatenbank vermeidet Datenverluste und stellt als zentrale Plattform die erstellten Modelle dem Entwicklerteam zur Verfügung, was die bei großen, unübersichtlichen Projekten oft auftretende Doppelarbeit minimiert.

Die Formalisierung auch der analogen Daten unterstützt die Sicherstellung der Konsistenz und der Eindeutigkeit der Spezifikation. Die bei menschlichen Sprachen auftretenden Bedeutungsdopplungen und kontextabhängigen Interpretationen entfallen bei formalen Sprachen. Die Ausdehnung der Formalisierbarkeit auf den Analogbereich auch in hohen Abstraktionsebenen ermöglicht eine bessere Prüfung der Spezifikation, als dies

bisher durch Reviewprozesse möglich ist. Damit einher geht die Möglichkeit, bereits während des Spezifikationsprozesses ein erstes Systemmodell zu generieren und zu simulieren. Die sog. „ausführbare Spezifikation“ in Form eines SystemC-AMS-Modells erleichtert die Aufgabe der Systementwickler durch schnelle Prüfbarkeit der getroffenen Entscheidungen.

Diese im Rahmen des Entwurfs getroffenen Festlegungen bleiben in der Datenbank gespeichert und ermöglichen so eine Vereinfachung der Dokumentation des Systems. Die Nachverfolgbarkeit bildet die Grundlage von Zertifizierungsvorschriften unter anderem im Automobil- und Medizintechnikbereich. Des weiteren entsteht durch die Gewinnung von Parametern während der Implementierung ein konsistentes Datenblatt für das Systemverhalten.

Das Systemmodell sowie die erfassten formalisierten Anforderungen können zur Erstellung von Verifikationsvorschriften („Properties“) und Testsequenzen genutzt werden. Die Prüfung der Gleichheit des Systemverhaltens mit der Spezifikation nach jedem Entwurfsschritt ist ein essentieller Punkt im Designprozess. Nur so kann sichergestellt werden, dass das am Ende gefertigte System auch den Wünschen des Kunden entspricht und seine Bedürfnisse erfüllt. Die Verifikation analoger Systemteile steht jedoch noch am Anfang des Forschungsprozesses, hierzu sind insbesondere durch die hohen Komplexitäten viele Arbeiten bis zu einer praxistauglichen Lösung nötig.

Die Übergänge zwischen den Beschreibungsmitteln, wie Spezifikations- und Entwurfsdatenbank, hybriden Verifikationstools, SystemC-AMS als Systemsimulationssprache und den als Ausgangsstufe zur Implementierung dienenden Sprachen VHDL bzw. VHDL-AMS werden durch Konverter syntaktisch unterstützt, so dass sich die Zahl der Fehlerquellen im Schnittstellenbereich, bei Datentypenkonvertierungen und bei grundlegenden Algorithmenumsetzungen reduziert. Damit entfallen im Designprozess Arbeiten für die manuelle Umsetzung der Modelle, so dass dem Entwerfer mehr Zeit für die kreative Umsetzung der Spezifikation bleibt und damit der in der Einführung geschilderte „Design Gap“ gemildert wird.

8.2.2. Schwächen und Erweiterungsmöglichkeiten

Der angegebene Designflow enthält einige Schwächen, die in zukünftigen Entwicklungen der Toolkette Berücksichtigung finden sollten. Der folgende Abschnitt stellt diese kurz dar und schlägt einige Lösungsmöglichkeiten vor.

Eine erste Schwäche des Konzepts stellen die derzeit noch eingeschränkten Prüfmöglichkeiten im Bereich der hybriden Automaten dar. So sind nur lineare Automaten (HyTech) bzw. Automaten geringer Komplexität mit den heutigen rechentechnischen Möglichkeiten behandelbar. Der Ansatz der hybriden Automaten stößt auf eine immer breitere Anwenderresonanz, so ist beispielsweise eine Integration der hybriden Automaten in Modelica geplant. Im Rahmen der damit neu entstehenden Werkzeuge und der fortschreitenden Möglichkeiten der Hochleistungsarithmetik sowie steigender Leis-

tungsfähigkeiten der verfügbaren Rechentechnik sollten auch komplexe hybride Automaten in absehbarer Zeit behandelbar sein.

Die Behandlung und Verfeinerung des analogen Teilsystems erfolgt im vorgestellten Ablauf zunächst ausschließlich in VHDL-AMS. Diese Sprache findet zwar immer weitere Verbreitung in der Industrie, jedoch stößt sie als quellcodebasierte Beschreibungsmöglichkeit auf Vorbehalte seitens der Entwerfer, die grafische Eingabemöglichkeiten („Schematics“) nutzen möchten. Die Unterstützung weiterer Sprachen, im elektrischen Bereich beispielsweise von Spice-Derivaten, würde die Akzeptanz des Entwurfsablaufs verbessern und die Toolkette flexibler gestalten. Auch eine Einbindung von Matlab/Simulink als weitverbreitetes signalflussbasiertes Modellierungswerkzeug würde die Toolkette bereichern und die umfangreichen Algorithmenbibliotheken in den Entwurf heterogener Systeme einbringen.

Im Zusammenhang mit der Unterstützung weiterer Sprachen stellt sich die Frage der Kopplung von Simulatoren. Die AMS-Sprachen wie SystemC-AMS und VHDL-AMS sollten vom Sprachumfang her auf Kopplungen verzichten können und die Komponenten selbst nachbilden. Im Zuge der weiter abnehmenden Strukturbreiten bis in den Nanometerbereich und der Erhöhung der Komplexität ist zu überlegen, ob eine Partitionierung in abstrakt beschriebene digitale sowie „unkritische“ analoge Komponenten (in SystemC-AMS) und exakt modellierte verhaltenskritische Analogkomponenten (in Spezialsimulatoren) zum Erfolg führen würde. Mit diesem Thema soll sich ab 2010 ein fakultätsübergreifendes Forschungsprojekt an der TU Chemnitz beschäftigen.

Im Bereich des Entwurfs der Mikromechaniken und darauf zugeschnittener Auswertestufen ist es derzeit kaum möglich, einen reinen Top-Down-Entwurf durchzuführen. Dazu benötigte Bibliotheken von Standardelementen (wie die entsprechenden Grundgatter im Digitalbereich) existieren noch nicht bzw. sind im Rahmen einer automatisierten Synthese noch nicht beherrschbar. Durch die häufigen Iterationen auf niedriger Abstraktionsebene ändern sich die Parameter der Schaltungen noch relativ häufig. Die im Entwurfsablauf bisher manuell erfolgende Parametergewinnung sollte in diesem Zusammenhang automatisiert werden. Dies bedingt eine eindeutige formale Identifizierbarkeit der Parameter sowie generische Bestimmungsmethoden. Die Ergebnisse des Projekts „ParaObsol“ zum Ersatz nicht mehr verfügbarer Bauteile sollen in diesem Punkt Abhilfe schaffen.

9. Beispiel Universelles Bewegungsanalysesystem

Im Rahmen des Sonderforschungsbereichs 379 entstand in Zusammenarbeit mit verschiedenen Professuren der TU Chemnitz ein Demonstrator „Universelles Bewegungsanalysesystem“ (UBAS). Auf diesen Demonstrator soll der in Kapitel 8 vorgestellte Entwurfsablauf beispielhaft angewandt werden.

Zweck des Systems ist die Messung von Bewegungen eines autonomen Roboters und daraus herleitend die Gewinnung von Positionsdaten. Damit erhält man ein System zur sog. „Inertialnavigation“ (INS), also zur Positionsbestimmung basierend rein auf am System selbst gemessenen Werten zu Beschleunigung und Drehrate. Diese INS sind unabhängig von externen Signalquellen (GPS-Satellitensignal) und damit robust gegen Signalabschattungen vor allem im Indoor-Bereich. Durch die an den Sensoren auftretenden Messfehler ergeben sich jedoch Abweichungen in der Positionsbestimmung, so dass diese inertialen Verfahren in der Regel mit Stützstellenverfahren kombiniert werden müssen. Stützstellen als bekannte „Landmarken“ ermöglichen die Korrektur des sich summierenden Fehlers der inertialen Positionsbestimmung.

9.1. Anforderungen an das System

Aus dem Zweck des Systems lassen sich die ersten Anforderungen erstellen, die in Zusammenarbeit mit dem Auftraggeber zu verfeinern sind. Diese bilden die Systemspezifikation und können (zunächst in informaler, textueller Form) in SpecScribe verwaltet werden. Die im folgenden angegebenen Anforderungen sind Bestandteil der umgesetzten Spezifikation des UBAS und daher bewusst in Englisch formuliert.

Die Systemspezifikation beginnt mit der Grundanforderung

”need a GPS independent navigation system”

die als *TopRequirement* mit dem Namen ”Inertial navigation system” den zentralen Punkt der Spezifikation bildet. An diese zentrale Anforderung schließen sich die ersten Konkretisierungen, jeweils in der Form Anforderungsname: ”Beschreibung”, an. Dabei entspricht die Einrückung der Hierarchieebene im Anforderungsbaum.

- Signal sources: ”measures acceleration and rotation”
  - Sensor System: ”needs 3 acceleration sensors and 3 rotation sensors, one for each axis”

  - digitalPart: "generate position information from sensor inputs"
    * Coordinates: "need a conversion from sensor coordinates to environment coordinates"
    * Supporting points: "need sensor-independent position updates for error correction"

- Accuracy: "should measure the relative position with 99 percent accuracy from origin"
  - digitalAccuracy: "resolution should be mm, largest displacement $\pm 500\,\mathrm{m}$"

Aus diesen Anforderungen lassen sich bereits erste Hinweise für die Implementierung entnehmen. So soll im Digitalteil ein Wertebereich von $\pm 500\,\mathrm{m}$ mit einer Auflösung von 1 mm sichergestellt werden, was 1.000.000 Stufen entspricht. Bei einer Festkommarealisierung sind hier also mindestens 20 bit Datenbreite erforderlich. Bild 9.1 zeigt die erweiterte Anforderungshierarchie als Ausschnitt der SpecScribe-Oberfläche.

TreeView	Type
Inertial Navigation System	specification
1. Signal sources	textual requirement
1.1. Sensor System	analog requirement
1.1.1. Vdd plane	analog requirement
1.1.2. measurement method for acceler...	textual requirement
1.1.2.1. AccelerationConversion	analog requirement (ref. signalform)
1.1.2.1.1. AccelerationSensor	component requirement
1.1.3. measurement method for rotation	textual requirement
1.1.3.1. RotationSensor	component requirement
1.1.3.1.1. rotation_sensor	behavioral component (FSM)
1.1.3.1.2. buy rotation	textual requirement
1.1.3.1.3. rotation accuracy	textual requirement
1.1.4. ADConverter	component requirement
1.1.4.1. ADC	behavioral component (FSM)
1.1.4.2. ADCRange	textual requirement
1.1.4.3. ADCSize	textual requirement
1.2. digitalPart	textual requirement
1.2.1. ErrorCorrection	component requirement
1.2.1.1. removePeaks	textual requirement
1.2.1.2. CompareMovement	textual requirement
1.2.2. Integration	component requirement
1.2.2.1. SimpsonIntegration	textual requirement
1.2.3. FinalErrorCorrection	component requirement
1.2.3.1. position_correction	behavioral component (FSM)
1.2.3.2. Map Correction	textual requirement
1.2.4. ShowData	component requirement

Bild 9.1.: *Baumansicht der Anforderungshierarchie*

Neben diesen textuellen Anforderungen sind auf Systemebene bereits erste formale Requirements angebbar. So kann beispielsweise die Umsetzung der aufgenommenen Beschleunigung in eine auswertbare elektrische Spannung als direkte Proportionalität über eine *ReferenceSignalForm* (siehe Abschnitt 5.3) definiert werden. Bild 9.2 zeigt den Screenshot dieser Definition in der SpecScribe-GUI.

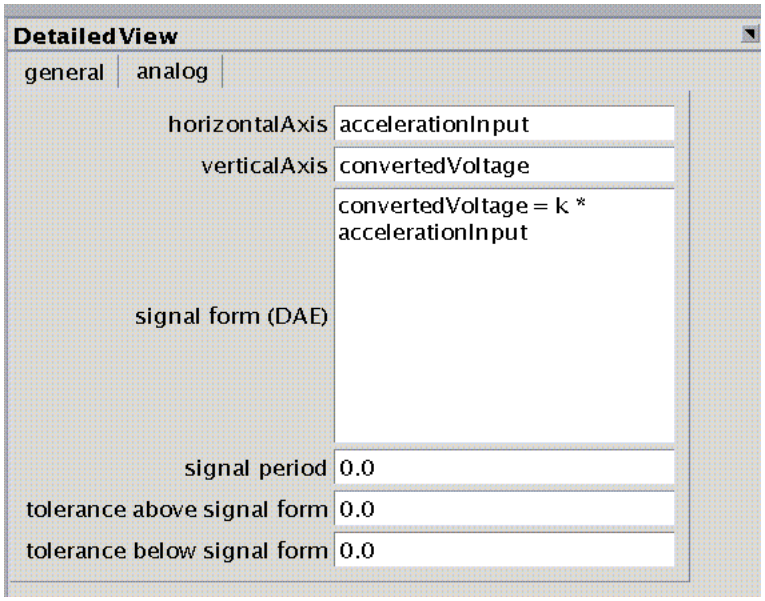


Bild 9.2.: *ReferenceSignalForm der Beschleunigungs-Spannungs-Wandlung*

9.2. Systemüberblick

Das Inertialnavigationssystem UBAS lässt sich in drei Subsystemene unterteilen: einen Analogteil mit Fehlerkorrektur, einen Digitalteil mit Koordinatentransformation und einen Softwareteil [138]. Bild 9.3 gibt dazu einen Überblick über das Systemkonzept vor der Untersetzung mit Einzelkomponenten.

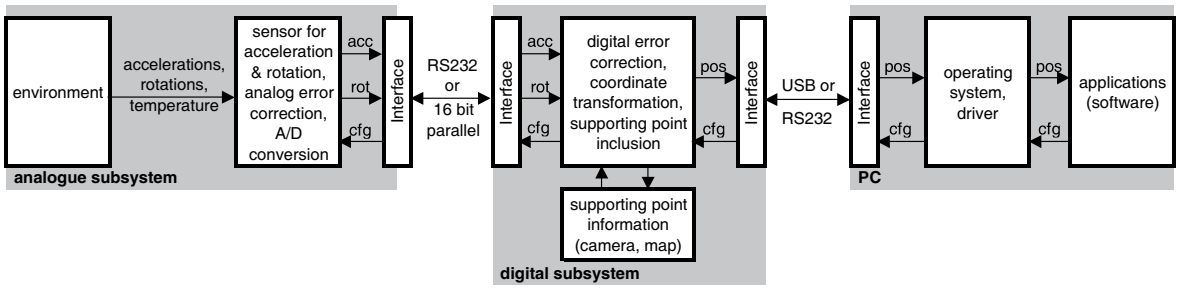


Bild 9.3.: *UBAS-Systemüberblick*

Die Beschleunigungs- und Rotationssensoren werden durch von der Testbenchkomponente „environment“ bereitgestellte Stimuli angeregt. Nach der Wandlung der Messgrößen in elektrische Größen ist eine erste sensornahe Fehlerkorrektur beispielsweise von Temperatureinflüssen nötig. Nach der Analog-Digital-Umsetzung (ADU) findet die weitere Bearbeitung im digitalen Teilsystem statt. Neben der digitalen Driftkompensation umfasst der Digitalteil die Koordinatentransformation, die Integration der Beschleunigungen und die Berücksichtigung von Stützstelleninformationen. Die so erhaltenen Positionsdaten dienen dem Softwareteil als Grundlage zur Darstellung des zurückgelegten Weges und zur Erzeugung von Steuerinformationen.

9.3. Formalisierung der Spezifikation

9.3.1. Umsetzung des Sensorverhaltens in hybride Automaten

Als Beispiel für die Umsetzung des analogen Systemverhaltens sollen drei Teilautomaten des kapazitiven Beschleunigungsaufnehmers kurz erläutert werden [139]. Dazu ist es notwendig, die elementaren Abläufe in der Sensoranordnung zu verstehen. Die einwirkende Beschleunigung *acc* verändert den Abstand der Kämme d und damit die Kapazität C_{Mess}. Durch eine zyklische Aufladung und Entladung mittels einer Dreieckspannung U kann diese Kapazität als Stromstärke I gemessen werden. Der Sensor selbst weist ein lineares Temperaturverhalten auf und erwärmt sich nach dem Einschalten durch die eigene Verlustwärme.

Dieses thermische Verhalten stellt der hybride Automat in Bild 9.4 dar. Nach dem Einschalten bei einer Umgebungstemperatur $T_0 = 20\,°C$ erwärmt sich der Sensor innerhalb von 2000 Zeiteinheiten um insgesamt $20\,°C$. Dazu enthält der Automat zwei Zustände *heat* (Eigenerwärmung) und *stable* (keine Temperaturänderung mehr).

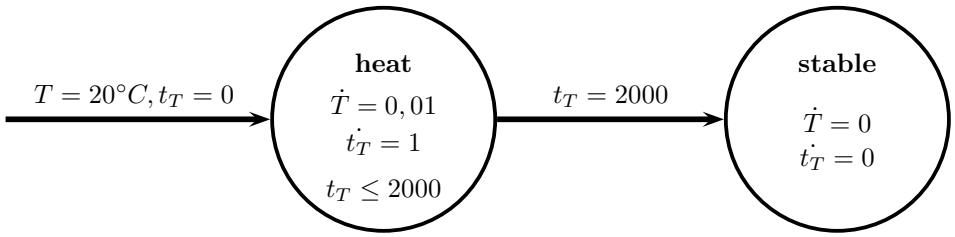


Bild 9.4.: *Hybrider Automat zum Temperaturverhalten*

Der in Bild 9.4 dargestellte Automat umfasst zwei Variablen $X = \{T, t_T\}$, wobei T die Temperatur und t_T die Erwärmungszeit darstellt. Die Zustände *heat* und *stable* bilden die Menge der Orte v, die zusammen mit dem Übergang E unter der Sprungbedingung $jump$ ($t_T = 2000$) den Graphen V formen. Der Startort *heat* wird mit der Initialbedingung $init = \{T = 20\,°C, t_T = 0\}$ erreicht. Der Ausdruck $t_T \leq 2000$ stellt die Invariante *inv* dar, die den am Ort *heat* möglichen Wertebereich für t_T angibt. Der Ort *heat* enthält außerdem zwei Transferfunktionen $flow = \{\dot{T} = 0,01; \dot{t_T} = 1\}$, die den zeitlichen Verlauf der Variablen angeben. Bild 9.5 zeigt die Zustandstabelle in der SpecScribe-GUI.

Die Temperatur des Sensors hat Einfluss auf die Messwertumsetzung, da die Sensorkennlinie linear von der Temperatur abhängt. Bild 9.6 zeigt diesen aus zwei Orten *normal* und *snap* bestehenden Automaten. Im Normalmodus erfolgt die Umsetzung der Beschleunigung *acc* in die Stromstärke I temperaturabhängig über die sich ändernde Kapazität C_{Mess}. Dabei gibt k_T den linearen Temperaturkoeffizienten bezogen auf $40\,°C$, C_0 die Kapazität in Ruhelage und c_{acc} den Proportionalitätsfaktor für die Umsetzung an. Es wird angenommen, dass auf Systemebene ein linearer Zusammenhang zwischen Beschleunigung und Kapazitätsänderung gilt.

DetailedView

general | ports | variables | states | transitions | fsmVisuals | types

	State	initState	C	O	OnInside/InState	Invariants
1	heat	x			derivative(t_temp) = 1; derivative(out_temp_var) = 0.01	t_temp <= 2000
2	stable				derivative(t_temp) = 0; derivative(out_temp_var) = 0	

Bild 9.5.: *GUI-Screenshot der Zustandstabelle des Temperaturautomaten*

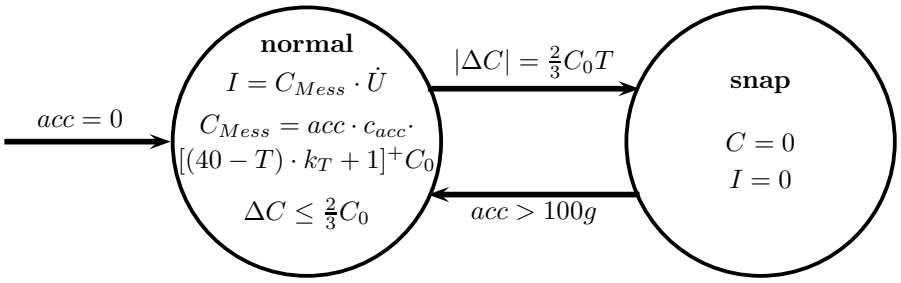


Bild 9.6.: *Hybrider Automat zur Umsetzung der Beschleunigung in einen Stromfluss*

Verursacht eine Beschleunigung jedoch eine zu große Auslenkung der kapazitiven Struktur, „verkleben“ die Elektroden miteinander. Als Annahme tritt der Vorgang in diesem Design sprunghaft bei einer Auslenkung auf, die den dreifachen Wert der Ruhekapazität C_0 aufweist. Ist diese Bedingung erfüllt, springt der hybride Automat an den Ort *snap*. Durch den Kurzschluss ist keine Kapazitätsänderung mehr detektierbar. Eine Reversion des Schnappeffekts (und damit Rückkehr in den Normalbetrieb) ist nur durch mechanische Einwirkung möglich.

Die Generierung der Messspannung zum Laden und Entladen der Kapazität ist Gegenstand des in Bild 9.7 abgebildeten dritten hybriden Automaten. Dieser ebenfalls zwei Orte – *load* und *unload* – umfassende Automat bildet den Dreieckspannungsgenerator nach. Ein Timer t_U schaltet zwischen steigender und fallender Flanke um, dabei wird von der Möglichkeit der Hybridtheorie Gebrauch gemacht, zum Umschaltzeitpunkt neue Variablenwerte zuzuweisen. In diesem Fall erfolgt ein Rücksetzen des Timers t_U.

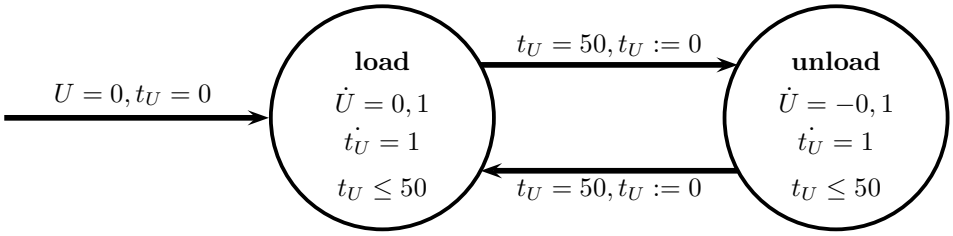


Bild 9.7.: *Hybrider Automat zur Generierung der Messspannung*

9.3.2. Umsetzung der Auswertung in digitale Automaten (FSM)

An die analoge Messwertaufnahme schließt sich die Analog-Digital-Wandlung an. Dem Digitalteil stehen die Beschleunigungswerte der drei Achsen sowie die Rotationswerte der drei Ebenen zur Verfügung, um daraus verlässliche Positionsinformationen zu generieren. Die bei der Messwerterfassung auftretenden Fehler erfordern Maßnahmen zur Verbesserung der Positionsgenauigkeit. Exemplarisch soll kurz diese Korrektureinheit in abstrakter formalisierter Form erläutert werden. Bild 9.8 zeigt den Automatengraphen für die Korrektureinheit. Diese FSM liest zyklisch die Positions- und Geschwindigkeitsdaten der Sensoren ein und prüft diese auf Plausibilität. So können Geschwindigkeitswerte größer als die 36 km/h verworfen werden, da die Trägerplattform des UBAS schnellere Bewegungen nicht zulässt. Ebenso kann sich das System nicht durch Wände und Decken bewegen, sofern dort nicht Öffnungen vorgesehen sind.

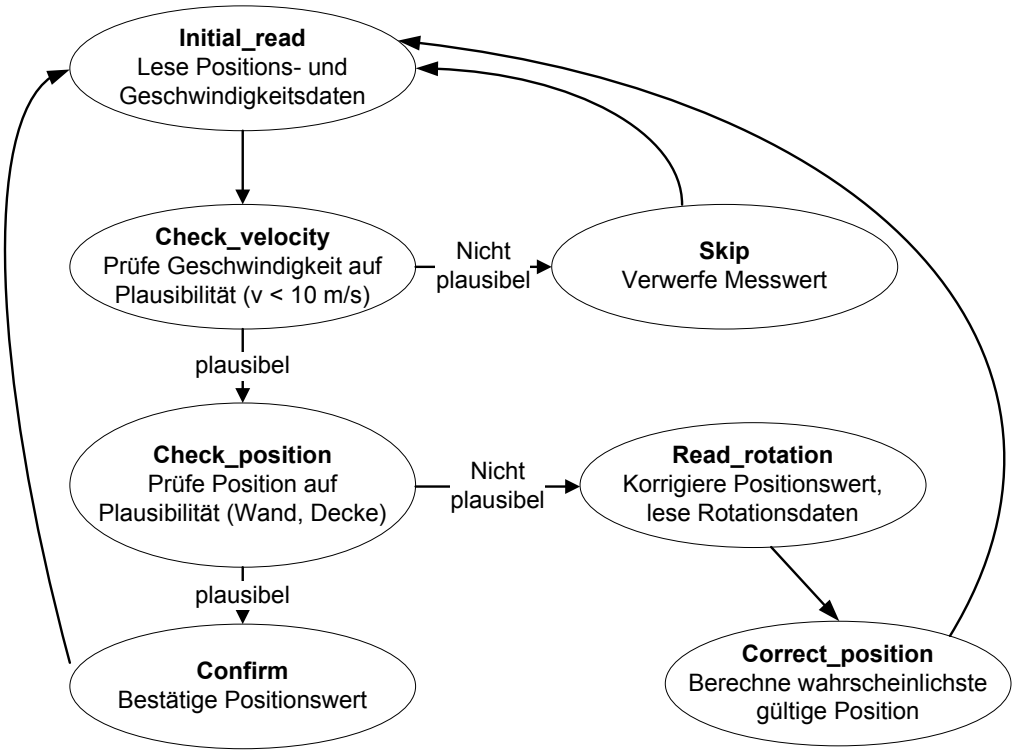


Bild 9.8.: *Digitaler Automat zu Positionskorrektur*

Zur Umsetzung in SpecScribe muss dieser Graph an dessen Datenstruktur angepasst und in der grafischen Oberfläche in die Entwurfsdatenbank eingegeben werden. Bild 9.9 zeigt die Definition des Automaten in SpecScribe, dabei entsprechen die Zustandsübergangsbedingungen den MTT und die Signalzuweisungen (*Actions*) den DTT der in Abschnitt 5.1.1 dargestellten ADeVA-Semantik.

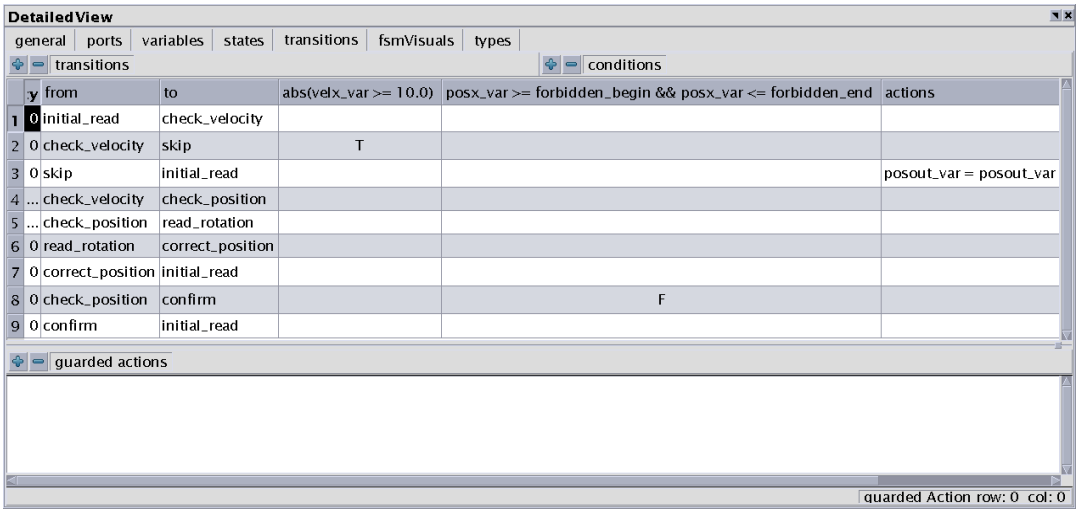


Bild 9.9.: *Screenshot der SpecScribe-Eingabe der Positionskorrektur*

9.4. Export nach SystemC-AMS

Die Generierung eines ersten Systemmodells erfolgte auf Basis der Beschreibungen des letzten Abschnitts. Die formal erfassten Spezifikationsdaten bilden die Grundlage des simulierbaren SystemC-AMS-Codes. Die Modulbeschreibungen in Form digitaler FSM bzw. hybrider Automaten und deren Schnittstellenangaben werden vom Codegenerator in compilierbaren Quellcode umgesetzt. Die Listings 9.1 (Headerdatei) und 9.2 (CPP-Datei) zeigen den vollautomatisch erzeugten, compilierbaren und simulierbaren Code für den hybriden Automat zur Messspannungsgenerierung (siehe Bild 9.7). Die Umsetzung der Beschreibung von Ableitungen durch die Nutzung der Wertedifferenz stellt eine Linearisierung der Ableitung dar. Alternativ kann eine manuelle Umsetzung in eine Laplace-Übertragungsfunktion erfolgen; mit der SystemC-AMS-Version 1.0 kann der Operator *dot()* genutzt werden. Der Codegenerator erzeugt weiterhin ein Makefile zum vereinfachten Compileraufruf sowie ein für die Simulation notwendiges Testbench-Template, das die Strukturinformationen des Moduls nutzt um die grundlegenden Codezeilen zu erstellen. Das Makefile enthält die Namen der Quelldateien und verweist auf ein generisches Makefile, das Bestandteil von „SpecScribe“ ist, jedoch durch Angaben von Umgebungsvariablen an die Rechnerplattform (z. B. Unix/Linux, Windows, MacOS) und den jeweils vorhandenen Compiler anzupassen ist.

9.5. Konkretisierung des Systems

Die bisherigen Betrachtungen des UBAS fanden auf hoher Abstraktionsebene statt. Zur Umsetzung der Spezifikation bzw. des Systemmodells in reale Hard- und Software müssen die Systemteile konkretisiert werden. In den folgenden Abschnitten stehen die Teilsysteme und deren Verfeinerung bis zur realen Umsetzung im Mittelpunkt.

```
#ifndef VOLTAGE_GENERATION_H
#define VOLTAGE_GENERATION_H

// voltage_generation for SystemC(-AMS) export by SpecScribe

#include "math.h"
#include "systemc-ams.h"

SCA_SDF_MODULE(voltage_generation) {
    // ports
    sca_sdf_out<double > voltage;

    // types/enums
    enum voltage_generationstatetype { load, unload};

    // signals and member vars
    double volt_var;
    int timer;
    voltage_generationstatetype voltage_generationstate;

    void sig_proc();

    SCA_CTOR(voltage_generation) {
        volt_var = 0;
        timer = 0;
        voltage_generationstate = load;
    }
};

// trace this module
void sc_trace( sc_trace_file *, const voltage_generation& );
void sc_trace( sc_trace_file *, const voltage_generation& ,
   const std::string &);

#endif
```

Listing 9.1: *SystemC-AMS Strukturbeschreibung des hybriden Automaten aus Bild 9.7*

```
// voltage_generation for SystemC(-AMS) export by SpecScribe
#include "voltage_generation.h"

void voltage_generation::sig_proc(){

    switch (voltage_generationstate){
        case(load):
            volt_var += (0.1) * voltage.get_T().to_seconds();
            timer += (1) * voltage.get_T().to_seconds();
            if (timer == 50) {
                timer= 0;
                voltage_generationstate = unload;
            }
            break;
        case(unload):
            volt_var += (-(0.1)) * voltage.get_T().to_seconds
               ();
            timer += (1) * voltage.get_T().to_seconds();
            if (timer == 50) {
                timer= 0;
                voltage_generationstate = load;
            }
            break;
    }
    voltage.write(volt_var);
}

void sc_trace( sc_trace_file *tf, const voltage_generation &
   module){
    sc_trace( tf, module, "" );
}

void sc_trace( sc_trace_file *tf, const voltage_generation &
   module,
     const std::string & hierarchyName){

    // internal signals
    // subcomponents
}
```

Listing 9.2: *Verhaltensbeschreibung des hybriden Automaten aus Bild 9.7*

9.5.1. Analoges Teilsystem

Die bisher verfolgte reine Top-Down Entwurfsmethodik kann im nichtelektrischen Bereich in der Regel nur in den hohen Abstraktionsebenen bis zur algorithmischen Ebene angewandt werden. Es existieren weder generische Bauelementebibliotheken, noch lassen sich technologische Parameter vorher so genau abschätzen, als dass automatisierte Synthesewerkzeuge zum Einsatz kommen könnten. Daher erfolgt üblicherweise aus den Festlegungen der algorithmischen Ebene zunächst ein Entwurf der Sensorstruktur auf der untersten Geometrieebene. Anhand dieser Struktur können durch geeignete Simulationsverfahren, wie z. B. FEM-Tools, die Nutz- und Störparameter der Struktur ermittelt werden. Diese dienen als Grundlage für die Erstellung abstrakterer Modelle (z. B. ermittelbar durch Ordnungsreduktion [75]). Die Simulationszeit der abstrakten Modelle ermöglicht eine Nutzung beim Entwurf der Umgebungsschaltung in Top-Down-Methodik.

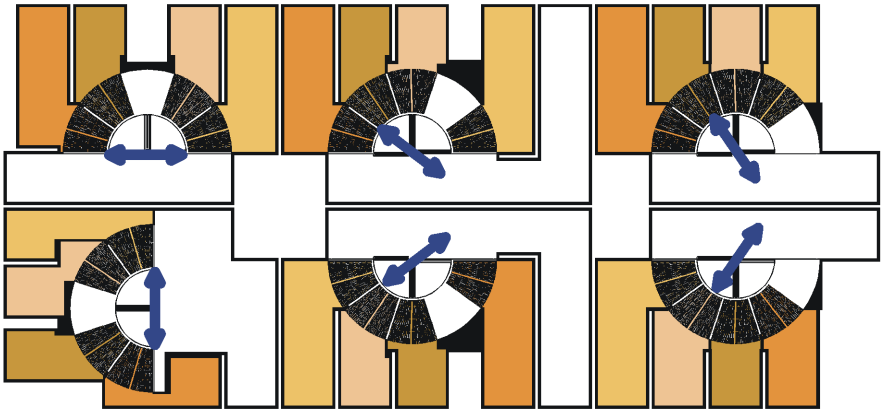


Bild 9.10.: *Beschleunigungssensorarray des UBAS [140]*

Für den Entwurf der elektrischen Auswerteschaltungen ist es notwendig, von den elektromechanischen Sensoren Modelle zu erstellen. Die vorliegende, in Bild 9.10 dargestellte, kapazitive Struktur [140] erlaubt eine direkte Umsetzung in Verhaltensmodelle basierend auf Differentialgleichungen, so dass auf ungenauere empirische Modelle verzichtet werden kann. Eine ausführliche Herleitung der Modellgleichungen erfolgte in [141], die gesamte Arrayschaltung und deren Simulation in SystemC-AMS ist Gegenstand von [117]. Grundlage der Modellierung bilden die Bewegungsgleichung für die Rotation 9.1 mit den Parametern einwirkendes Drehmoment M, Trägheitsmoment J, Dämpfung k und Federkonstante c zum Drehwinkel α sowie die kapazitive Umsetzung in einen auswertbaren Strom 9.2.

$$M = J \cdot \ddot{\alpha} + k \cdot \dot{\alpha} + c \cdot \alpha \tag{9.1}$$

$$I = C \cdot \dot{U} + U \cdot \dot{C} \tag{9.2}$$

Die kapazitive Anordnung mit 4 Segmenten pro Einzelsensor ermöglicht neben einer Auswertung von je zwei Segmenten als Differentialkondensator den rückgekoppelten Betrieb des Sensors. Dazu muss eine Regelschleife das Ausgangssignal des Sensors beobachten und diesen über elektrostatische Kräfte durch Anlegen von Steuerspannungen in seine Ausgangslage zurückführen. Das Messsignal ist in diesem Fall eine Funktion der Steuerspannung und nicht mehr allein der Auslenkung. Diese Möglichkeit der Rückführung wurde modelliert [142, 143], eine Umsetzung in Hardware war aufgrund der berechneten Steifigkeit der Federanordnung nicht notwendig, da diese Federn von sich aus eine zu große Annäherung der Kammstrukturen und damit den schon im hybriden Automaten „Umsetzung der Beschleunigung in einen Stromfluss“ (siehe Bild 9.6) modellierten Schnappeffekt verhindern.

Die vorgestellte Sensorstruktur erfordert eine analoge Auswerteschaltung, da die Messgröße Strom nicht direkt im spannungsbasiert arbeitenden Bereich der Digitalelektronik nutzbar ist. Als Schaltung kommt eine in Bild 9.11 dargestellte $\Delta C - U$-Wandlerschaltung zum Einsatz [144]. Kontrollsimulationen zeigten, dass die Schaltung vollständig im linearen Bereich arbeitet, so dass die Netztopologie direkt als SystemC-AMS-Modell unter Nutzung der simulatoreigenen linearen Komponenten umgesetzt werden konnte. Die sich anschließende Analog-Digital-Umsetzung wandelt das Messsignal für die digitale Auswertung um.

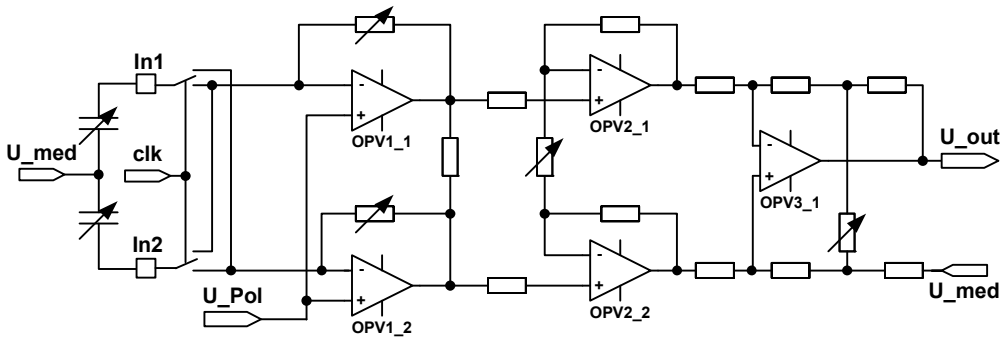


Bild 9.11.: $\Delta C - U$-*Wandlerschaltung*

9.5.2. Digitales Teilsystem

Das digitale Teilsystem bereitet die Messwerte auf und errechnet aus ihnen die aktuelle Position des Systems in seiner Umgebung relativ zum Bewegungsanfang. Zunächst erfolgt eine Fehlerkorrektur der Messsignale durch Fusion der sechs Einzelsensorwerte eines Arrays, die die Kompensation von Offsetparametern ermöglicht. Nach der zweifachen Integration der Beschleunigungsdaten unter Nutzung des Simpson-Algorithmus sind die erhaltenen Wegdaten vom Sensorkoordinatensystem in das Umweltkoordinatensystem zu transferieren. Dies erfolgt unter Nutzung von Quaternionen [145] und wird in [146] näher erläutert.

Die schwierig zu implementierenden Digitalalgorithmen führten zur Untersuchung der Möglichkeit, Neuronale Netze einzusetzen [147]. Diese können über Trainingssequenzen angelernt werden und setzen dann relativ robust den erlernten Algorithmus um. Dies scheiterte jedoch an der notwendigen Größe der Digitallogik, welche die verfügbaren FPGA-Ressourcen überstieg. Darüber hinaus bestand die Notwendigkeit, die fehlerbehafteten inertialen Messwerte durch Stützstelleninformationen zu korrigieren. Hierzu fand eine Untersuchung verschiedener Korrekturalgorithmen statt [148], welche die Positionsergebnisse deutlich verbesserten. Zusätzlich erwies sich die Implementierung eines Umgebungsmodells als hilfreich [149], das durch Angabe von Positionsinformationen zu Wänden und Türen den Standort des INS korrigieren half. Nach einem prototypischen Aufbau mit 4 Einzel-FPGA wurde ein optimiertes Digitaldesign in einen Xilinx-Spartan3-4000-FPGA erfolgreich implementiert [150].

9.5.3. Gesamtsystem

Das aus der Spezifikation erstellte einfache SystemC-AMS-Systemmodell wurde durch die in den letzten beiden Abschnitten genannten Komponenten verfeinert, was eine schnelle Simulation des Verhaltens ermöglicht und die Einzelentwickler in die Lage versetzt, ihre konkretisierte Komponente im Gesamtsystemkontext simulieren zu können. Dabei sind jedoch die Grenzen zu beachten, die SystemC-AMS mit sich bringt, so dass teilweise ein Übergang nach VHDL-AMS, wie in Abschnitt 9.7 beschrieben, notwendig ist. Dies betrifft insbesondere den Bereich der Einzelsensoren im Beschleunigungssensorarray, wo das SystemC-AMS-Modell durch eine Parallelmodellierung mit VHDL-AMS unter Nutzung von Ergebnissen der FEM-Simulation verifiziert wurde.

Einen Überblick über das verfeinerte Systemmodell gibt ein Artikel zur Modellierung von MEMS mit SystemC-AMS [151]. Die Einzelkomponenten des heterogenen INS sind derzeit nicht in einen gemeinsamen Chip integrierbar. Zur Verbindung der Module entstanden daher mehrere Leiterplatten [152], deren Aufbau Bild 9.12 zeigt.

9.6. Bewertung von Implementierungsvarianten

Zur Unterstützung der Untersuchung des Entwurfsraums wurde das in Kapitel 7 erläuterte Verfahren beim INS eingesetzt, um für das kapazitive Beschleunigungssensorarray eine optimale analoge Auswerteschaltung zu finden [128]. Die prinzipiellen Auswerteschaltungen sind bekannt, jedoch muss noch die Anzahl und Zusammenschaltung dieser Komponenten in Erfahrung gebracht werden. Es stehen drei verschiedene Realisierungsvarianten zur Auswahl:

A Pro Einzelsensor wird ein $\Delta C - U$-Wandler und ein eigener ADU implementiert.

B Für das gesamte Array wird ein einzelner $\Delta C - U$-Wandler mit ADU im Zeitmultiplex genutzt, direkt am Eingang ist ein Multiplexer für das Umschalten zwischen den Einzelsensoren einzusetzen.

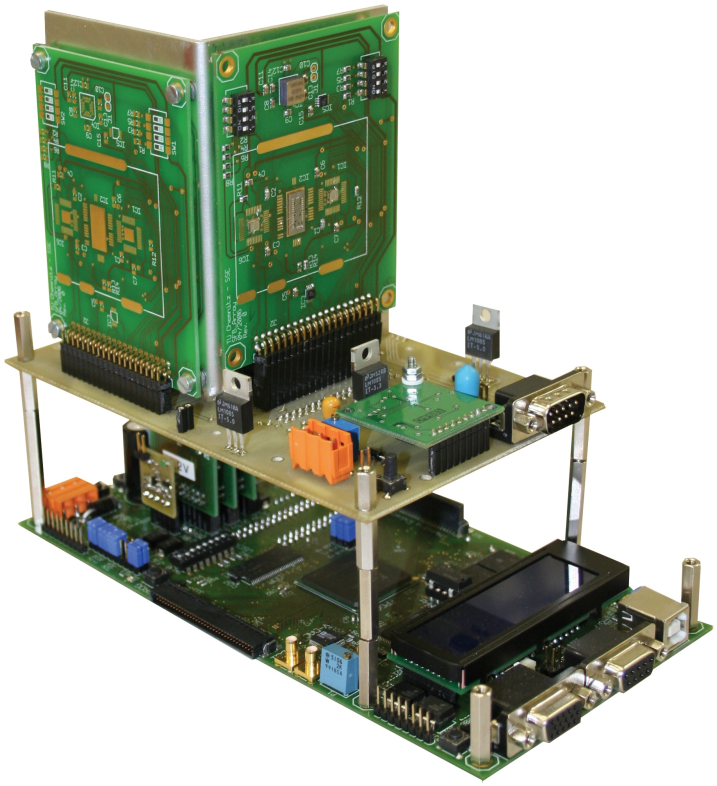


Bild 9.12.: *Foto des UBAS-Aufbaus*

C Es wird für jeden Einzelsensor ein eigener $\Delta C - U$-Wandler implementiert. Diese übergeben ihre Werte jedoch über einen Multiplexer an einen einzelnen ADU.

Für jedes Systemelement werden drei Kostenwerte berücksichtigt: Chipfläche, Verarbeitungszeit und Entwurfszeit. Tabelle 9.1 gibt eine Übersicht über die zugewiesenen Kostenwerte in der jeweiligen Realisierungsvariante.

Die Entwurfszeiten pro Element sind dabei Schätzwerte aus bereits implementierten Designs. Für die Pads werden Bibliothekselemente genutzt, was deren Entwurfsaufwand erheblich reduziert. Für die Berechnung des globalen Gütemaßes zum Vergleich der Realisierungsvarianten wird die Verarbeitungszeit übergewichtet, da diese maßgeblich für die spätere Systemgeschwindigkeit ist. Die benötigte Chipfläche hat bis zu einer bestimmten Grenze ($6{,}25\,\mathrm{mm}^2$) kaum Einfluss auf die Herstellungskosten, daher wird sie hier stark untergewichtet. Sie spielt jedoch als absolute Grenze eine wichtige Rolle als k.o.-Kriterium für die Realisierbarkeit. Folgende Parameter werden für die Systembeurteilung verwendet:

Grenzen: Fläche $6{,}25\,\mathrm{mm}^2$; Verarbeitungszeit: 3,5 s; Entwurfszeit: 7500 Zeiteinheiten
Gewichte: Fläche $0,01$; Verarbeitungszeit 10; Entwurfszeit $0,001$

Die Anwendung der Kostenmodellierung auf die Systemvarianten A – C erzeugt die in den Listings 9.3 bis 9.5 angegebenen Bildschirmausgaben.

```
system consists of:(cost vector: 10.654 2.7 10315 cost limit:
    #6.25 3.5 #7500)
  DeltaC_U  cost vector: 0.252 0.1 300  cost limit: -1 -1 -1
  DeltaC_U  cost vector: 0.252 0.1 300  cost limit: -1 -1 -1
  DeltaC_U  cost vector: 0.252 0.1 300  cost limit: -1 -1 -1
  TriGen    cost vector: 0.049 0 400  cost limit: -1 -1 -1
  Padring   cost vector: 2.964 0 15  cost limit: -1 -1 -1
  ADU  cost vector: 2.295 0.8 3000  cost limit: -1 -1 -1
  ADU  cost vector: 2.295 0.8 3000  cost limit: -1 -1 -1
  ADU  cost vector: 2.295 0.8 3000  cost limit: -1 -1 -1
WARNING: 1. cost value in module sys_ana1 exceeds cost limit!
WARNING: 3. cost value in module sys_ana1 exceeds cost limit!
WARNING: Design is NOT realisable with this configuration!
Weighted scaled costs for System1 7.446993
Check_System: Design is NOT realisable
```

Listing 9.3: *Variante A*

```
system consists of:(cost vector: 4.674 3.2 6610 cost limit:
   6.25 3.5 7500)
  DeltaC_U  cost vector: 0.252 0.4 1000  cost limit: -1 -1 -1
  TriGen  cost vector: 0.049 0 400  cost limit: -1 -1 -1
  MUX3  cost vector: 0.01 0.15 100  cost limit: -1 -1 -1
  MUX3  cost vector: 0.01 0.15 100  cost limit: -1 -1 -1
  Padring  cost vector: 2.058 0 10  cost limit: -1 -1 -1
  ADU  cost vector: 2.295 2.5 5000  cost limit: -1 -1 -1

 Weighted scaled costs for System1 9.151217
 Check_System: Design is realisable
```

Listing 9.4: *Variante B*

```
system consists of:(cost vector: 5.168 3.1 6410  cost limit:
   6.25 3.5 7500 )
  DeltaC_U  cost vector: 0.252 0.1 300  cost limit: -1 -1 -1
  DeltaC_U  cost vector: 0.252 0.1 300  cost limit: -1 -1 -1
  DeltaC_U  cost vector: 0.252 0.1 300  cost limit: -1 -1 -1
  TriGen  cost vector: 0.049 0 400  cost limit: -1 -1 -1
  MUX3  cost vector: 0.01 0.3 100  cost limit: -1 -1 -1
  Padring  cost vector: 2.058 0 10  cost limit: -1 -1 -1
  ADU  cost vector: 2.295 2.5 5000  cost limit: -1 -1 -1

 Weighted scaled costs for System1 8.866266
 Check_System: Design is realisable
```

Listing 9.5: *Variante C*

Element	Variante	Anzahl Elemente	Fläche pro Element $[mm^2]$	Verarb.-zeit für 3 Werte $[\mu s]$	Entwurfszeit pro Element [Zeiteinheiten]
Analog-Digital-Umsetzer	A	3	2,295	0,8	3000
	B, C	1	2,295	2,5	5000
$\Delta C - U$-Wandler	A, C	3	0,252	0,1	300
	B	1	0,252	0,4	1000
Multiplexer	A	-	-	-	-
	B	2	0,010	0,3	100
	C	1	0,010	0,3	100
Padring	A	60 Pads	2,964	0	15
	B, C	42 Pads	2,058	0	10
Spannungserzeugung	A - C	1	0,049	0	400
globale Verdrahtung	A	-	0,050	0	600
	B	-	0,025	0	250
	C	-	0,030	0	300

Tabelle 9.1.: *Kostenwerte der Einzelkomponenten*

Variante A scheidet wegen der Kostenüberschreitung bei den Kostenwerten „Fläche“ und „Entwurfszeit“ aus. Dies wird in der Auflistung der Kostenwerte durch ein Doppelkreuz sowie als separate Warnung angezeigt. Sowohl Variante C als auch Variante B erfüllen die gesetzten Grenzen. Variante C ist im gegebenen Anwendungsfall zu bevorzugen, da sie bei Einhaltung der Grenzen das globale Gütemaß minimiert.

9.7. Export von SystemC-AMS nach VHDL-AMS

Die in Abschnitt 9.5 erfolgte Konkretisierung des Systems in SystemC-AMS ermöglicht eine schnelle Systemsimulation. Um die analogen Einzelkomponenten weiter verfeinern zu können, bietet sich ein Übergang nach VHDL-AMS an. Die Übersetzung der Digitalkomponenten nach VHDL ermöglicht eine Synthese mit Standard-Entwurfswerkzeugen, da die Digitalsynthese von SystemC nur für ein Subset implementiert wurde [114]. Die Analogmodellierung in VHDL-AMS bietet gegenüber SystemC-AMS die Möglichkeit, implizit vom Simulator zu lösende Differentialgleichungssysteme anzugeben. Dies ist jedoch nur für einzelne Komponenten sinnvoll, da die Simulationszeit des Gesamtsystems mit VHDL-AMS im Fall des INS bei etwa 10 Stunden pro Sekunde Realzeit liegt.

Exemplarisch sollen im Folgenden je eine Digital- und eine Analogkomponente des INS von SystemC-AMS nach VHDL-AMS übertragen werden. Der Konverter ermöglicht auch die Übertragung von hierarchischen Modulen, dies wurde am Beispiel der OPV-Schaltung des $\Delta C - U$-Wandlers gezeigt [116]. Die Hauptschwierigkeit lag hierbei in der Beschaffung der Strukturinformationen der hierarchisch untergeordneten Komponenten für die VHDL-COMPONENTs-Definition im Elternmodul.

Digitales Modul: Quaternionentransformation

Das Modul zur Quaternionentransformation konvertiert die Sensordaten aus dem Sensorkoordinatensystem in das Umweltkoordinatensystem, um die Positionsinformationen auf einer Karte nachvollziehen zu können. Die Eingangswerte liegen dabei als 16 bit breite Vektoren vor, diese Beschleunigungen der drei auf den Sensor bezogenen Raumachsen berechnet die sich an den ADU anschließende Eingangsfehlerkorrektur. Die Quaternionentransformation wandelt diese Beschleunigungen durch Integration und Drehung in Positionswerte des Umweltkoordinatensystems.

Das Modul dient als Beispiel für ein rein digitales Modul, daher ist die Nutzung von „shared variables“ zulässig, solange ein Digitalsimulator genutzt wird. Tabelle 9.2 beschränkt sich aus Platzgründen auf den Bereich der Schnittstellen- und Variablendefinitionen und zeigt den Originalcode in SystemC und den daraus automatisch generierten VHDL-Code. Die zugehörige Verhaltensbeschreibung im Rahmen der SystemC-Methode „data_processing“ wurde ebenfalls nach VHDL gewandelt und dort erfolgreich simuliert und synthetisiert.

SystemC	VHDL

SC_MODULE(quat_trafo){
sc_in<sc_bv<16>>ax, ay, az;
sc_out<sc_bv<32>>posx, posy, posz;
sc_in <sc_bit>clk_fpga;
int integ_count;
 	
void data_processing();
 	
SC_CTOR(quat_trafo){
 	
SC_METHOD(data_processing);
 	
sensitive_pos <<clk_fpga;}
 	
}
	ENTITY quat_trafo IS
	PORT (
	SIGNAL ax: IN bit_vector(15 downto 0);
	SIGNAL ay: IN bit_vector(15 downto 0);
	SIGNAL az: IN bit_vector(15 downto 0);
	[...]
	SIGNAL clk_fpga: IN bit);
	END;
	
	ARCHITECTURE rtl OF quat_trafo IS
	SHARED VARIABLE integ_count: integer;
	data_processing: PROCESS (clk_fpga)
	BEGIN
	IF rising_edge(clk_fpga) THEN
	[...]
	END;

Tabelle 9.2.: *Konvertierung eines Digitalmoduls nach VHDL*

Analoges Datenflussmodul OPV_gain

Diese Komponente beschreibt in einfacher Form eine Differenzverstärkung mit Begrenzung durch die Betriebsspannung *vdd* bzw. *vss* mit der Verstärkung *gain*. Im Rahmen des Modells des $\Delta C-U$-Wandlers wird der OPV für die Differenzverstärkung der drei

Stufen genutzt. Er ist dabei in die Umgebung der linearen Netzwerke durch Wrapper eingebunden, so dass die Schleifenverstärkung der OPV-Schaltung wie in Realität durch Widerstände einstellbar ist.

Tabelle 9.3 beschränkt die Darstellung der Komponente auf die der Verhaltensbeschreibung dienende Funktion *sig_proc*(), der umgebende Code entspricht, bis auf die Zeitkontinuität der SDF-Signale, der in Tabelle 9.2 dargestellten Schnittstellenbeschreibung des Digitalmoduls Quaternionentransformation.

SystemC-AMS	VHDL-AMS

void opv_gain::sig_proc(){
double internal_val, inposval, innegval;
inposval = in_pos.read();
innegval = in_neg.read();
internal_val =
(inposval - innegval) * gain;
if (internal_val <vss){
out_port.write(vss);
return;}
if (internal_val >vdd){
out_port.write(vdd);
return;}
out_port.write(internal_val);
}
	PROCEDURE sig_proc(
	SIGNAL in_pos: IN REAL;
	SIGNAL in_neg: IN REAL;
	SIGNAL out_port: OUT REAL) IS
	VARIABLE internal_val: REAL;
	VARIABLE inposval: REAL;
	VARIABLE innegval: REAL;
	BEGIN
	inposval := in_pos;
	innegval := in_neg;
	internal_val :=
	(inposval - innegval) * gain;
	IF internal_val <vss THEN
	out_port <= vss;
	RETURN;
	END IF;
	IF internal_val >vdd THEN
	out_port <= vdd;
	RETURN;
	END IF;
	out_port <= internal_val;
	END;

Tabelle 9.3.: *Konvertierung eines Datenflussmoduls nach VHDL-AMS*

9.8. Konkretisierung der Systemparameter

Die simulativ bestimmten bzw. auf Systemebene abgeschätzten Parameter können durch die Verfeinerung der Modelle bis hin zum differentialgleichungsbasierten VHDL-AMS-Modell der späteren Realisierung immer weiter angenähert werden. Jede Konkretisierung erfordert eine Prüfung, ob sich diese noch im Rahmen der in der initialen Spezifikation angegebenen Zielparameter des Systems bewegen. Gerade im Analogbereich sind viele Parameter zu beachten, die schon bei kleinen Abweichungen zu Fehlfunktionen führen können. Im dargestellten Beispiel UBAS bewegten sich beispielsweise die zu detektierenden Kapazitätsänderungen im Femtofaradbereich, so dass beim Entwurf

der Auswerteelektronik intensive Vorkehrungen zur Vermeidung parasitärer Effekte notwendig waren.

Die im Laufe des Entwurfsprozesses gewonnenen Parameterinformationen fließen in das Systemmodell ein und ermöglichen so ein dem jeweiligen Implementierungsstand der Komponenten entsprechendes Simulationsergebnis. So konnten beim Entwurf des UBAS das anfänglich nur grob durch hybride Automaten beschriebene Verhalten des Beschleunigungssensors immer weiter an die Realität angepasst werden, was schließlich zu einem implementierungsnahen Sensormodell sowie einem daran angepassten Modell der Auswerteschaltung führte. Mit Hilfe dieses Modells konnte die Analog-Digital-Umsetzung dimensioniert und die digitale Auswertung bereits frühzeitig an die analoge Hardware angepasst werden.

9.9. Erfahrungen aus dem Beispiel

Das Beispiel UBAS demonstriert die Anwendung des in Kapitel 8 dargestellten Entwurfsablaufs auf ein komplexes Mikrosystem. Die Teile der vorgestellten Toolkette sind einsatzbereit, jedoch zeigten sich im Bereich der formalen Prüfung der hybriden Automaten noch deutliche Komplexitätsprobleme. Dieser Teil der Toolkette benötigt eine weitergehende Beobachtung der auf dem Markt verfügbaren Modelchecker, um die dargestellten Werkzeuge HybridSAL bzw. HyTech durch für den Mikrosystementwurf geeignetere Software zu ersetzen.

Die in Kapitel 5 gezeigten analogen Erweiterungen von SpecScribe erfüllten die Anforderungen im Rahmen des UBAS-Beispiels. Alle Punkte der Spezifikation konnten in die Datenbank aufgenommen werden, die automatisierte Erstellung eines ersten abstrakten Systemmodells in SystemC-AMS war erfolgreich. Die sich anschließende Verfeinerung der Implementierung führte jedoch zu einem Pendeln zwischen den Sprachen VHDL-AMS und SystemC-AMS, hier sind noch Untersuchungen zu einer besseren Einkopplung kritischer Komponentenmodelle in das Systemmodell nötig. Die Codeumsetzung selbst erfolgte für Standardkonstrukte problemlos, Spezialkonstrukte (siehe Tabelle 6.1) erfordern jedoch noch eine manuelle Nachbehandlung. Mit der Verfügbarkeit von SystemC-AMS-Simulatoren der Version 1.0 sind hier Anpassungsarbeiten nötig, die auch den Aufwand zur manuellen Nachbearbeitung verringern sollen.

10. Zusammenfassung und Ausblick

10.1. Zusammenfassung

Dem Entwurf von Mikrosystemen steht eine rasante fertigungstechnologische Entwicklung gegenüber. Die sich bietenden Entwurfschancen führen zu hohen Designkomplexitäten, die mit bisherigen Mitteln nur unter großem Personalaufwand noch ausnutzbar sind. Die Werkzeuge der zu Beginn dieser Arbeit geschilderten Entwurfsebenen und –formen müssen für die neuen Technologien und Komplexitäten angepasst werden. Dabei stellt die Heterogenität von Mikrosystemen besondere Anforderungen an die Entwurfswerkzeuge, da neben Software und digitaler Hardware auch analoge elektrische und nichtelektrische Komponenten zum Einsatz kommen, die darüber hinaus Wechselwirkungen zu weiteren Domänen, wie z. B. der thermischen Domäne, aufweisen.

Die Dissertationsschrift stellt eine Toolkette zum abstrakten Entwurf von Mikrosystemen vor. Dabei wird ausgehend von der ersten, meist textuellen, Spezifikation der Designprozess für Mikrosysteme erläutert und um neue Verknüpfungen bereichert. Die Sammlung der Spezifikationsdaten in einer Entwurfsdatenbank als Teil von SpecScribe erlaubt eine zentrale Verwaltung und einen gemeinsamen Zugriff des Entwerferteams. Die Formalisierung der textuellen Beschreibungen der Anforderungen bzw. deren Ergänzung um andere Beschreibungsmöglichkeiten wie Bilder oder Tondokumente ermöglicht eine rechnerunterstützte Verarbeitung und damit eine Unterstützung des Designteams.

Die Voraussetzung für die Formalisierung der Daten ist die Entwicklung geeigneter Datenstrukturen zur Aufnahme der Parameter und Algorithmen. Die dazu nötige Analyse der bei Mikrosystemen notwendigen Konstrukte aus den verschiedenen Domänen ergab eine Anforderungssammlung, die auf Systemebene derzeit noch von keiner Sprache voll unterstützt wird. SystemC-AMS erfüllt jedoch bis auf die Darstellung von Ortsabhängigkeiten alle Voraussetzungen für eine abstrakte Modellierung, daher bildet diese Sprache neben der Anforderungsdatenbank und der damit verbundenen Plattform zur Anforderungsüberwachung den Kern dieser Arbeit.

Die Formalisierung analoger Parameter stellt eine notwendige Voraussetzung für eine Konsistenzprüfung der Spezifikation von Mikrosystemen dar. Die vorliegende Arbeit entwickelt dazu geeignete Konstrukte zur Aufnahme derartiger Spezifikationsdaten und integriert sie zusammen mit digitalen Konstrukten in eine gemeinsame Plattform zur Spezifikation und Entwurfsunterstützung namens „SpecScribe“. Dieses Werkzeug bietet neben der Entwurfsdatenbank Mechanismen zur Überwachung der Anforderungen

sowie Schnittstellen zu Hardwarebeschreibungssprachen und weiteren Entwurfswerkzeugen. Im Rahmen der Dissertation werden dazu die Schnittstellen zu hybriden Modelcheckern und zu SystemC-AMS entwickelt und genutzt, ein Export des digitalen Spezifikationsteils nach VHDL ist ebenfalls möglich.

Das aus der formalisierten Spezifikation erzeugte SystemC-AMS-Modell fungiert als „ausführbare Spezifikation“ und ermöglicht so erste Simulationen bereits auf Systemebene. Dieses Modell bildet den Ausgangspunkt zur weiteren Verfeinerung bzw. zur schrittweisen Implementierung der Anforderungen in analoge und digitale Hardware sowie Software. Reichen die Modellierungsmöglichkeiten von SystemC-AMS nicht mehr aus, um auf niedrigen Abstraktionsebenen das Verhalten von Komponenten sinnvoll zu beschreiben, erfolgt für diese Komponente der Übergang zu einer geeigneteren Beschreibungsform. Im Analogbereich wurde dies für VHDL-AMS gezeigt, im Digitalbereich ist ein ähnlicher Übergang nach VHDL möglich. Die sich daran anschließenden Arbeiten zur Generierung von Schaltungen auf Transistorebene erfolgen im Digitalbereich mit Hilfe von gängigen Synthesewerkzeugen. Im Analogbereich, insbesondere im Bereich der nichtelektrischen Komponenten, muss dieser Prozess in der Regel manuell unter Mithilfe von FEM-Simulatoren durchgeführt werden.

Im Laufe des Entwurfsprozesses entstehen neue Informationen über das System. Die gewonnenen Parameter und Implementierungsdetails fließen in die Entwurfsdatenbank zurück, entweder durch Konkretisierung vorhandener oder durch Erstellung neuer Parameter. Dieser Prozess erfolgt momentan manuell, eine Automatisierung wird angestrebt.

Im Rahmen dieser Arbeit entstand ein neuer Entwurfsablauf für Mikrosysteme. Dabei wurden die folgenden Teilaufgaben gelöst:

- Analyse von Mikrosystemen und Ableitung von Modellierungsanforderungen
- Untersuchung verschiedener Sprachen zur Beschreibung heterogener Systeme
- Umsetzung eines vorhandenen Konzepts zur Erfassung digitaler Anforderungen in Python für das Werkzeug SpecScribe
- Entwicklung und Umsetzung eines Konzepts zur Erfassung analoger und heterogener Anforderungen in Form von hybriden Automaten, Signalverläufen und Schaltplaninformationen
- Erstellung von Codegeneratoren zu digitalen und AMS-Sprachen für SpecScribe
- Erweiterung eines Codeumsetzers von SystemC nach VHDL für die Konvertierung von SystemC-AMS-Code nach VHDL-AMS
- Entwicklung einer Ergänzung zu SystemC zur Verknüpfung von Kostenparametern im Systementwurf
- Anwendung des entstandenen Designflows auf ein Mikrosystem zur Inertialnavigation

10.2. Ausblick

Die vorliegende Dissertation schildert Werkzeuge und Konzepte zum Entwurf von Mikrosystemen. Die Komplexität solcher Systeme stellt hohe Anforderungen an die Leistungsfähigkeit der Rechentechnik, die insbesondere im Bereich der formalen Prüfung von analogen Eigenschaften an ihre Grenzen stößt. In diesem Bereich sind Anstrengungen zur Optimierung der Prüfalgorithmen sowie zur automatisierten Reduktion der Designkomplexitäten nötig.

Die vorgestellte Toolkette bietet Möglichkeiten zur Unterstützung der Verifikation und des Tests, die in dieser Arbeit nur angerissen wurden. Da die Prüfung der Funktionsfähigkeit eines Systems mit steigender Komplexität immer höhere Kosten und Zeitaufwand verursacht, sollten hier Untersuchungen zur besseren Integration des Tests in den Entwurfsablauf erfolgen. Erste Arbeiten auf Basis von PSL verliefen erfolgversprechend [113].

Die Sammlung der Implementierungsdetails in der Entwurfsdatenbank stellt die Grundlage für die Dokumentation des entstehenden Systems dar. Neben einer Automatisierung der Sammlung bzw. Aktualisierung der gewonnenen Parameter stellt die Dokumentationsgenerierung einen weiteren nützlichen Punkt im Designprozess dar, dessen Realisierung Gegenstand zukünftiger Arbeiten sein sollte. Hier kann auf die Erkenntnisse aus dem Werkzeug ASPECTOR [10] zurückgegriffen werden.

A. Literaturverzeichnis

[1] Moore, Gordon E.: *Cramming more components onto integrated circuits.* Electronics, Vol. 38(8), April 1965

[2] Murmann, M. und Boser, B.: *Closing the gap between analog and digital.* ACM Queue, Vol. 2(1), März 2004

[3] ITRS: *International Technology Roadmap for Semiconductors: Design part.* Techn. Bericht, 2007

[4] *VHDL Language Reference Manual.* IEEE Standard 1076, 2000

[5] *Verilog hardware description language.* IEEE Standard 1364, 1995

[6] Junge, Hans-Dieter und Müller, Germar: *Lexikon Elektrotechnik.* VCH Verlagsgesellschaft, 1994, ISBN 3-527-28154-1

[7] Kories, Ralf und Schmidt-Walter, Heinz: *Taschenbuch der Elektrotechnik.* Verlag Harri Deutsch, 1998, ISBN 3-8171-1563-6

[8] *4th European Framework IT.* EU Call for Participation, September 1996

[9] Mescheder, Ulrich: *Mikrosystemtechnik.* B. G. Teubner Verlag, 2004, ISBN 3-519-16256-3

[10] Barthel, Tom: *Eine Methode zur Spezifikationserfassung heterogener Systeme.* Dissertation, Technische Universität Chemnitz, 1999

[11] Schlegel, Michael: *Mixed-Level-Simulation heterogener Systeme mit VHDL-AMS durch Multi-Architecture-Modellierung.* Dissertation, Technische Universität Chemnitz, 2005

[12] Herrmann, Göran und Müller, Dietmar: *ASIC - Entwurf und Test.* Fachbuchverlag Leipzig im Carl Hanser Verlag, 2004, ISBN 3-446-21709-6

[13] Walker, R. und Thomas, D.: *A Model of Design Representation and Synthesis.* In 22nd Design Automation Conference, 1985

[14] Huss, Sorin A.: *VHDL-AMS Tutorial: Modellrepräsentationen und Rollen im Entwurfsablauf für gemischt analog/digitale Schaltungen.* In ANALOG'02, 2002

[15] Moser, V.; Amann, H. P. und Pellandini, F.: *Behavioural Modelling of Analogue Systems with ABSynth.* Current Issues in Electronic Modeling, Vol. 10, 1997

[16] IABG Industrieanlagen-Betriebsgesellschaft mbH: *V-Modell XT*. Techn. Bericht, Ottobrunn, 2006

[17] Heinkel, Ulrich: *Das V-Modell im Systementwurf*. Vorlesungsskript EDA-Tools, TU Chemnitz, 2009

[18] Gerlach, Gerald und Dötzel, Wolfram: *Einführung in die Mikrosystemtechnik*. Hanser Fachbuchverlag, 2006, ISBN 3-446-22558-7

[19] Swart, Nicholas R.: *A design flow for micromachined electromechanical systems*. IEEE Design and Test of Computers, Vol. 16(4), pp. 39–47, Oktober 1999, ISSN 0740-7475

[20] *The Spice Page*
URL `http://bwrc.eecs.berkeley.edu/Classes/icbook/SPICE/UserGuide`

[21] Mukherjee, Tamal: *MEMS Design and Verification*. In Proceedings International Test Conference, Vol. 1, pp. 681–690, September 2003

[22] Kampe, Jürgen: *Struktursynthese für analoge Systemkomponenten*. Logos Verlag, 2005, ISBN 3-8325-0956-9

[23] Grimm, Christoph; Brame, Florian; Schroll, Rüdiger und Waldschmidt, Klaus: *Top-Down Design analog/digitaler Systeme mit SystemC-AMS*. In 10. GI/ITG/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", pp. 131–140, März 2007, ISBN 978-3-8322-5956-3

[24] Grimm, Christoph und Waldschmidt, Klaus: *Spezifikation analog/digitaler Systeme*. it+ti Informationstechnik und Technische Informatik, Vol. 40(3), pp. 23–26, 1998

[25] ANACAD Electrical Engineering Software: *HDL-A Language and Users Reference Manual*. 1996

[26] *The VeronA website*
URL `http://www.edacentrum.de/verona/`

[27] *The FEST website*
URL `http://www.em.cs.uni-frankfurt.de/cluster/index.php?id=37`

[28] Grabowski, D.; Platte, D.; Hedrich, Lars und Barke, Erich: *Time Constrained Verification of Analog Circuits using Model-Checking Algorithms*. In First Workshop on Formal Verification of Analog Circuits (FAC), pp. 37–52, 2005

[29] Lämmermann, Stefan; Weiss, Roland; Ruf, Jürgen; Kropf, Thomas; Jesser, Alexander; Hedrich, Lars und Rosenstiel, Wolfgang: *An Assertion-Based Verification Methodology for SystemC-AMS Designs*. In 15th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI), pp. 434–435, Okinawa, Japan, März 2009

[30] *The DETAILS website*
URL `http://www.edacentrum.de/ekompass/projekte/details/`

[31] Vachoux, Alain; Grimm, Christoph; Kakerow, Ralf und Meise, Christian: *Embedded Mixed-Signal Systems: New Challenges for Modeling and Simulation.* In IEEE International Symposium on Circuits and Systems (ISCAS2006), Mai 2006

[32] *FZI Karlsruhe: Karlsruher SystemC Parser Suite*
URL `http://www.fzi.de/sim/kascpar.html`

[33] Harold, Elliotte Rusty und Means, W. Scott: *XML in a Nutshell.* 3rd Ed., September 2004, ISBN 978-0-59600-764-5

[34] *The AVACS website*
URL `http://avacs.org`

[35] *ROBUST-Projekthomepage*
URL `http://www.edacentrum.de/projekte/edaclusterforschung/robust.html`

[36] Paul, Oliver: *Microtransducer Operation.* In Korvink, Jan G. und Paul, Oliver, (Eds.), MEMS: a practical guide to design, analysis and application, Kap. 1, William Andrew Publishing, 2006, ISBN 0-8155-1497-2

[37] Menz, Wolfgang; Mohr, Jürgen und Paul, Oliver: *LIGA-Verfahren.* In Mikrosystemtechnik für Ingenieure, Kap. 8, Wiley-VCH Verlag GmbH & Co. KGaA, 2005, ISBN 3-527-30536-X

[38] Pelesko, John A. und Bernstein, David H., David: *Modeling MEMS and NEMS.* Chapman and Hall/CRC, 2003, ISBN 1-58488-306-5

[39] Walther, Eduard et al.: *Technische Formeln.* Fachbuchverlag Leipzig, 1991, ISBN 3-343-00249-6

[40] Nguyen, Nam-Trung: *Mikrofluidik.* Dissertation, Technische Universität Chemnitz, 2004

[41] Lyshevski, Sergey Edward: *Nano- and Micro-Electromechanical Systems.* CRC Press, 2005, ISBN 0-8493-2838-1

[42] Ahn, Yongchul und Guckel, Henry: *Thermoelastic effect of silicon for strain sensing.* Journal of Micromechanics and Microengineering, Vol. 11(5), pp. 443–451, 2001
URL `http://stacks.iop.org/0960-1317/11/443`

[43] Sun, Changsen; Wang, Liqin; Wang, Yu und Lin, Junxiu: *Design of high-sensitivity photoelastic optical fiber pressure sensor: a differential approach.* Photonics Technology Letters, IEEE, Vol. 9(7), pp. 976–978, Juli 1997, ISSN 1041-1135

[44] Wagemann, Hans-Günther und Schmidt, Andreas: *Grundlagen der optoelektronischen Halbleiterbauelemente.* B. G. Teubner Verlag, 1997, ISBN 3-519-03240-6

[45] Jäckle, Josef: *Über die Ursache der Thermospannung*. Techn. Bericht, Universität Konstanz, Fakultät für Physik, Juli 1998

[46] Lueken, Heiko: *Grundlagen der optoelektronischen Halbleiterbauelemente*. B. G. Teubner Verlag Leipzig/Stuttgart, 1999, ISBN 3-519-03530-8

[47] Zhang, Peng-Gang und Irvine-Halliday, Dave: *Faraday effect optical current sensor*. In Canadian Conference on Electrical and Computer Engineering, Vol. 2, pp. 871–875, Mai 1996

[48] Benedix, Roland: *Bauchemie - Einführung in die Chemie für Bauingenieure*. Teubner Verlag, ISBN 3-8348-0584-3

[49] van't Hoff, Jacobus Henricus: *The role of osmotic pressure in the analogy between solution and gases*. The London, Edinburgh, and Dublin Philosophical magazine and journal of science, August 1888

[50] *Die Chemieseite*
URL `http://www.chemieseite.de`

[51] Albrecht, Steffen; Brandl, Herbert und Zimmermann, Thomas: *Chemolumineszenz*. Hüthig Verlag, 1996, ISBN 3-7785-2501-8

[52] Bruno, P.; Suzuki, Y. und Chappert, C.: *Magneto-optical Kerr effect in a paramagnetic overlayer on a ferromagnetic substrate: A spin-polarized quantum size effect*. Physical Review B, Vol. 53(14), pp. 9214–9220, April 1996, doi: 10.1103/PhysRevB.53.9214

[53] Friedman, L.R.; Soref, R.A. und Khurgin, J.B.;: *Linear and quadratic electrooptic effects in symmetric and asymmetric quantum-well structures*. IEEE Journal of Quantum Electronics, Vol. 31(2), pp. 219–227, Februar 1995

[54] Pagel, Gajus: *Extremale Steuerstrategien für Sonnensegler am Beispiel von Bahntransferproblemen zum Erdmond*. Dissertation, TU Berlin, 2002

[55] Miklós, András; Hess, Peter und Bózoki, Zoltán: *Application of acoustic resonators in photoacoustic trace gas analysis and metrology*. Review of Scientific Instruments, Vol. 72(4), April 2001

[56] Enz, U.; Lems, W.; Metselaar, R.; Rijnierse, P. J. und Teale, R.W.: *Photomagnetic Effects*. IEEE Transactions on Magnetics, Vol. MAG-5(3), pp. 467–472, September 1969

[57] Herve, Yannick: *VHDL-AMS: Anwendungen und industrieller Einsatz*. Oldenbourg Wissenschaftsverlag, 2006, ISBN 978-3-48657-787-7

[58] Hoppe, Peter: *Übertragungsverhalten analoger Schaltungen*. B. G. Teubner Verlag, 1994, ISBN 3-519-06169-4

[59] Alur, Rajeev und Dill, David L.: *Automata for Modelling Real-Time Systems*. In Paterson, M. S., Editor, Proceedings of the 17th International Colloquium on

Automata, Languages and Programming (ICALP 1990), pp. 332–335, Springer Verlag, 1990

[60] Beyer, Dirk: *Formale Verifikation von Realzeit-Systemen mittels Cottbus Timed Automatas*. Dissertation, Brandenburgische Technische Universität Cottbus, 2002

[61] Henzinger, Thomas A.: *The Theory of Hybrid Automata*. In Eleventh Annual IEEE Symposium on Logic in Computer Science, pp. 278–292, 1996

[62] Atterer, Richard: *Hybride Automaten*. Seminararbeit, TU München, Juni 2001

[63] Henzinger, Thomas A.; Ho, Pei-Hsin und Wong-Toi, Howard: *HYTECH: A Model Checker for Hybrid Systems*. International Journal on Software Tools for Technology Transfer, Vol. 1(1–2), pp. 110–122, 1997

[64] Henzinger, Thomas A.; Horowitz, Benjamin; Majumdar, Rupak und Wong-toi, Howard: *Beyond HYTECH: Hybrid systems analysis using interval numerical methods*. In Hybrid Systems Computation and Control - HSCC, pp. 130–144, Springer, 2000

[65] Tiwari, Ashish: *HybridSAL: Modeling and Abstracting Hybrid Systems*. Techn. Bericht, SRI International, Juni 2003

[66] Carlson, Bjorn und Gupta, Vineet: *Hybrid CC and interval constraints*. In Henzinger, Thomas A und Sastry, Sankar, (Eds.), Hybrid Systems 98: Computation and Control, Lecture notes in computer science, Vol 1386, pp. 80–94, Springer Verlag, April 1998

[67] Deshpande, Akash; Göllü, Aleks und Semenzato, Luigi: *The SHIFT Programming Language for Dynamic Networks of Hybrid Automata*. In IEEE Transactions on Automatic Control, Vol. 43, pp. 584–587, April 1998

[68] Bjørner, Nikolaj S.; Browne, Anca; Colón, Michael A; Finkbeiner, Bernd; Z., Manna; Sipma, Henny B. und T.E., Uribe: *Verifying temporal properties of reactive systems: A step tutorial*. In Formal Methods in System Design, 1999

[69] Stauner, Thomas Markus: *Systematic Development of Hybrid Systems*. Dissertation, Technische Universität München, 2001

[70] Romberg, Jan und Grimm, Christoph: *Refinement of hybrid systems: from formal models to design languages*. In Languages for system specification: Selected contributions on UML, systemC, system Verilog, mixed-signal systems, and property specification from FDL'03, pp. 315–330, Kluwer Academic Publishers, Norwell, MA, USA, 2004, ISBN 1-4020-7990-7

[71] Podelski, Andreas und Wagner, Silke: *Model checking of hybrid systems: From reachability towards stability.* In Hybrid Systems: Computation and Control, volume 3927 of LNCS, Springer, 2006

[72] Steinhorst, Sebastian und Hedrich, Lars: *Model Checking of Analog Systems using an Analog Specification Language.* In DATE'08, März 2008, ISBN 3-9810801-3-1

[73] Maler, Oded und Pnueli, A.: *Extending PSL for Analog Circuits.* Techn. Bericht, PROSYD Deliverable D 1.3/1, 2005
URL `http://www.prosyd.org/twiki/pub/Public/DeliverablePageWP1/Prosyd_d1.3_1.pd`

[74] Mehner, Jan: *Entwurf in der Mikrosystemtechnik.* Habilitation, Technische Universität Chemnitz, 2000

[75] Schlegel, Michael; Bennini, Fouad; Mehner, Jan; Herrmann, Göran; Müller, Dietmar und Dötzel, Wolfram: *Analyzing and Simulation of MEMS in VHDL-AMS Based on Reduced Order FE-Models.* IEEE Sensors Journal, Vol. 5(5), Oktober 2005

[76] Accelera: *Verilog-AMS Language Reference Manual.* November 2004
URL `http://www.verilog-ams.com`

[77] *VHDL Analog and Mixed-Signal Extensions.* IEEE Standard 1076.1, 1999

[78] Einwich, Karsten et al.: *Manual for SystemC-AMS*
URL `http://www.systemc-ams.org`

[79] Modelica Association: *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling.* Februar 2005
URL `http://www.modelica.org/`

[80] Kundert, Kenneth S. und Zinke, Olaf: *The Designer's Guide to Verilog-AMS.* Kluwer Academic Publishers, 2004, ISBN 1-4020-8044-1

[81] Roßberg, Christian; Markert, Erik; Herrmann, Göran und Heinkel, Ulrich: *Modellierung und Simulation eines Gyroskopes.* In Workshop: Simulation technischer Systeme, Grundlagen und Methoden in Modellbildung und Simulation, März 2009, ISBN 978-3-8167-7981-0

[82] Schwarz, Peter; Clauß, Christoph; Haase, Joachim und Schneider, André: *VHDL-AMS und Modelica - ein Vergleich zweier Modellierungssprachen.* In 15. Symposium Simulationstechnik ASIM2001, pp. 85–94, September 2001

[83] *The OpenModelica website*
URL `http://www.openmodelica.org/`

[84] *The Mosilab website*
URL `http://www.mosilab.de/`

[85] Fritzson, Peter: *Introduction to Object-Oriented Modeling and Simulation with OpenModelica.* Techn. Bericht, Pelab, Linköping University, Sweden, 2006

[86] Abel, Andreas und Nähring, Tobias: *Frequency-Domain Analysis Methods for Modelica Models*. In Modelica Conference, pp. 383–391, The Modelica Association, März 2008

[87] Ashenden, Peter; Peterson, Gregory und Teegarden, Darell: *The system designer's guide to VHDL-AMS*. Morgan Kaufmann, 2003, ISBN 1-55860-749-8

[88] *SystemC Language Reference Manual*. IEEE Standard 1066, 2005

[89] Einwich, Karsten; Bastian, Jens; Clauss, Christoph; Eichler, Uwe und Schneider, Peter: *SystemC-AMS Extension Library for Modeling Conservative Nonlinear Dynamic Systems*. In Forum on Specification and Design Languages (FDL), September 2006

[90] Einwich, Karsten et al.: *Analog Mixed Signal Extensions for SystemC - White paper and proposal for the foundation of an OSCI Working Group*. Techn. Bericht, 2002
URL `http://www.systemc-ams.org`

[91] Einwich, Karsten: *Application of SystemC/SystemC-AMS for the Specification of a Complex Wired Telecommunication System*. In Forum on Specification and Design Languages (FDL), September 2005

[92] Schneider, Peter; Eichler, Uwe; Einwich, Karsten und Schwarz, Peter: *Simulationsgestützter Entwurf von Mess- und Prüfsystemen für Mikrosysteme*. In 10. GMM-Workshop "Methoden und Werkzeuge für den Entwurf von Mikrosystemen", pp. 153–160, Oktober 2004, ISSN 0947-1413

[93] Markert, Erik: *Design of Microsystems using SystemC-AMS*. In C/C++-Based Modelling of Embedded Mixed-Signal Systems, pp. 141–154, Juni 2007

[94] Markert, Erik; Schlegel, Michael; Herrmann, Göran und Müller, Dietmar: *Beschreibung von mechatronischen Systemen mit SystemC-AMS*. In 10. GMM-Workshop "Methoden und Werkzeuge für den Entwurf von Mikrosystemen", pp. 59–64, Oktober 2004, ISSN 0947-1413

[95] Markert, Erik; Schlegel, Michael; Michel, Matteo; Herrmann, Göran und Müller, Dietmar: *Untersuchung der Anwendbarkeit von SystemC-AMS bei der Beschreibung von MEMS*. In 5. GMM/GI/ITG-Workshop "Multi-Nature Systems", pp. 13–18, Februar 2005

[96] Schlegel, Michael; Herrmann, Göran und Mueller, Dietmar: *Eine neue Hardware-Komponente zur Fuzzy-Pattern-Klassifikation*. In Dresdner Arbeitstagung Systementwurf, pp. 21–26, 2004

[97] Einwich, Karsten; Vachoux, Alain; Grimm, Christoph und Barnasconi, Martin: *Draft Standard SystemC AMS Extensions Language Reference Manual*. Dezember 2008

[98] Grimm, Christoph; Barnasconi, Martin; Vachoux, Alain und Einwich, Karsten: *An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS Extensions*. Juni 2008
URL `http://www.systemc-ams.org`

[99] Wasson, Charles S.: *System Analysis, Design, and Development: concepts, principles, and practices*. Wiley-Interscience, 2006, ISBN 978-0-471-39333-7

[100] *IBM - Telelogic DOORS*
URL `http://www.telelogic.com/products/doors/`

[101] Proß, Uwe; Markert, Erik; Langer, Jan; Richter, Andreas; Drechsler, Chris und Heinkel, Ulrich: *A Platform for Requirement Based Formal Specification (short paper)*. In Forum on Specification and Design Languages (FDL), pp. 237–238, September 2008, ISBN 978-1-4244-2266-1

[102] Proß, Uwe; Richter, Andreas; Langer, Jan; Markert, Erik und Heinkel, Ulrich: *SpecScribe - Specification data capture, analysis and exploitation*. In Software Demonstration at DATE´08 University Booth, März 2008

[103] Schneider, Axel; Bluhm, Thomas; Renner, Tobias; Heinkel, Ulrich; Knäblein, Joachim und Zavala, Reynaldo: *Formal Verification of Abstract System and Protocol Specifications*. In 30th Annual IEEE/NASA Software Engineering Workshop, 2006

[104] Haas, Werner; Heinkel, Ulrich und Gossens, Stefan: *Sematics of a Formal Specification Language for Advanced Design and Verification of ASICs (ADeVA)*. In 11. E.I.S.-Workshop, 2003

[105] *Python Programming Language - Official Website*
URL `http://www.python.org/`

[106] *Nokia-Trolltech Qt Website*
URL `http://trolltech.com/`

[107] Proß, Uwe; Richter, Andreas und Heinkel, Ulrich: *Plattform zur formalisierten Spezifikationserfassung*. In 8. Chemnitzer Fachtagung Mikromechanik und Mikroelektronik, pp. 152–153, November 2007, ISBN 978-3-00-022168-2

[108] Cutuli, Giuseppe; Imperiale, Francesco; Lissoni, Roberto und Marchese, Mario: *Design failure mode effect analysis (DFMEA)*, U.S. Patent 7035769, 2006

[109] de Moura, L.; Owre, S. und Shankar, N.: *The SAL language manual. SRI-CSL-01-02 (Rev. 2)*. Techn. Bericht, SRI International, August 2003

[110] Markert, Erik; Proß, Uwe und Heinkel, Ulrich: *SpecScribe Analog - A Specification Tool Extension for Heterogeneous Systems (short paper).* In Forum on Specification and Design Languages (FDL), pp. 243–244, September 2008, ISBN 978-1-4244-2266-1

[111] *OneSpin Solutions 360MV website*
URL `http://www.onespin-solutions.com/360mv.php`

[112] Markert, Erik; Proß, Uwe und Tischendorf, Claudia: *Ergebnisbericht MxMobile Teilprojekt TU Chemnitz.* online verfügbar bei TIB Hannover, 2009

[113] Tischendorf, Claudia; Langer, Jan; Proß, Uwe und Heinkel, Ulrich: *Generierung von VHDL-Modellen aus PSL-Eigenschaften.* In Dresdner Arbeitstagung Schaltungs- und Systementwurf DASS, Special Session URANOS, pp. 37–42, Mai 2008, ISBN 3-9810287-2-4

[114] *Synopsys Inc. "Describing Synthesizable RTL in SystemC. (Version 1.1)"*
URL `http://www.synopsys.com`

[115] Kühn, Sven: *Implementation of a RTL SystemC to VHDL converter.* Studienarbeit, Technische Universität Chemnitz, 2003

[116] Markert, Erik; Kühn, Sven; Langer, Jan; Herrmann, Göran und Heinkel, Ulrich: *Ein SystemC-AMS nach VHDL-AMS Konverter.* In 10. GI/ITG/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", pp. 151–160, März 2007, ISBN 978-3-8322-5956-3

[117] Markert, Erik; Dienel, Marco; Herrmann, Göran; Müller, Dietmar und Heinkel, Ulrich: *Modeling of a new 2D Acceleration Sensor Array using SystemC-AMS.* In Journal of Physics: Conference Series, Vol. 34, Mai 2006, ISSN 1742-6596

[118] Parr, Terence P.: *Language Translation using PCCTS and C++.* Automata Publishing Company, 1993, ISBN 0-9627488-5-4

[119] *The ANTLR website*
URL `http://www.antlr.org`

[120] Eles, P.; Peng, Z.; Kuchcinski, K. und Doboli, A.: *System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search.* In Journal on Design Automation for Embedded Systems, Vol. 2, pp. 5–32, 1997

[121] De Bernardis, E.; Nuzzo, P. und Sangiovanni-Vincitelli, A.: *Mixed Signal Design Space Exploration through Analog Platforms.* In Proceedings of Design Automation Conference (DAC), pp. 875–880, Juni 2005

[122] Ragan, Daniel; Sandborn, Peter und Stoaks, Paul: *A detailed cost model for concurrent use with hardware/software co-design.* In DAC '02: Proceedings of the 39th annual Design Automation Conference, pp. 269–274, ACM, New York, NY, USA, 2002, ISBN 1-58113-461-4, doi:http://doi.acm.org/10.1145/513918.513989

[123] Sangeetha, M.; RajaPaul Perinbam, J. und Revathy: *Hardware Estimation and Synthesis for a Codesign System.* In Signal Processing, Communications and Networking, 2007. ICSCN '07. International Conference on, pp. 189–194, Feb. 2007

[124] Kim, Tae-Woo und Shin, Hyunchul: *Hardware cost estimation techniques for C-level description.* In VLSI and CAD, 1999. ICVC '99. 6th International Conference on, pp. 85–88, 1999

[125] Zhao, Yuan; Tan, Hee Beng Kuan und Zhang, Wei: *Software cost estimation through conceptual requirement.* In Quality Software, 2003. Proceedings. Third International Conference on, pp. 141–144, Nov. 2003

[126] Schlegel, Michael; Herrmann, Göran und Müller, Dietmar: *Erweiterte Kostenmodellierung mit VHDL/VHDL- AMS.* In GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Shaker Verlag, Aachen, 2004

[127] Wang, Hailu: *Konvertierung der Kostenmodellierung nach SystemC(-AMS).* Studienarbeit, Technische Universität Chemnitz, Oktober 2005

[128] Markert, Erik; Wang, Hailu; Herrmann, Göran und Heinkel, Ulrich: *Kostenmodellierung mit SystemC/SystemC-AMS.* In Dresdner Arbeitstagung Schaltungs- und Systementwurf DASS, pp. 49–54, Mai 2007, ISBN 978-3-940046-28-4

[129] Stammermann, A. et al.: *ORINOCO: Verlustleistungsanalyse und Optimierung auf der algorithmischen Abstraktionsebene.* In 10. E.I.S.-Workshop, 2001

[130] Lauwers, E. und Gielen, G.: *Power Estimation Methods for Analog Circuits for Architectural Exploration of Integrated Systems.* In IEEE Transactions on VLSI Systems, Vol. 10, 2002

[131] Van der Plas, G.; Vandenbussche, J.; Gielen, G und Sansen, W.: *EsteMate: a Tool for Automated Power and Area Estimation in Analog Top-Down Design and Synthesis.* In IEEE Custom Integrated Circuits Conference, Mai 1997

[132] *Synopsys NanoSim*
URL `http://www.synopsys.com/tools/verification/amsverification/circuitsimulati`

[133] Markert, Erik; Proß, Uwe; Herrmann, Göran und Heinkel, Ulrich: *A top-down methodology for MEMS design.* In Smart Systems Integration Conference and Exhibition, VDE Verlag Berlin, März 2007, ISBN 978-3-8007-3009-4

[134] Zehnder, Carl August: *Informationssysteme und Datenbanken.* vdf, Hochschulverlag an der ETH, 8. Ed., 2005

[135] Markert, Erik; Proß, Uwe und Heinkel, Ulrich: *A Specification environment for MEMS.* In Smart Systems Integration Conference and Exhibition, pp. 484–487, März 2009, ISBN 978-3-89838-616-6

[136] *TLM Transaction Level Modeling Library Release 2.0.* Osci standard, 2008

[137] Siegmund, Robert; Proß, Uwe und Müller, Dietmar: *SVE: A Methodology for the Design of Protocol Dominated Digital Systems.* In Müller, Wolfgang, Rosenstiel, Wolfgang und Ruf, Jürgen, (Eds.), SystemC - Methodologies and Applications, Kluwer Academic Publishers, 2003, ISBN 1-4020-7479-4

[138] Markert, Erik; Herrmann, Göran und Müller, Dietmar: *System Model of an Inertial Navigation System using SystemC-AMS (short paper).* In Forum on Specification and Design Languages (FDL), pp. 73–76, September 2005

[139] Markert, Erik; Herrmann, Göran und Heinkel, Ulrich: *Hybride Automaten zum Systementwurf von Mikrosystemen.* In 8. Chemnitzer Fachtagung Mikromechanik und Mikroelektronik, pp. 73–78, November 2007, ISBN 978-3-00-022168-2

[140] Dienel, Marco; Billep, Detlef und Dötzel, Wolfram: *Development of a drift compensated acceleration sensor array.* In 50. Internationales wissenschaftliches Kolloquium, pp. 227–228, September 2005, ISBN 3-932633-98-9

[141] Markert, Erik; Schlegel, Michael; Dienel, Marco; Herrmann, Göran und Müller, Dietmar: *Modeling of a new acceleration sensor as part of a 2D Sensor Array in VHDL-AMS.* In NSTI Nanotechnology Conference and Trade Show, part 3, pp. 399–402, Mai 2005, ISBN 0-9767985-2-2

[142] Markert, Erik; Herrmann, Göran; Müller, Dietmar und Heinkel, Ulrich: *High-level model of an acceleration sensor with feedback as part of an inertial navigation system.* In 1st International Conference on Sensing Technology, pp. 191–195, November 2005, ISBN 0-473-10504-7

[143] Markert, Erik; Herrmann, Göran und Müller, Dietmar: *SystemC-AMS-Modell eines rückgekoppelten Beschleunigungssensors als Teil eines Inertialnavigationssystems.* In 7. Chemnitzer Fachtagung Mikromechanik und Mikroelektronik, pp. 137–142, Oktober 2005, ISBN 3-00-016889-3

[144] Markert, Erik; Zeun, Hendrik; Herrmann, Göran; Müller, Dietmar und Heinkel, Ulrich: *SystemC-AMS-Modell eines DeltaC-U-Wandlers für ein Inertialnavigationssystem.* In 9. ITG/GMM-Fachtagung Analog'06 "Entwicklung von Analogschaltungen mit CAE-Methoden", pp. 243–248, September 2006, ISBN 3-8007-2988-1

[145] Hart, J. C.; Francis, G. K. und Kauffman, L. H.: *Visualizing Quaternion Rotation.* ACM Transactions on Graphics, Vol. 13(3), pp. 256–276, Juli 1994

[146] Goller, Sebastian; Markert, Erik; Herrmann, Göran; Müller, Dietmar und Heinkel, Ulrich: *Implementierung von Transformations- und Integrationsalgorithmen für eine 3D-inertiale Messeinheit.* In 17. Internationale Wissenschaftliche Konferenz Mittweida, pp. 26–29, November 2005, ISSN 1473-7624

[147] Fross, André; Markert, Erik; Lange, Robert und Heinkel, Ulrich: *Entwicklung einer generischen FPGA-Implementierung Neuronaler Netze.* In Dresdner Arbeitstagung Systementwurf, pp. 47–52, Mai 2006

[148] Wang, Hailu; Markert, Erik; Herrmann, Göran und Heinkel, Ulrich: *Ein Korrekturverfahren für die Inertialnavigation einer Gehmaschine.* In 8. Chemnitzer Fachtagung Mikromechanik und Mikroelektronik, pp. 87–90, November 2007, ISBN 978-3-00-022168-2

[149] Weitz, Martin: *Implementierung einer 3D-Karte für die In-door-Positionsbestimmung einer inertialen Messeinheit.* Diplomarbeit, Technische Universität Chemnitz, Februar 2005

[150] Markert, Erik und Herrmann, Göran: *Ergebnisbericht Teilprojekt A2-Systementwurf.* Abschlussbericht SFB 379 Mikromechanische Sensor- und Aktorarrays, 2006

[151] Markert, Erik; Dienel, Marco; Herrmann, Göran und Heinkel, Ulrich: *SystemC-AMS assisted design of an Inertial Navigation System.* IEEE Sensors Journal - Special Issue on Intelligent Sensors, Vol. 7(5), pp. 770–777, Mai 2007, ISSN 1530-437X

[152] Wolf, Peter; Markert, Erik; Herrmann, Göran und Heinkel, Ulrich: *Eine generische Plattform zur Sensorsignalauswertung.* In 8. Chemnitzer Fachtagung Mikromechanik und Mikroelektronik, pp. 123–124, November 2007, ISBN 978-3-00-022168-2

B. Tabellenverzeichnis

C. Abbildungsverzeichnis

D. Thesen

1. Ein durchgehender Entwurfsfluss für Mikrosysteme ausgehend von der Erfassung der textuellen Spezifikation über die Gesamtsystemsimulation mit Hard- und Softwareanteilen bis hin zur Differentialgleichungsebene existierte bisher nicht.
2. Die steigende Komplexität von Mikrosystemen erfordert neue Beschreibungsmittel auf Systemebene.
3. Mikrosysteme vereinen Einflüsse aller physikalischer Domänen wie Mechanik, Elektrik, Magnetik, Thermodynamik, Chemie und Strahlung.
4. Zur Simulation von Mikrosystemen ist aufgrund der heterogenen Zusammensetzung der Komponenten ein domänenübergreifendes Werkzeug nötig.
5. Hybride Automaten stellen eine Beschreibungsmöglichkeit für heterogene Systeme auf hoher Abstraktionsebene dar.
6. Die Formalisierung von Spezifikationen im Analogbereich ermöglicht eine frühzeitige Prüfung des Systemverhaltens.
7. Bei der Spezifikation von Mikrosystemen sind neben der Erfassung direkter Zeitbezüge auch Differentialgleichungen nötig. Dabei reicht jedoch die Betrachtung als Summe gerichteter Netzwerkpfade aus, eine Lösung des gesamten Differentialgleichungssystems mit entsprechenden Iterationsschritten ist nicht erforderlich.
8. Das Werkzeug SpecScribe ermöglicht die plattformunabhängige Erfassung von Spezifikationen heterogener Systeme.
9. Die Erfassung von Schaltplan- und Layoutdaten in tabellarischer Form beschleunigt den Entwurfsprozess bei der Anpassung vorhandener Schaltungen.
10. Die Entwurfsmethoden des Digitalbereichs bilden die Grundlage für neue Werkzeuge im Analogbereich.
11. SystemC-AMS ermöglicht eine umfassende Modellierung heterogener Systeme auf funktionaler und algorithmischer Ebene.
12. Eine Beschreibung von kompletten Mikrosystemen auf Differentialgleichungsbasis führt zu intolerabel langen Simulations- und Modellentwicklungszeiten.
13. Die Nutzung einer gemeinsamen Datenbank für alle Entwurfsschritte unterstützt die Konsistenz des Designprozesses.

14. Der Einsatz von SystemC-AMS auf Differentialgleichungsebene ist möglich, führt aber zu einem höheren Modellierungsaufwand als bei VHDL-AMS.

15. Die Integration von Kostenparametern in das zur Systemsimulation genutzte Werkzeug vereinfacht die Entwurfsraumuntersuchung und hilft, die Kostenwerte konsistent mit dem jeweiligen Entwurfsstand zu halten.

16. Eine rein additive Verknüpfung von Kostenparametern ist bei der Entwurfsraumuntersuchung nicht ausreichend, die Erweiterung um multiplikative sowie Absolutwertverknüpfungen erlaubt die Berechnung weiterer wichtiger Parameter.

17. Keine der derzeit verfügbaren Hardwarebeschreibungssprachen eignet sich sowohl für die Modellbeschreibung auf Systemebene als auch auf Geometrieebene, im Laufe des Entwurfsprozesses sind Modelltransfers zwischen Sprachen unumgänglich.

18. Die Erfassung der im Laufe des Entwurfsprozesses verfeinerten Systemparameter in der Spezifikationsdatenbank bildet die Grundlage einer automatisierte Erstellung von Systemdokumentationen.

19. Die Simulation hybrider Automaten kann sowohl mit Spezialsimulatoren als auch mit AMS-Sprachen erfolgen.

20. Die verfügbaren Analysewerkzeuge für hybride Automaten sind aus Komplexitätsgründen zur Behandlung ganzer Mikrosysteme noch nicht geeignet, die genutzten Variablen sind auf einen minimal notwendigen Wertebereich zu beschränken.

21. Die neue Methodik unterstützt den Entwurf von Mikrosystemen von der Spezifikation über die Festlegung der Algorithmen bis hin zur Implementierung und zum Test.

E. Liste eigener Veröffentlichungen

begutachtete Veröffentlichungen

1. Markert, E.; Pross, U.; Richter, A.; Heinkel, U.: *A specification environment for MEMS.* Smart Systems Integration 2009, Brüssel, Belgien, 10.-11. März 2009, pp. 484-487, ISBN 978-3-89838-616-6

2. Markert, E.; Pross, U.; Heinkel, U.: *SpecScribe Analog - A Specification Tool Extension for Heterogeneous Systems.* FDL´08 (Forum on Specification and Design Languages), Stuttgart, 23.-25. September 2008, ISBN 978-1-4244-2266-1, pp. 243-244

3. Markert, E.; Herrmann, G.; Heinkel, U.: *Hybride Automaten zum Systementwurf von Mikrosystemen.* 8. Chemnitzer Fachtagung Mikrosystemtechnik, Chemnitz, 14.-15. November 2007, S. 73-78, ISBN 978-3-00-022168-2

4. Markert, E.: *Design of Microsystems using SystemC-AMS.* Workshop „C/C++-Based Modelling of Embedded Mixed-Signal Systems“, Dresden, Juni 2007, pp. 141-154

5. Markert, E.; Dienel, M.; Herrmann, G.; Heinkel, U.: *SystemC-AMS assisted design of an Inertial Navigation System.* IEEE Sensors Journal - Special Issue on Intelligent Sensors, Mai 2007, vol. 7, no. 5, pp. 770, ISSN 1530-437X

6. Markert, E.; Hailu Wang; Herrmann, G.; Heinkel, U.: *Kostenmodellierung mit SystemC/SystemC-AMS.* Dresdner Arbeitstagung Schaltungs- und Systementwurf DASS, Dresden, 8.-9. Mai 2007, S. 49-54, ISBN 978-3-940046-28-4

7. Markert, E.; Pross, U.; Herrmann, G.; Heinkel, U.: *A top-down methodology for MEMS design.* Smart Systems Integration 2007, Paris, France, 27.-28. März, 2007, pp. 545-548, ISBN 978-3-8007-3009-4, VDE Verlag Berlin

8. Markert, E.; Kühn, S.; Langer, J.; Herrmann, G.; Heinkel, U.: *Ein SystemC-AMS nach VHDL-AMS Konverter.* 10. GI/ITG/GMM-Workshop „Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen“, Erlangen, 5.-7. März 2007, S. 151-160, ISBN 978-3-8322-5956-3

9. Markert, E.; Zeun, H.; Herrmann, G.; Müller, D.; Heinkel, U.: *SystemC-AMS-Modell eines DeltaC-U-Wandlers für ein Inertialnavigationssystem.* 9. ITG/GMM-Fachtagung Analog'06 „Entwicklung von Analogschaltungen mit CAE-Methoden“, Dresden, 27.-29. September 2006, pp. 243-248, ISBN 3-8007-2988-1

10. Markert, E.; Dienel, M.; Herrmann, G.; Mueller, D.; Heinkel, U.: *Modeling of a new 2D Acceleration Sensor Array using SystemC-AMS.* International MEMS Conference 2006 (iMEMS 2006), 9.-12. Mai, Biopolis, Singapore; published: Journal of Physics: Conference Series, vol. 34, 2006, ISSN 1742-6596 (online), ISSN 1742-6588 (print)

11. Markert, E.; Herrmann, G.; Müller, D., Heinkel, U.: *High-level model of an acceleration sensor with feedback as part of an inertial navigation system.* 1st International Conference on Sensing Technology (ICST 2005), Palmerston North, New Zealand, 21.-23. November 2005, ISBN 0-473-10504-7, pp.191-195

12. Markert, E.; Herrmann, G.; Müller, D.: *SystemC-AMS-Modell eines rückgekoppelten Beschleunigungssensors als Teil eines Inertialnavigationssystems.* 7. Chemnitzer Fachtagung Mikrosystemtechnik, Chemnitz, 26.-27. Oktober 2005, S. 137-142, ISBN 3-00-016889-3

13. Markert, E.; Dienel, M.; Zeun, H.: *Entwicklung eines Universellen Bewegungsanalysesystems auf Basis inertialer Messwerte.* 7. Chemnitzer Fachtagung Mikrosystemtechnik, Chemnitz, 26.-27. Oktober 2005, S. 199-200, ISBN 3-00-016889-3

14. Markert, E.; Herrmann, G.; Müller, D.: *System Model of ab Inertial Navigation System using SystemC-AMS.* Forum on Specification and Design Languages (FDL) 2005, Lausanne, 27.-30. September 2005, pp. 73-76

15. Markert, E.; Schlegel, M.; Dienel, M.; Herrmann, G.; Müller, D.: *Modeling of a new acceleration sensor as part of a 2D Sensor Array in VHDL-AMS.* 2005 NSTI Nanotechnology Conference and Trade Show, Anaheim CA, USA, 8.-12. Mai 2005, part 3, pp.399-402, ISBN 0-9767985-4-9

16. Markert, E.; Schlegel, M.; Michel, M.; Herrmann, G.; Müller, D.: *Untersuchung der Anwendbarkeit von SystemC-AMS bei der Beschreibung von MEMS.* 5. GMM/ITG/GI-Workshop „Multi-Nature-Systems: Optoelektronische, mechatronische und andere gemischte Systeme“, Dresden, 18. Februar 2005, S. 13 ff.

17. Markert, E.; Schlegel, M.; Herrmann, G.; Müller, D.: *Beschreibung von mechatronischen Systemen mit SystemC-AMS.* 10. GMM-Workshop „Methoden und Werkzeuge für den Entwurf von Mikrosystemen“, Cottbus, 20.-22. Oktober 2004, ISSN 0947-1413, S. 59 ff.

18. Shende, M.; Roßberg, C.; Markert, E.; Herrmann, G.; Heinkel, U.: *Modeling and simulation of a gyroscope using Matlab/Simulink and VHDL-AMS.* 9. Chemnitzer Fachtagung Mikrosystemtechnik, Chemnitz, 5.-6. November 2009

19. Masoumi, M.; Markert, E.; Heinkel, U.; Gielen, G.: *Ultra Low Power Flash ADC for UWB Transceiver Applications.* European Conference on Circuit Theory and Design EECTD, Antalya, Turkey, 23.-27. August 2009, ISBN 978-1-4244-3896-9

20. Roßberg, C.; Markert, E.; Herrmann, G.; Heinkel, U.: *Modellierung und Simulation eines Gyroskopes.* Workshop „Simulation technischer Systeme Grundlagen und Methoden in Modellbildung und Simulation“, Dresden, 5.-6. März 2009, ISBN 978-3-8167-7981-0

21. Pross, U.; Markert, E.; Langer, J.; Richter, A.; Drechsler, C.; Heinkel, U.: *A Platform for Requirement Based Formal Specification.* FDL´08 (Forum on Specification and Design Languages), Stuttgart, 23.-25. September 2008, ISBN 978-1-4244-2266-1, pp. 237-238

22. Pross, U.; Richter, A.; Langer, J.; Markert, E.; Heinkel, U.: *SpecScribe - Specification data capture, analysis and exploitation.* DATE´08, Munich, 10.-14. März 2008, Software Demonstration at DATE´08 University Booth

23. Wolf, P.; Markert, E.; Herrmann, G.; Heinkel, U.: *Eine generische Plattform zur Sensorsignalauswertung.* 8. Chemnitzer Fachtagung Mikrosystemtechnik, Chemnitz, 14.-15. November 2007, S. 123-124, ISBN 978-3-00-022168-2

24. Wang, H.; Markert, E.; Herrmann, G.; Heinkel, U.: *Ein Korrekturverfahren für die Inertialnavigation einer Gehmaschine.* 8. Chemnitzer Fachtagung Mikrosystemtechnik, Chemnitz, 14.-15. November 2007, S. 87-90, ISBN 978-3-00-022168-2

25. Froß, A.; Markert, E.; Lange, R.; Heinkel, U.: *Entwicklung einer generischen FPGA-Implementierung Neuronaler Netze.* Dresdner Arbeitstagung Schaltungs- und Systementwurf DASS'06, Dresden, 10.-11. Mai 2006, S. 47-52

26. Goller, S.; Markert, E.; Herrmann, G.; Müller, D.; Heinkel, U.: *Implementierung von Transformations- und Integrationsalgorithmen für eine 3D-inertiale Messeinheit.* 17. Internationale Wissenschaftliche Konferenz Mittweida, Mittweida, 03.-04. November 2005, Teilband Robotik, ISSN 1473-7624, S. 26-29

nicht begutachtete Veröffentlichungen

1. Markert, E.; Goller, S.; Pross, U.; Schneider, A.; Knäblein, J.; Heinkel, U.: *Ethernet Based In-Service Reconfiguration of SoCs in Telecommunication Networks.* in: Voros, N.; Rosti, A.; Hübner, M. (Eds.): „Dynamic System Reconfiguration in Heterogeneous Platforms - The MORPHEUS Approach.“ Springer Verlag, 2009, ISBN 978-90-481-2427-5, pp. 195-203

2. Markert, E.; Zeun, H.; Herrmann, G.; Müller, D.; Heinkel, U.: *SystemC-AMS-Modell eines DeltaC-U-Wandlers für ein Inertialnavigationssystem.* Workshop „Detektion mit und in Biosystemen“, 10. April 2008, Bio City Leipzig

3. Markert, Erik: *UBAS - Universelles Bewegungsanalysesystem.* Abschlussvortrag Sonderforschungsbereich 379, Teilprojekt A2 Systementwurf, März 2007, SIT - Messe Chemnitz

4. Markert, Erik; Hermann, Göran; Heinkel, Ulrich: *Overview of an inertial navigation system based on acceleration sensor arrays.* Jahresbericht 2006, Zentrum für Mikrotechnologien Chemnitz

5. Markert, Erik; Müller, Dietmar; Heinkel, Ulrich: *Acceleration sensor modelling using SystemC-AMS.* Jahresbericht 2005, Zentrum für Mikrotechnologien Chemnitz

6. Markert, Erik; Schlegel, Michael; Herrmann, Göran; Müller, Dietmar: *Examination of the Applicability of SystemC-AMS for the Description of MEMS.* Jahresbericht 2004, Zentrum für Mikrotechnologien Chemnitz