Mixed-Level-Simulation heterogener Systeme mit VHDL-AMS durch Multi-Architecture-Modellierung

von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität Chemnitz

genehmigte Dissertation zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

vorgelegt von
Diplom-Ingenieur (FH) Michael Schlegel
geboren am 7. Mai 1974 in Annaberg-Buchholz

eingereicht am 28. Februar 2005

Gutachter: **Prof. Dr.-Ing. habil. Dietmar Müller**
Prof. Dr.-Ing. Klaus D. Müller-Glaser
Dr.-Ing. habil. Peter Schwarz

Tag der Verleihung: 4. Oktober 2005

Michael Schlegel

Mixed-Level-Simulation heterogener Systeme mit VHDL-AMS durch Multi-Architecture-Modellierung

Mixed-Level-Simulation heterogener Systeme mit VHDL-AMS durch Multi-Architecture-Modellierung

von der Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität Chemnitz

genehmigte Dissertation
zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

vorgelegt von

Diplom-Ingenieur (FH) Michael Schlegel

Bibliographische Angaben

Mixed-Level-Simulation heterogener Systeme mit VHDL-AMS durch Multi-Architecture-Modellierung

Schlegel, Michael. – 2005 – 183 Seiten;
43 Abbildungen, 20 Tabellen, 23 Quelltextauszüge, 137 Literaturquellen
Chemnitz, Technische Universität, Fakultät für Elektrotechnik und Informationstechnik, Dissertationsschrift

Referat

Die Simulation heterogener Systeme auf hoher Abstraktionsebene gewinnt auf Grund der zunehmenden Komplexität technischer Systeme stetig an Bedeutung. Unter heterogenen Systemen versteht man technische Systeme, die aus analoger und digitaler Elektronik, aus Komponenten verschiedener physikalischer Domänen wie mechanischen Strukturen, thermischen und optischen Komponenten sowie aus Software bestehen können. Genügte es bisher, die einzelnen Komponenten für sich in ihrer eigenen Domäne mit einem speziellen Simulator zu simulieren, so ist es heute unerläßlich, auch die Interaktionen zwischen den Komponenten zu erfassen. Um solche Systeme mit einer einheitlichen Beschreibungsform erfassen zu können, entstand aus der digitalen Hardwarebeschreibungssprache VHDL die Systembeschreibungssprache VHDL-AMS.

Bei der Modellierung eines Systems muß das tatsächliche Verhalten der Komponenten abstrahiert werden, um mathematisch erfaßbar und in begrenzter Zeit simulierbar zu sein. Der Grad der Abstraktion beeinflußt jedoch die Genauigkeit der Simulationsergebnisse wesentlich. Dabei muß bzw. kann das Verhalten in unterschiedlichen Komponenten unterschiedlich stark abstrahiert werden, um noch akzeptable Simulationsgenauigkeiten erzielen zu können. VHDL-AMS erlaubt die Beschreibung von Komponenten auf unterschiedlichen Abstraktionsniveaus. Man kann die unterschiedlich abstrakten Modelle der Komponenten aber nur schwer in einer Systemsimulation gemeinsam simulieren, da unterschiedlich abstrakte Modelle auch unterschiedlich abstrakte Schnittstellen aufweisen, so daß die Modelle nur mühsam miteinander verbunden werden können. Ein Austausch eines abstrakten Modells einer Komponente gegen ein weniger abstraktes Modell oder umgekehrt ist mit vielen fehleranfälligen und zeitaufwendigen Anpassungsschritten verbunden.

Im Rahmen dieser Arbeit wird ein methodischer Ansatz vorgestellt, der es auf der Basis einer Vereinheitlichung der Modellschnittstellen ermöglicht, unterschiedlich abstrakte Modelle gemeinsam zu simulieren und einzelne Modelle gegen abstraktere oder weniger abstrakte Modelle ohne nennenswerten Zeit- und Modellierungsaufwand auszutauschen. Es werden die zu verwendenden Interfaceobjekte und Datentypen für digitale, analoge elektrische und nichtelektrische Schnittstellen unter VHDL-AMS und SystemC-AMS vorgestellt. Ebenso werden Methoden vorgestellt, die digitales, ereignisdiskretes Verhalten auf konservative elektrische Schnittstellen bzw. nichtkonservatives analoges Verhalten auf digitale Schnittstellen abbilden. Weiterhin wird erläutert, wie sich digitale Protokolle über Abstraktionsebenen hinweg übertragen lassen und ein modifizierter Top-Down Design-Flow vorgestellt. Die Demonstration der Anwendbarkeit der Methode erfolgt anhand eines Beispiels.

Schlagworte

Abstraktionsniveau, Entwurfsmethodik, heterogene Systeme, Mikrosysteme, Mixed-Level-Simulation, Schnittstellen, SystemC-AMS, VHDL-AMS

Inhaltsverzeichnis

Verzeichnis der verwendeten Formelzeichen und Symbole

μ	Differenzgröße, z. B. elektrische Spannung
λ	Flußgröße, z. B. elektrischer Strom
h_{min}	kleinster zulässiger Zeitschritt eines analogen Simulators
h_{max}	größter zulässiger Zeitschritt eines analogen Simulators
ε	Genauigkeitsparameter
u, U	elektrische Spannung
i, I	elektrischer Strom
R	elektrischer Widerstand
t	Zeit
R_s	Ausgangswiderstand eines starken Treibers
R_w	Ausgangswiderstand eines schwachen Treibers
R_l	Eingangswiderstand
N_w	Anzahl schwacher Treiber mit Zustand „unbestimmt“ auf einem Signal
N_l	Anzahl schwacher Treiber mit Zustand `'0'` auf einem Signal
N_h	Anzahl schwacher Treiber mit Zustand `'1'` auf einem Signal
N_0	Anzahl starker Treiber mit Zustand `'0'` auf einem Signal
N_1	Anzahl starker Treiber mit Zustand `'1'` auf einem Signal
U_0	Ausgangsspannung Zustand `'0'`
U_1	Ausgangsspannung Zustand `'1'`
U_x	Ausgangsspannung Zustand „unbestimmt“
U_{i0}	maximale Eingangsspannung um den Zustand `'0'` zu erkennen
U_{i1}	minimale Eingangsspannung um den Zustand `'1'` zu erkennen
soll	Sollgröße eines Reglers
ist	Istgröße eines Reglers
error	Regelabweichung
regel	Reglerausgangsgröße
G_{Regler}	Übertragungsfaktor eines p-Reglers
$G_{Strecke}$	Übertragungsfaktor der zu regelnden Strecke

$K_{\text{j val}}$	Kostenwert für Kostenkriterium *j*
$K_{\text{j lim}}$	Kostengrenzen für Kostenkriterium *j*
$K_{\text{j val,limit}}$	2-Tupel aus Kostenwert und -grenze für Kostenkriterium *j*
$\forall j(criterion(j))$	alle Kostenkriterien *j* aus der Menge der zu optimierenden Kriterien
S	skalares Gütemaß
W	Wichtungsvektor

Verzeichnis der verwendeten Abkürzungen und Begriffe

Diese Arbeit enthält eine Vielzahl von Begriffen aus der englischen Sprache sowie Begriffe aus Programmier- bzw. Hardwarebeschreibungssprachen. Englische Begriffe sind *kursiv* gedruckt und im folgenden, neben weiteren Begriffen, erklärt. Bestandteile von Programmiersprachen sind in der Schriftart `Courier` gedruckt, sofern sie zur näheren Erklärung eines Sachverhaltes dienen. Schlüsselworte von Programmiersprachen, die im direkten Textfluß vorkommen, werden wie Substantive gebraucht und sind nicht extra hervorgehoben. Quelltextauszüge sind ebenfalls in der Schriftart `Courier` gedruckt, wobei für Kommentare die normale Schriftart verwendet wird.

AC-Simulation	Bei der AC-Simulation wird der Frequenzgangs eines Systems simuliert. Die Ergebnisse sind eine Funktion der Frequenz.
ADA	Eine strukturierte Programmiersprache, die in den 1970ern entworfen wurde und nach der ersten Programmiererin Lady Ada Lovelace benannt ist.
ADC	**A**nalog to **D**igital **C**onverter, A/D-Wandler
analog solver	Analoglöser, Teil des Analogsimulators, der das aus den Modellgleichungen entstehende Gleichungssystem löst, analoge Werte propagiert und neue Rechenzeitpunkte bestimmt.
API	Das **A**pplication **P**rogramming **I**nterface stellt eine Schnittstelle zur Applikationsprogrammierung dar, die ein Betriebssystem/Softwaresystem anderen Programmen zur Verfügung stellt.
Backannotation	Die Backannotation ist das Erweitern einer Gatternetzliste um Informationen zu parasitären Effekte (z. B. Verzögerungszeiten), die aus dem Layout gewonnen werden.
Bottom-Up	Eine Entwurfsstrategie, bei der ausgehend von verfügbaren Grundelementen mit zunehmender Abstraktion ein System entworfen wird.
C++	C++ ist eine objektorientierte Programmiersprache für allgemeine Anwendungen. Sie basiert auf der Programmiersprache C und wurde 1979 bei AT&T entwickelt.
CAN-Bus	Das **C**ontroller **A**rea **N**etwork ist ein asynchrones, serielles Bussystem, das von Bosch für die Vernetzung von Steuergeräten im Automobil entworfen wurde.
concurrent statement	Die nebenläufigen Anweisungen sind ein Gruppe von VHDL-Anweisungen, die quasiparallel abgearbeitet werden (siehe auch Abschnitt 2, S. 37 ff.).

COSSAP	Der nachrichtentechnische Systemsimulator COSSAP arbeitet nach dem Datenstrom-Prinzip. Das Systemmodell besteht aus miteinander verbundenen Blöcken mit definierten Ein- und Ausgängen, denen Verhaltensmodelle auf der Basis von C-Prozeduren hinterlegt sind. COSSAP wurde 1988 von Cadis vorgestellt. 1994 erfolgte die Übernahme von Cadis durch Synopsis.
Datenobjekt	In dieser Arbeit als Oberbegriff für die Elemente Signal, Variable, Konstante, Quantity und Terminal der Sprache VHDL bzw. VHDL-AMS verwendet, um Verwechslungen mit dem Begriff Objekt der objektorientierten Sprachen zu vermeiden.
DC-Simulation	Bei der DC-Simulation wird der Arbeitspunkt eines Systems als Funktion von Parametern bestimmt.
Design-Flow	Entwurfsablauf
definieren	Im Zusammenhang mit VHDL verwendeter Begriff beim Beschreiben von Funktionen, Prozeduren und Konstanten. Beim Definieren einer Funktion oder Prozedur wird sowohl deren Kopf als auch deren Inhalt beschrieben. Konstanten werden beim Definieren angelegt und erhalten einen Wert zugewiesen. Im Zusammenhang mit C++ spricht man vom Definieren, wenn nur der Funktionskopf beschrieben wird, aber nicht deren Inhalt.
deklarieren	Im Zusammenhang mit VHDL verwendeter Begriff beim Beschreiben von Funktionen, Prozeduren und Konstanten. Beim Deklarieren wird nur der Kopf einer Funktion oder Prozedur beschrieben und nicht deren Inhalt. Konstanten werden beim Deklarieren nur angelegt, erhalten aber keinen Wert zugewiesen.
event	Ein VHDL-Simulator arbeitet ereignisbasiert, d. h., der Zeitpunkt, zu dem eine Lösung berechnet wird, bestimmt sich aus dem Auftreten digitaler Zustandsänderungen, den *events*.
FEM	Die **F**inite-**E**lemente-**M**ethode ist ein numerisches Verfahren zur näherungsweisen Lösung von partiellen Differentialgleichungen mit Randbedingungen.
FhG IIS/EAS	Fraunhofer-Gesellschaft, Institut für Integrierte Schaltungen/Außenstelle Entwurfsautomatisierung
Handshake	Ein Handshake-Verfahren ist eine Methode, wie sich zwei an einer Datenübertragung beteiligte Teilnehmer nach jedem Übertragungsvorgang durch unmittelbare Quittungssignale synchronisieren.
HDL	**H**ardware **D**escription **L**anguage; engl. für Hardwarebeschreibungssprache

heterogenes System	Ein System, das durch digitales, analoges elektrisches und/oder nichtelektrisches Verhalten charakterisiert wird.
I^2C Bus	Der **I**nter-**I**ntegrated **C**ircuit Bus (IIC) ist ein von Philips entwickelter serieller Bus für Geräte mit geringer Übertragungsgeschwindigkeit.
IC	**I**ntegrated **C**ircuit, engl. für integrierter Schaltkreis
IEEE	Das **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers ist ein weltweiter Berufsverband von Ingenieuren aus den Bereichen Elektro- und Informationstechnik.
IMC	Ein **I**nterface **M**ethod **C**all ist eine spezielle Zugriffsmethode auf Datenübertragungskanäle von SystemC (s. Abschnitt 5.3).
implementieren	Im Zusammenhang mit C++ spricht man vom Implementieren, wenn für eine Funktion sowohl deren Kopf als auch deren Inhalt beschrieben wird.
IP	**I**ntellectual **P**roperty, geistiges Eigentum
konservative Systeme	Konservative Systeme sind technische oder elektronische Systeme, in denen die Kirchhoffschen Gesetze in ihrer verallgemeinerten Form gelten: $\sum_{Masche} \mu_n = 0, \sum_{Knoten} \lambda_m = 0$
LVDS	Das **L**ow **V**oltage **D**ifferential **S**ignaling ist eine Hochgeschwindigkeitsdatenübertragung mit bis zu 2 GBit/s, wobei differentielle Signalleitungen benutzt werden.
MAM	Die **M**ulti-**A**rchitecture-**M**odellierung ist das im Rahmen dieser Arbeit entwickelte Verfahren zur mixed-level Simulation heterogener Systeme. Bei diesem Ansatz werden verschieden abstrakte Modelle für die einzelnen Komponenten hinterlegt. Vereinheitlichte Schnittstellen erlauben den Austausch der Modelle über Abstraktionsebenen hinweg.
MCM	**M**ulti **C**hip **M**odul
Meet-In-The-Middle	Eine Entwurfsstrategie, bei der sowohl Aspekte des Top-Down- als auch des Bottom-Up-Ansatzes berücksichtigt werden [21].
MEMS	**M**icro **E**lectro **M**echanical **S**ystem, engl. für Mikrosystem
mixed-level	Bei einer Mixed-Level-Simulation werden unterschiedliche abstrakte Modelle zur Berechnung verwendet.
mixed-mode	siehe mixed-signal

mixed-signal	Ein mixed-signal System enthält sowohl digitales als auch analoges Verhalten.
MOC	Das **M**odel **o**f **C**omputation (Berechnungsmodell) beschreibt die Art und Weise wie ein Simulationsmodell berechnet wird. Das MOC legt z. B. fest, zu welchen Zeitpunkten die Berechnungen durchzuführen sind (feste Zeitschritte, dynamisch bestimmte Zeitschritte, beim Eintreten von Ereignissen), ob die Berechnungen in Form von Zuweisungen oder durch Lösen eines Gleichungssystems erfolgen und in welcher Reihenfolge die einzelnen Modelle des Systems berechnet werden können. Die Wahl des MOC für einen bestimmten zu simulierenden Sachverhalt hat einen großen Einfluß auf Geschwindigkeit und Genauigkeit der Simulation.
nichtkonservative Systeme	Nichtkonservative Systeme sind technische oder elektronische Systeme, bei denen die Kirchhoffschen Gesetze nicht gelten.
open source	Der Source-Code (Quellcode) einer Hard- oder Softwarekomponente ist frei zugänglich und kann vom Nutzer an einen entsprechenden Anwendungsfall angepaßt werden.
OPV	**Op**erations**v**erstärker; elektronisches Bauelement
OSCI	Die **O**pen **S**ystem**C** **I**nitiative ist eine Initiative, die die Entwicklung der Sprache SystemC koordiniert.
PWM	**P**uls**w**eiten**m**odulation
RASSP-Modell	Beim **R**apid Prototyping of **A**pplication **S**pecific **S**ignal **P**rocessors Modell handelt es sich um ein Entwurfsmodell für technische Systeme.
reduced order modell	Ein aus einem FEM-Modell abgeleitetes Modell mit reduzierter Anzahl von Systemgleichungen (reduzierte Ordnung), das aber eine ähnlicher Genauigkeit wie das FEM-Modell besitzt.
refactoring methodology	Refactoring ist ein Begriff des Software-Engineerings und ist eine Methode zur systematischen Restrukturierung von Programm-Code, bei der die interne Struktur des Systems verändert wird, das Verhalten nach außen hin aber unverändert bleibt. Die grundlegende Idee bei dieser Restrukturierung besteht darin, immer nur kleine Stücke des Codes zu transformieren, so daß eventuell gemachte Fehler leicht zu entdecken sind [22].
resolution function	Eine spezielle Funktion in VHDL, die Konflikte auflöst, wenn mehrere Treiber zur selben Zeit auf ein Signal schreiben.

resolved signal	Ein Signal in VHDL, dem eine *resolution function* zugeordnet ist.
Reuse	Einmal entworfene Hard- oder Softwarekomponenten werden in weiteren Anwendungen wiederverwendet.
ROM	siehe **R**educed **O**rder **M**odel
RS232-Schnittstelle	Die RS232-Schnittstelle (auch EIA-232-Schnittstelle) ist ein Standard für serielle Schnittstellen.
RT-Ebene	**R**egister-**T**ransfer-Ebene, s. RTL
RTL	Der **R**egister **T**ransfer **L**evel ist eine Abstraktionsebene beim Entwurf digitaler Schaltungen, bei dem die Schaltung taktgenau auf der Basis von Flip-Flops (Registern) und Transferfunktionen beschrieben wird.
RT-Modell	Ein RT-Model ist ein Modell auf Register Transfer Level.
sequential statement	Die sequentiellen Anweisungen sind ein Gruppe von VHDL-Anweisungen, die nacheinander (sequentiell) abgearbeitet werden (siehe auch Abschnitt 2, S. 37 ff.).
SFB	Ein **S**onder**f**orschungs**b**ereich ist ein von der Deutschen Forschungsgemeinschaft (DFG) gefördertes Forschungsprojekt.
simultaneous statement	Die simultanen Anweisungen sind eine Gruppe von VHDL-Anweisungen, die ein Gleichungssystem bilden (siehe auch Abschnitt 2, S. 37 ff.).
Solver	Der Solver ist der Teil des Simulationswerkzeuges, der die Berechnung der Modellgleichungen durchführt.
Strukturmodell	Im Rahmen dieser Arbeit mit der Bedeutung „hierarchisches Modell“ verwendet. Der Begriff wird in anderen Publikationen auch für Modelle benutzt, die die geometrische Struktur einer Komponente beschreiben.
SystemC	SystemC ist eine C++-Klassenbibliothek, die es ermöglicht, Hardware mit der Programmiersprache C++ zu entwerfen.
SystemC-AMS	SystemC-AMS ist eine Erweiterung von SystemC für analoge und mixed-signal Systeme.
template	Schablone
Testbench	Ein Modell oder Programm, das Stimuli für zu testende Modelle (*device under test*) erzeugt und deren Ausgangsdaten analysiert.
time to market	Die Zeit, die für die Entwicklung eines Systems von der Produktidee bis zur Marktreife benötigt wird.

Top-Down	Eine Entwurfsstrategie, bei der ausgehend von einer Systemspezifikation die notwendigen Komponenten unter Verringerung des Abstraktionsgrades entwickelt werden.
Transceiver	transmitter/receiver, kombinierter Sender und Empfänger
Transient-Simulation	Bei der transienten Simulation erfolgt eine zeitliche Simulation eines Systems, die Ergebnisse sind eine Funktion der Zeit.
typical case	typischer Fall
UART	Der **U**niversal **A**synchronous **R**eceiver **T**ransmitter ist eine elektronische Komponente, die als universaler asynchroner Sender und Empfänger arbeitet.
Überladen eines Operators	In VHDL besteht die Möglichkeit, eine Funktion wie einen vorhandenen Operator zu benennen. Man spricht dann vom Überladen eines Operators. Auf diese Art und Weise kann man den Funktionsumfang von Operatoren abhängig vom Datentyp der Operanden erweitern.
unresolved signal	Ein Signal in VHDL, dem keine *resolution function* zugeordnet ist.
Verhaltensmodell	Ein Verhaltensmodell ist die Beschreibung des physikalischen Verhaltens einer Komponente (physikalisches Verhaltensmodell) oder die Beschreibung des Klemmenverhaltens einer Komponente (empirisches Verhaltensmodell).
Verilog	Verilog ist eine Sprache zum Entwurf digitaler Schaltungen/Schaltkreise.
VHDL	Die **V**ery **H**igh Speed Integrated Circuit **H**ardware **D**escription **L**anguage ist eine Sprache zum Entwurf digitaler Schaltungen/Schaltkreise.
VHDL-AMS	**VHDL**-**A**nalog and **M**ixed **S**ignal ist eine Erweiterung von VHDL zur Modellierung und Simulation analoger und mixed-signal Systeme sowie heterogener Systeme.
wired OR, wired AND	Herstellung einer logischen OR- bzw. AND-Verknüpfung durch die Zusammenschaltung mehrerer Open-Collector-, Open-Drain- oder Tristate-Ausgänge.
workaround	provisorische Lösung
worst case	ungünstigster anzunehmender Fall

Vorwort

Die vorliegende Arbeit entstand zwischen 2000 und 2004. In dieser Zeit war ich als wissenschaftlicher Mitarbeiter an der Professur Schaltungs- und Systementwurf der Technischen Universität Chemnitz tätig. Die Motivation für das behandelte Thema und einige Abschnitte resultieren aus der Forschungstätigkeit im Rahmen des Sonderforschungsbereiches (SFB) 379, der von der Deutschen Forschungsgemeinschaft (DFG) gefördert wird.

Ich möchte an dieser Stelle Personen danken, die in besonderer Weise zum Entstehen dieser Arbeit beigetragen haben:

Mein Dank gilt zu allererst Herrn Prof. Dr.-Ing. habil. Dietmar Müller für das entgegengebrachte Interesse, die gebotenen Arbeitsmöglichkeiten und die stete Unterstützung, die sich nicht nur auf fachliche Aspekte beschränkte. Ebenso danken möchte ich Herrn Prof. Dr.-Ing. Klaus D. Müller-Glaser und Herrn Dr.-Ing. habil. Peter Schwarz für die Übernahme der Gutachten.

Herrn Prof. Dr.-Ing. habil. Göran Herrmann möchte ich für die kritische Durchsicht von Teilen des Manuskriptes danken, ebenso Herrn Karsten Einwich für die Bereitstellung der SystemC-AMS-Klassenbibliothek.

Mein Dank gilt ebenso meinen Kollegen an der Professur Schaltungs- und Systementwurf und meinen Kollegen im SFB 379 für die gute Zusammenarbeit und Unterstützung im Rahmen meiner Forschungstätigkeit.

1 Einführung

Die hier vorliegende Arbeit beschäftigt sich mit Möglichkeiten zur Optimierung des Entwurfsprozesses heterogener Systeme. Dieses Kapitel soll daher einen kurzen Überblick über bestehende Entwurfsmethoden und -strategien heterogener Systeme gewähren, aus denen sich die dieser Arbeit zu Grunde liegenden Problemstellungen ableiten.

1.1 Entwurfsmethodik: Überblick über den Stand der Technik

Wie der Mikroelektronik-Entwurf ist der Mikrosystemtechnik-Entwurf stark modell- und simulationsbasiert. Das Systemverhalten von Mikrosystemen erweist sich aber als ungleich komplexer, da dieses von der teilweise engen Kopplung unterschiedlicher physikalischer Domänen beeinflußt wird. Simulation und Modellbildung auf unterschiedlichen Abstraktionsebenen und in verschiedenen physikalischer Domänen sind notwendige Etappen im Entwicklungsprozeß [23], [24], [25]. Dabei liegt der Automatisierungsgrad im Entwurf von Mikrosystemen noch immer unter dem Automatisierungsgrad im Mikroelektronik-Entwurf. Es existieren eine Vielzahl von Lösungen, die einzelne Entwurfsaufgaben unterstützen, den Anwendern wird allerdings kein geschlossener Entwurfsablauf geboten, wie es z. B. beim ASIC-Entwurf der Fall ist.

In [26] werden im Zusammenhang mit der Weiterentwicklung der Entwurfswerkzeuge für den Mixed-Signal- und Mikrosystem-Entwurf u. a. die folgenden Problembereiche genannt:

- Modellierung und Simulation über unterschiedliche physikalische Domänen und Abstraktionsebenen,
- Simulation von Systemen mit zeitkontinuierlichen (analogen) und zeitdiskreten (digitalen) Signalen,
- Verfügbarkeit von Bibliotheken mit Modellen von Komponenten und Teilsystemen,
- Methoden zur Parameterextraktion von Modellen.

Dabei werden bezüglich des Entwurfs und der Simulation heterogener Systeme und Komponenten die folgenden Aussagen getroffen:

- akzeptable Rechenzeiten für komplexe Systeme sind nur durch den Einsatz von Verhaltensmodellen möglich,
- die Leistungsfähigkeit von Mixed-Mode-Simulatoren muß steigen,
- Modellierungs- und Simulationszeit müssen reduziert werden.

Obwohl diese Ausführungen bereits 1996 entstanden, haben sie ihre Gültigkeit bis heute behalten, da sich neben der Weiterentwicklung der Entwurfswerkzeugen und Methoden für den Entwurf von Mikrosystemen auch die Anforderungen an die Mikrosysteme vergrößert haben.

Am weitesten fortgeschritten ist die Entwicklung im Bereich Modellierung und Simulation. Dafür stehen dem Ingenieur leistungsfähige Hardwarebeschreibungssprachen wie VHDL-AMS [27] und Verilog-AMS oder auch die Systembeschreibungssprachen Modelica [28], [29] und zukünftig auch SystemC-AMS [30], [31] zur Verfügung. Mit diesen Beschreibungssprachen hat man die Möglichkeit, heterogene Systeme, also Systeme die analoge, digitale und nichtelektrische Kompo-

nenten enthalten, im Ganzen auf unterschiedlichen Abstraktionsebenen [32] zu beschreiben und zu simulieren [33], [34], [35], [36], [37].

Die technischen Voraussetzungen für den Einsatz der aus dem Mikroelektronik-Entwurf bekannten Entwurfsmethoden wie High-Level-Modellierung, *Reuse* und IP-Design (*Intellectual Property*) im Mikrosystem-Entwurf [38], [39], [40], [41], [42], [43], [44] haben sich damit verbessert. Einen weiteren Beitrag dazu soll die im Rahmen dieser Arbeit geschaffenen Methode der Multi-Architecture-Modellierung leisten, die den Austausch von Modellen auf unterschiedlichen Abstraktionsebenen erleichtert.

Für VHDL-AMS werden mittlerweile leistungsfähige Simulatoren wie „AdvanceMS“, „Smash“, „Simplorer“ u. a. teils als kommerzieller Simulator, teils als kostenlose Demo-/Testversion angeboten. Daneben existieren auch sehr leistungsfähiger Tools mit eigenen Beschreibungsformen, z. B. „Matlab/Simulink“, die ebenfalls für den Entwurf heterogener Systeme/Mikrosysteme auf höherer Abstraktionsebene geeignet sind.

Da sich komplexe physikalische Sachverhalte (z. B. elektromagnetische Felder, mechanische Deformationen) nicht immer mit analytischen Ausdrücken beschreiben lassen, waren auch in der Systemsimulation mitunter FEM-Simulationen notwendig. Wegen der daraus entstehenden langen Rechenzeiten gibt es weltweit Bestrebungen, FEM-Modelle durch Parameter-/Ordnungsreduktion auf einen überschaubaren Satz von Differentialgleichungen zu reduzieren. Auf diesem Gebiet konnten in letzter Zeit erhebliche Fortschritte erzielt werden, wodurch sich der Einsatz von FEM-Simulation auf Systemebene oft vermeiden läßt [45], [46], [47], [48].

Auch auf dem Gebiet des Mikroelektronik-Entwurfes geht die Entwicklung neuer Methoden und Werkzeuge stetig weiter. So werden heute häufig ganze mikroelektronische Systeme auf einem einzelnen Chip oder aus mehreren Chips in einem Gehäuse integriert. Man spricht dabei vom „System on Chip“ (SoC) bzw. vom „System in a Package“ (SiP). Sowohl SoC als auch SiP vereinen oft analoge, digitale und softwaretechnische Komponenten [49], [50], [51], wofür eine Beschreibung mit VHDL nicht mehr effizient genug ist. Einen Ansatz zur Simulation solcher Systeme stellt die Systembeschreibungssprache SystemC dar. Diese Sprache erlaubt einerseits die Beschreibung digitaler Komponenten sowohl auf gleichem Abstraktionsniveau wie VHDL als auch auf höherem Abstraktionsniveau. Da SystemC eine C++-Klassenbibliothek ist, kann mit einem solchen System auch die Software des SoC oder SiP beschrieben und getestet werden. Für die Beschreibung analoger oder nichtelektrischer Komponenten ist SystemC zur Zeit noch nicht geeignet. Auf der Basis von Arbeiten zur Erweiterung von SystemC für mixed-signal Systeme [30] wurde im Rahmen der Open SystemC Initiative (OSCI) eine Arbeitsgruppe gegründet, die einen Quasi-Standard für eine Erweiterung von SystemC auf SystemC-AMS schaffen soll, mit dem auch analoges und nichtelektrisches Verhalten beschrieben werden kann [31].

Weit weniger angeglichen an den Entwicklungsstand beim Entwurf digitaler Systeme stellt sich der Stand der Forschung auf den Gebieten der Analog-Synthese und Synthese von Mikrosystemen (MEMS) dar. Da der Entwurfsraum (*design space*) bei analogen und nichtelektrischen Systemen wesentlich größer als bei digitalen Komponenten ist, existieren zwar Syntheselösungen für begrenzte Einsatzfälle, aber noch kein allgemeingültiger Ansatz, der einen automatisierten, durch-

gehenden Top-Down-Entwurf einer analogen oder gar nichtelektrischen Komponente von einer höheren Abstraktionsebene aus ermöglicht [52], [53], [54], [55], [56]. So gibt es z. B. für die Synthese nichtelektrischer Komponenten Ansätze auf der Basis von Reuse-Methoden in Form von Bibliotheken mit mechanischen/elektromechanischen Grundelementen. Eine darauf aufbauende Methode ist die Anwendung von genetischen Algorithmen zur Optimierung der Kombination dieser Grundelemente [57], [58]. Die Entwicklung und Anwendung von SystemC-AMS wird auch auf diesem Gebiet neue Perspektiven ermöglichen [24].

1.2 Entwurfsstrategien

Während sich im klassischen digitalen Entwurf ein strenger Top-Down-Entwurf bewährt und auch für den Entwurf von Mikrosysteme erste Ansätze zum Top-Down-Entwurf existieren [59], so gestaltet sich diese Herangehensweise dennoch als problematisch, da z. B. bei mikromechanischen Komponenten technologische und physikalische Grenzen viel eher erreicht werden, als in der klassischen Mikroelektronik. Deshalb hat sich im Bereich der Mikrosystemtechnik der Ansatz Meet-In-The-Middle [21] als geeignet erwiesen, der sowohl Top-Down- als auch Bottom-Up-Strategie in sich vereint. Dabei entwirft man zunächst z. B. parametrisierbare elektromechanische Grundstrukturen Bottom-Up auf der Basis der technologischen Realisierbarkeit. Ausgehend von der Soll-Funktionalität erfolgt dann Top-Down der Entwurf des eigentlichen Mikrosystems auf der Basis der realisierbaren Komponenten.

Genauer betrachtet handelt es sich beim Top-Down-Ansatz der digitalen Mikroelektronik ebenfalls um ein Meet-In-The-Middle-Verfahren. Der ASIC-Hersteller stellt eine Bibliothek mit herstellbaren Grundfunktionen Bottom-Up zur Verfügung, auf deren Basis am Ende des Top-Down-Entwurfs dann die Synthese der endgültigen digitalen Schaltung erfolgt. Da aber diese letzten Schritte des Entwurfs digitaler Schaltungen im wesentlichen vollautomatisiert sind, und – abgesehen von Timing- und Ressourcengrenzen – sich praktisch jede digitale Schaltung abbilden läßt, entfällt für den Entwerfer eines digitalen Systems der Bottom-Up-Schritt, so daß der Entwurf als reiner Top-Down-Entwurf erscheint.

Ähnliche Vorgehensweisen existieren mittlerweile auch für den Entwurf analoger Schaltungen [60], [61], wobei hier der Meet-In-The-Middle-Ansatz bereits stärker in den Vordergrund tritt. Da neben der eigentlichen Funktion der Schaltung Faktoren wie Signal-zu-Rauschverhältnis, Grenzfrequenzen, Phasenlaufzeiten, harmonische Verzerrungen, Intermodulation, Spannungs- und Stromoffsets, thermisches Verhalten u. v. a. eine entscheidende Rolle spielen, kann nicht mehr davon ausgegangen werden, daß sich praktisch jede Sollfunktionalität realisieren läßt. Es existieren Bibliotheken mit Grundelementen wie Dioden, Transistoren, OPV, A/D- und D/A-Wandler, Filter usw. Gehen die Anforderungen der Schaltung jedoch über die Eigenschaften der Grundelemente hinaus, so muß der Analog-Entwerfer sich die entsprechenden Elemente auf der Basis der physikalischen und technologischen Grenzen selbst erstellen.

Im Mikrosystementwurf kommt der Meet-In-The-Middle-Ansatz am stärksten zur Geltung. Ein strikter Top-Down-Entwurf, ausgehend von einer Systemspezifikation, kann z. B. bei mechanischen Komponenten schnell an physikalische oder technologische, insbesondere topologische

Grenzen stoßen. So können seismische Massen in Vibrationssensoren für tiefe Frequenzen nicht beliebig groß gewählt werden, da sich sonst die Federn der Aufhängung durchbiegen. Beim Entwurf von vertikalen Strukturen lassen sich bestimmte Verhältnisse von Tiefe zu Spaltbreite aus technologischen Gründen nicht oder nur mit erheblichem Aufwand überschreiten. Der Entwerfer ist somit gezwungen, eine Vielzahl von benötigten Grundelementen zu Beginn der Entwicklungsarbeiten zu erstellen, bevor der Entwurf des eigentlichen Systems beginnen kann. Zwar stehen ebenfalls auch Bibliotheken mit Grundelementen zur Verfügung, allerdings decken diese aufgrund des immens großen Entwurfsraumes von Mikrosystemen in der Regel nur einen Bruchteil der benötigten Strukturen ab.

1.3 Abstraktionsebenen und Entwurfsmodelle im Systementwurf

Die Entwicklung von elektronischen Systeme und Mikrosystemen ist bei der ständig steigenden Komplexität, Integrationsdichte und Leistungsfähigkeit nur durch strukturierte Vorgehensweisen beherrschbar. Wachsender Konkurrenzdruck und steigende Kundenanforderungen bedingen immer kürzere Entwicklungszeiten (*time to market*). Deshalb wird beim Entwurf idealerweise der in Abschnitt 1.2 beschriebene Top-Down Entwurf angestrebt [62].

1.3.1 Y-Diagramm

Beim Entwurf elektronischer Systeme unterscheidet man üblicherweise die drei Sichtweisen Verhalten, Struktur und Geometrie, was durch das sogenannte Y-Diagramm nach Gajski-Walker [63] verdeutlicht wird (Bild 1-1). Während die Äste des Y-Diagramms die drei Sichtweisen repräsentieren, stellen die Kreise mit unterschiedlichen Radien die verschiedenen Abstraktionsebenen im Entwurf dar, wobei ein großer Radius hohe Abstraktion bedeutet. Den Entwurf eines elektronischen Systems kann man nun vereinfacht betrachtet als eine Reihe von Transformationen und Verfeinerungen beginnend auf dem äußersten Kreis und endend auf dem innersten Kreis betrachten.

Am Anfang des Systementwurfes steht dabei die Systemspezifikation. Diese sollte vom Auftraggeber vollständig und fehlerfrei erstellt werden. Dabei sind Spezifikationen in Papierform nach wie vor nicht unüblich, es existieren allerdings auch Ansätze zu formalen, ausführbaren Spezifikationen als Top-Level Einstieg in den Systementwurf [64], [65]. Ausgehend von der Systemspezifikation erfolgt dann die Partitionierung des Systems (funktionale Dekomposition) und damit die Zuordnung der grundsätzlichen Funktionen zu den einzelnen Teilsystemen. Anschließend wird der Entwurf soweit schrittweise mit weiteren Details versehen, bis die zur Fertigung notwendigen Daten vorliegen.

Eine Erweiterung des Y-Diagramms stellt das X-Diagramm dar, welches in [66] vorgestellt wird. Es erweitert das Y-Diagramm um Aspekte des Tests. Darüber hinaus führt man in [66] ein Modell des Entwurfsprozesses über mehrere Abstraktionsebenen ein, das mit dem Y-Diagramm korreliert.

Die Anwendung des Y-Diagramms auf analoge Systeme zeigt Bild 1-2. Die Darstellung ist eine Kombination aus dem Y-Diagramm für analoge Systeme nach [56] und den Abstraktionsebenen für den Analog-Entwurf nach [67]. Eine weiteres Modell des Entwurfsraumes für analoge Systeme wird in Abschnitt 1.3.4 vorgestellt.

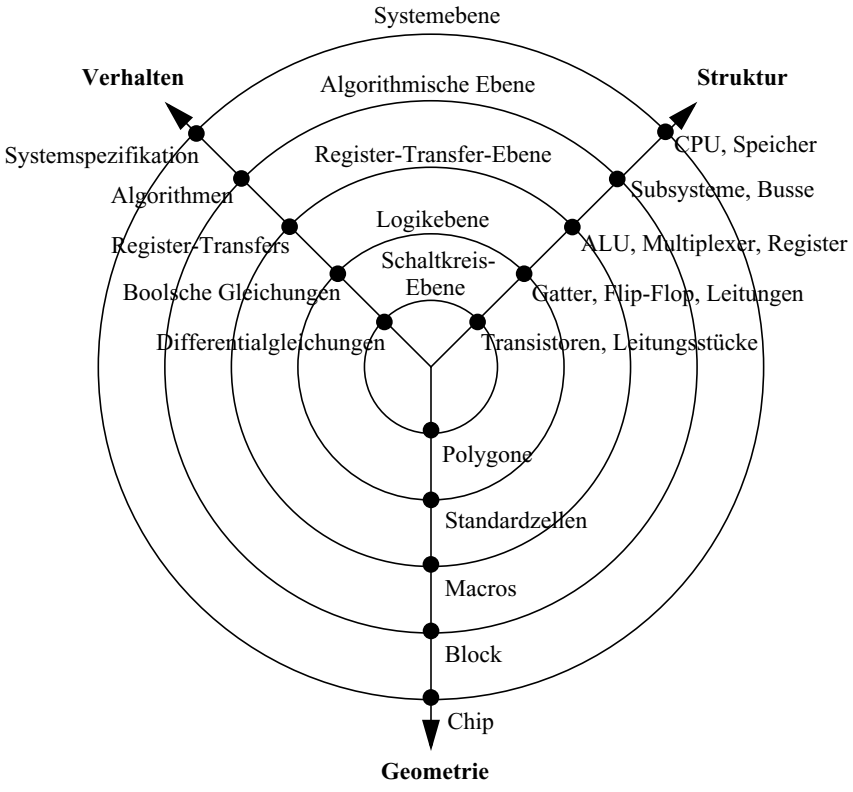


Bild 1-1 Y-Diagramm nach Gajski, Walker [63]

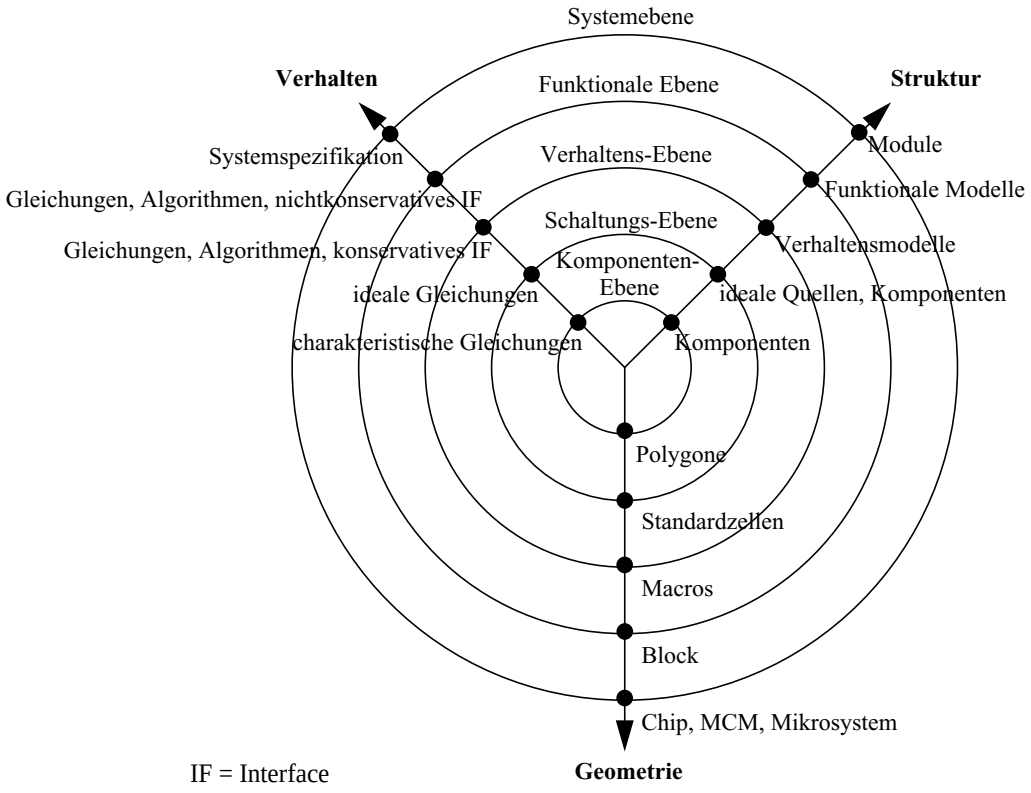


Bild 1-2 Y-Diagramm für analoge Systeme nach Moser [56] und Huss [67]

1.3.2 Entwurfswürfel

In [68] betrachtet man dagegen nur die Modellierung und definiert darauf aufbauend ein Entwurfswürfel (Bild 1-3). Er beruht darauf, daß sich die Modellierungsmöglichkeiten primär durch die Aspekte Beschreibungsform, Zeit und Wert darstellen lassen. Im Unterschied zum Y-Diagramm wird davon ausgegangen, daß man die Aspekte der Modellierung als unabhängig voneinander ansehen kann.

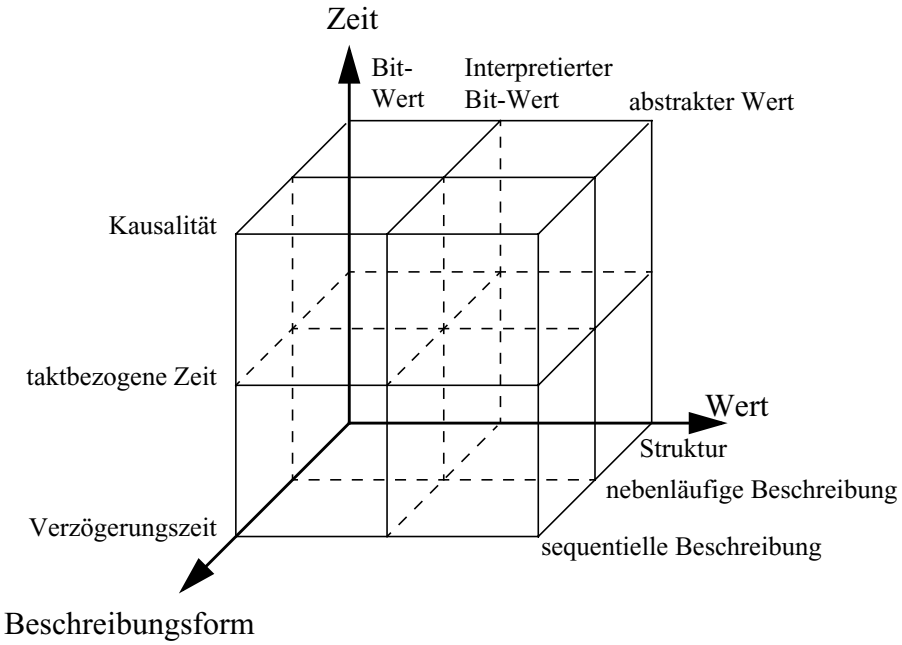


Bild 1-3 Der Entwurfswürfel nach Ecker [68]

In [69] und [70] wird dieses Modell des Entwurfsraumes verfeinert sowie Anwendungen auf dem Gebiet Entwurf und Verifikation vorgestellt. Eine Weiterentwicklung dieses Entwurfsraummodells, z. B. durch Einfügen von Eigenschaften wie Nichtdeterminismus und Superzeichenbildung, erfolgt in [71].

1.3.3 RASSP-Modell

Ebenfalls eine Weiterentwicklung des Entwurfswürfels stellt das RASSP-Modell (Rapid Prototyping of Application Specific Signal Processors, Bild 1-4) dar [72]. Das RASSP-Modell unterscheidet wie der Entwurfswürfel die Zeitauflösung und die Wertauflösung, verwendet zusätzlich jedoch neue Koordinaten wie „Instruktionszyklus“ oder „Token“ (nicht interpretiert). Weiterhin führt das RASSP-Modell die Unterscheidung nach Strukturauflösung (Hardwarestruktur) und funktionaler Auflösung (Umfang der Verhaltensbeschreibung) ein. Darüber hinaus wird bezüglich interner Eigenschaften (Verhaltensbeschreibung) und externer Eigenschaften (Schnittstellenbeschreibung) unterschieden. Damit erlaubt dieses Entwurfsmodell die Beschreibung weniger abstrakter Schnittstellen für abstrakte Verhaltensmodelle im Entwurfsprozeß. Weiterhin enthält dieses Modell des Entwurfsprozesses eine Achse für Softwareentwicklung im Rahmen des Hardware/Software-Codesigns.

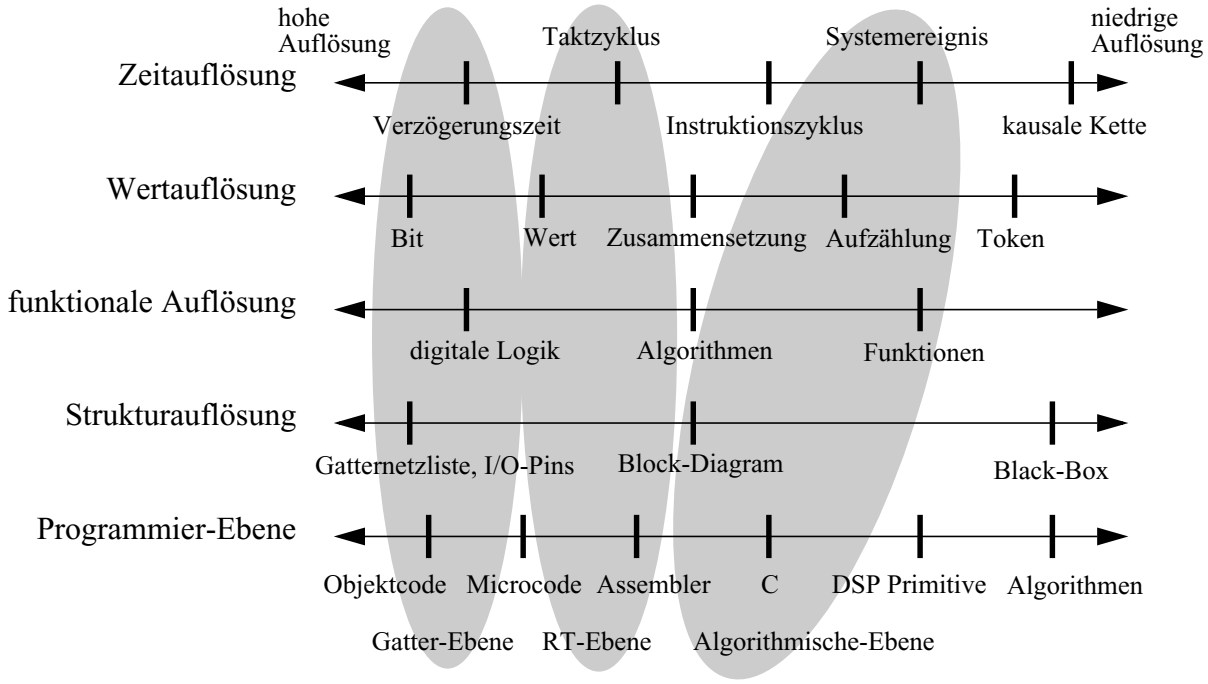


Bild 1-4 RASSP Modell [72]

1.3.4 Abstraktionsebenen für den Entwurf heterogener Systeme

Die bisher vorgestellten Entwurfsmodelle beziehen sich auf den Entwurf digitaler Hardware-Systeme bzw. auf den Entwurf digitaler Systeme auf der Basis des Hardware/Software-Codesigns. Für den Entwurf analoger, mixed-signal bzw. heterogener Systeme wird in [35] ein Entwurfsablauf von der Spezifikation bis zur Gatter-/Schaltungsebene vorgestellt (Bild 1-5).

Nach der Partitionierung des Systems führt man in der Regel eine Machbarkeitsstudie durch, um möglichst früh im Entwurfsprozeß entscheiden zu können, ob das System mit den zur Verfügung stehenden Ressourcen realisierbar ist und in welchem Rahmen sich die Kosten dafür bewegen.

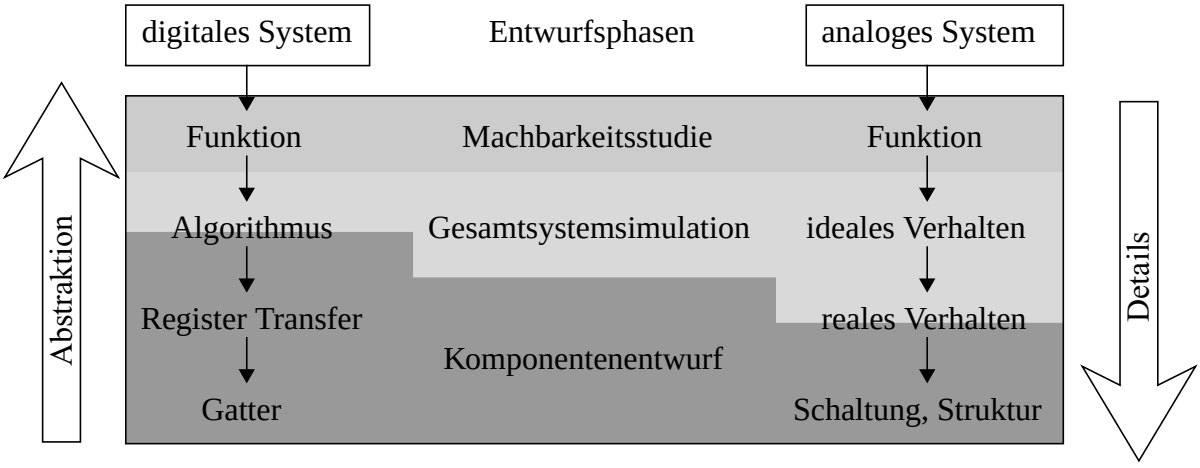


Bild 1-5 Abstraktionsebenen des Systementwurfes nach Huss [35]

Um während der Machbarkeitsstudie verschiedene Lösungsideen modellieren und validieren zu können, benötigt man Modelle, die einfach zu erstellen und zu ändern und deren Simulationszeiten kurz sind. Typisch für solche Modelle ist die abstrakte Darstellung der Funktionalität, wobei man den physikalischen Aufbau der Komponenten oder parasitäre Effekte in der Regel nicht berücksichtigt. Nach [35] werden die Schnittstellen nur definiert, aber nicht implementiert. Auf Basis der Systemspezifikation und der während der Machbarkeitsstudie gewonnenen Erkenntnisse entstehen konsistente Spezifikationen der Teilsysteme, die in den nächsten Entwurfsphasen implementiert werden.

Spätestens mit der Gesamtsystemsimulation wird eine ausführbare Spezifikation erzeugt, die als Testbench für Komponenten dienen kann. Die Modelle für die Gesamtsystemsimulation beschreiben das beobachtbare, ideale Verhalten der Komponente. Dazu kommen ähnlich der Machbarkeitsstudie empirische Verhaltensmodelle, d. h. Modelle ohne Bezug zum physikalischen Verhalten oder Modelle mit stark abstrahierten physikalischen Verhalten zum Einsatz. Nach [35] werden zur Gesamtsystemsimulation die Schnittstellen definiert, um eine parallele Entwicklung der Teilsysteme durchführen zu können.

Zur Implementierung der Komponenten kann man nun, wie in Abschnitt 1.2 beschrieben, bei digitalen Teilsystemen Synthesetools einsetzen, die die digitale Schaltung auf Gatternetzlisten abbilden. Für analoge Schaltungen oder mikromechanische Komponenten muß die Abbildung der Modelle auf Schaltungsnetzlisten in der Regel noch im Bottom-Up-Verfahren durchgeführt werden. Mit der Abbildung der Modelle auf Basiszellen steigt die Genauigkeit, und der Gültigkeitsbereich der Modelle wächst, dafür sinkt die Simulationsgeschwindigkeit.

Was in dem Beispiel in [35] nicht notwendig ist, ist die Berücksichtigung des Problems der sich mit der Weiterentwicklung (Verfeinerung) der Modelle möglicherweise ändernden Schnittstellen. So kann ein weiterentwickeltes Komponentenmodell, welches z. B. Erhaltungssätze berücksichtigt, nur mit zusätzlichem Aufwand in die Gesamtsystemsimulation integriert werden, deren restliche Modelle das Verhalten abstrakt beschreiben. Dies stellt eine zusätzliche Fehlerquelle dar und erschwert daher die Verwendung des Gesamtsystemmodells als Testbench. In diesem Punkt modifiziert der im Rahmen dieser Arbeit entstandene methodische Ansatz der Multi-Architecture-Modellierung den Entwurfsablauf heterogener Systeme. Das Ziel dabei ist es, die Modelle auf Gesamtsystemebene mit Schnittstellen auszustatten, die es erlauben, abstrakte Modelle gegen weiterentwickelte Komponentenmodelle auszutauschen, ohne Änderungen am Gesamtsystemmodell vornehmen zu müssen.

1.4 Hardwarebeschreibungssprachen

Um den Herausforderungen beim Entwurf elektronischer Systeme wie

- hohe Komplexität,
- kürzere Entwurfszeiten (*time to market*) und
- Minimierung der Kosten

gerecht zu werden, erfolgte bereits in den 80er Jahren die Entwicklung von standardisierten Hard-

warebeschreibungssprachen wie VHDL, da die bis dahin gebräuchlichen Beschreibungsformen schnell veralteten und den Anforderungen an einen effizienten Entwurf nicht mehr gerecht wurden. Zur Meisterung der Herausforderungen unterstützen die neuen Hardwarebeschreibungssprachen neben ihrer eigentlichen Bestimmung, der Modellierung und Simulation, die folgenden Ansätze:

- Ermöglichen eines strukturierten Entwurfes (z. B. Top-Down-Strategie),
- Beschreiben der Modelle auf unterschiedlichen Abstraktionsebenen mit verschiedenen Sichtweisen,
- Beschreiben von Verhalten und Struktur von Hardwaresystemen mit der Möglichkeit zur Partitionierung und Hierarchiebildung,
- Unterstützen der Entwurfsautomatisierung (z. B. Synthese),
- Bereitstellen einer entwurfsbegleitenden Dokumentation,
- Bereitstellen einer Basis für die Wiederverwendung von Komponenten (*Reuse*).

Der Aufbau einer Hardwarebeschreibungssprache (HDL) ist dabei ähnlich dem einer Programmiersprache, wobei die HDL zusätzlich parallele (nebenläufige) Anweisungen erlaubt, die die Funktion von Hardware abbilden können. Die HDLs dienen als Kommunikationsmedium zwischen Hardwareentwicklern und Entwurfswerkzeugen.

1.4.1 VHDL-AMS und Verilog-AMS

Modellierung und Simulation von Komponenten und Systemen mit zeitdiskreten und zeitkontinuierlichen Signalen spielen in der Elektronik und zunehmend in der Mikrosystemtechnik eine große Rolle [32]. War man bei der Simulation heterogener Systeme vor einigen Jahren noch auf gekoppelte Simulatoren oder Analogietransformationen angewiesen, so stehen dem Entwickler heute relativ leistungsfähige, aber teilweise noch nicht ausgereifte Simulationstools zusätzlich zur Verfügung, die eine Beschreibung und Simulation solcher Systeme mit einer einheitlichen Beschreibungsform ermöglichen. Beispiele für solche einheitlichen Beschreibungsformen sind die Hardwarebeschreibungssprachen VHDL-AMS [27], [73] und Verilog-AMS [74]. Da diese Sprachen nicht an einen Simulator gebunden und sowohl kommerzielle als auch kostenlose Test-/Demoversionen von Simulatoren verfügbar sind, werden in zunehmenden Maße Modellbibliotheken unter Beachtung der Anforderungen bezüglich *Intellectual Property* (IP) und *Reuse* erstellt.

1.4.2 SystemC-AMS

Moderne elektronische Systeme und Mikrosysteme enthalten oft einen beträchtlichen Anteil von Softwarekomponenten. Allerdings lassen sich Softwarekomponenten nicht direkt mit VHDL-AMS oder Verilog-AMS beschreiben. Eine Lösung dieses Problems kann die Systembeschreibungssprache SystemC darstellen. Da SystemC auf der Programmiersprache C++ basiert, können digitale Hardware und Software gemeinsam beschrieben werden.

SystemC hat seinen Ursprung in einer Referenz-Implementierung der Firma Synopsys Inc. und wird im Rahmen der OSCI (Open SystemC Initiative) kontinuierlich weiterentwickelt. SystemC entstand unter dem Gesichtspunkt der Schaffung eines Open-Source-Standards für ein auf C++

basierendes System-Level-Design. Ziel dieser einheitlichen Sprachplattform ist es, die Spezifikation und Implementierung von Hard- und Softwarekomponenten eingebetteter digitaler Systeme zu ermöglichen und somit ältere, proprietäre Entwurfsumgebungen, wie z. B. den nachrichtentechnische Systemsimulator COSSAP, abzulösen. Für diese Zwecke stellt SystemC die für die Hardwarespezifikation notwendigen Konstrukte für die Modellierung von Parallelität, Reaktivität und Zeitverhalten bereit. SystemC ist dabei eine C++-Klassenbibliothek, mit der die oben genannten hardwaretypischen Eigenschaften mit der Programmiersprache C++ beschrieben werden können. Das in Form einer Menge von Makros realisierte *Application Programming Interface* (API) dieser Klassenbibliothek stellt die Sprachelemente von SystemC bereit. Die Klassenbibliothek selbst implementiert den zur Simulation digitaler Systeme notwendigen deltazyklusbasierten, ereignisgesteuerten Simulationskern [75].

Frühe Versionen von SystemC (Versionsnummern 1.X) erlaubten die Systemmodellierung ausschließlich auf Register-Transfer-Ebene, wodurch sich die Expressivität der Sprache nicht wesentlich von der von VHDL oder Verilog unterschied. Ab Version 2.0 ermöglicht SystemC eine abstraktere Modellierung auf Systemebene, insbesondere die High-Level-Modellierung der Systemkommunikation auf der Basis von Kanälen mit prozeduralen Interfaces [75].

Die Formulierung der Sprache SystemC ausschließlich auf der Basis der Expressivität von C++ hat entscheidende Vorteile. So genügt für die Simulation von SystemC-Modellen ein Rechner mit C++-Compiler, der das Modell zusammen mit dem Simulationskernel in ein auf der jeweiligen Rechnerplattform ausführbares Programm übersetzt. Weiterhin kann SystemC sehr einfach durch Hinzufügen zusätzlicher Sprachkonstrukte zur SystemC-Klassenbibliothek erweitert werden, so daß die Notwendigkeit der Adaption entsprechender Compiler entfällt.

Da mit SystemC nur digitale Systeme beschrieben werden können, entstanden erste Ansätze, auch analoges und nichtelektrisches Verhalten mit SystemC zu beschreiben. Aus diesen Aktivitäten heraus entwickelte sich eine SystemC-Working-Group, die an der Schaffung einer einheitlichen Beschreibungsform für SystemC-AMS arbeitet. SystemC-AMS soll dabei für die unterschiedlichen Aspekte analoger Simulation verschiedene Models of Computation (MOC) bereitstellen. Dies ist für eine zeiteffiziente Simulation von großer Bedeutung, da unterschiedliche zu simulierende Probleme mit verschiedenen Simulationsstrategien berechnet werden können. So kann ein System mit gerichteten Datenflüssen mit einem darauf ausgelegten Simulator wesentlich schneller berechnet werden als z. B. mit einem Netzwerksimulator. Zur Simulation eines Netzwerkes muß man dagegen einen solchen Netzwerksimulator einsetzen, um hinreichend genaue Ergebnisse zu erzielen. So stellt SystemC-AMS z. B. Lösungsalgorithmen für Datenfluß-Systeme und für einfache Netzwerke bereit. Für die Simulation von nichtlinearen elektrischen oder mechanischen Netzwerken wird eine Kopplung mit numerischen Differentialgleichungslösern angestrebt, wobei durch die offene, flexible Architektur von SystemC-AMS jederzeit weitere Lösungsalgorithmen angebunden werden können. Dadurch kann im Bedarfsfall auch für analoge Teilsysteme die Genauigkeit von Spice oder VHDL-AMS/Verilog-AMS erreicht werden [24], [30], [31], [76].

1.4.3 Weitere Hardwarebeschreibungssprachen

Neben VHDL-AMS, Verilog-AMS und SystemC existieren noch andere Beschreibungssprachen für heterogene Systeme. Eine davon ist Modelica [28], [29]. Die Sprache Modelica dient der Modellierung beliebiger physikalisch/technischer Systeme. Die Sprache ist objektorientiert und unterstützt den Modellaustausch und die Wiederverwendung von Modellteilen.

Darüber hinaus gibt es eine Reihe kommerzieller, sehr leistungsfähiger Systemsimulationsumgebungen, die über simulatoreigene Beschreibungsformen verfügen. Beispiele hierfür sind „Matlab/ Simulink“ [77] oder „Simplorer“ [78]. Solche Systeme haben den Nachteil einer nur begrenzten Austauschbarkeit der Modelle mit anderen Simulationssystemen.

1.5 Motivation zu dieser Arbeit

Wie bereits mehrfach angedeutet, steigt die Komplexität moderner technischer Systeme ständig. Durch Simulation kann der Entwerfer die Funktionalität solch komplexer Systeme im voraus bestimmen und durch Auswahl verschiedener Parameter oder Designalternativen sein System optimieren. Insbesondere bei analogen, mixed-signal oder heterogenen Systemen besteht jedoch das Problem, daß trotz rasanter Fortschritte in der Rechentechnik die Simulation der ebenfalls immer komplexer werdenden Systeme sehr viel Zeit in Anspruch nehmen kann. Außerdem existieren nur wenige Ansätze zur Synthese analoger Strukturen, so daß im Vergleich zum digitalen Schaltungsentwurf wesentlich mehr „Handarbeit“ notwendig ist (vgl. Abschnitt 1.1).

Zur Verringerung der Simulationszeiten werden in der Regel die Modelle der Komponenten abstrahiert (vgl. Abschnitt 1.3). Allerdings geht mit der Abstraktion i. allg. ein Verlust an Genauigkeit einher. Auf der anderen Seite ist eine Gesamtsystemsimulation ausschließlich aus detaillierten Modellen oft nicht nötig. Daher wäre es von Vorteil, wenn der Ingenieur sein Systemmodell, wie in Bild 1-6 angedeutet, während des Entwurfs durch Auswahl abstrakter oder detaillierter Komponentenmodelle zwischen Geschwindigkeit und Genauigkeit in Abhängigkeit davon skalieren kann, welcher Aspekt des Systemverhaltens mit der Simulation verifiziert werden soll. Ein weiterer Vorteil dieses Ansatzes würde darin bestehen, daß die abstrakten Modelle der Systemsimulation gleichzeitig als Testbench für detaillierte Modelle dienen können.

Ziel dieser Arbeit ist es daher, einen solchen methodischen Ansatz zu entwickeln, in dem für eine Komponente unterschiedlich abstrakte Modelle (Architekturen) hinterlegt werden. Entsprechend den Y-Diagrammen nach Bild 1-1 und Bild 1-2 sollen dabei für digitale Modelle die Abstraktionsebenen von algorithmischer Ebene bis Logikebene und für analoge Modelle von funktionaler Ebene bis Komponentenebene abgedeckt werden. Dabei entsteht jedoch das Problem, daß in der Regel der Abstraktionsgrad eines Modells den Abstraktionsgrad der Modellschnittstellen bestimmt. So entstehen in einem Top-Down-basierten Systementwurf zu Beginn Modelle auf einem hohen Abstraktionsniveau. Das Abstraktionsniveau der Modelle verringert sich mit Fortschreiten des Entwurfsprozesses. Dies kann dazu führen, daß die Schnittstellen der Modelle mit jedem Entwicklungsschritt auf Grund der damit verbundenen Verringerung des Abstraktionsniveaus angepaßt werden müßten, was einen zeitaufwendigen und fehleranfälligen Prozeß darstellt.

Zur Umgehung dieses Problems werden für die unterschiedlich abstrakten Modelle zu Beginn des Entwurfs einheitliche, abstraktionsebenenübergreifende Schnittstellen geschaffen, indem in der Regel diejenige Schnittstelle, die auf niedrigem Abstraktionsniveau notwendig sein wird bereits auf hoher Abstraktionsebene zur Anwendung kommt. In diesem Zusammenhang auftretende Probleme und deren Lösung sind in Abschnitt 3, S. 43 ff. am Beispiel von VHDL-AMS beschrieben. Die Anwendung dieses methodischen Ansatzes wird in Abschnitt 4, S. 69 ff. erläutert und die Übertragung des Ansatzes auf andere Hardwarebeschreibungssprachen am Beispiel von SystemC-AMS in Abschnitt 5, S. 89 ff. demonstriert.

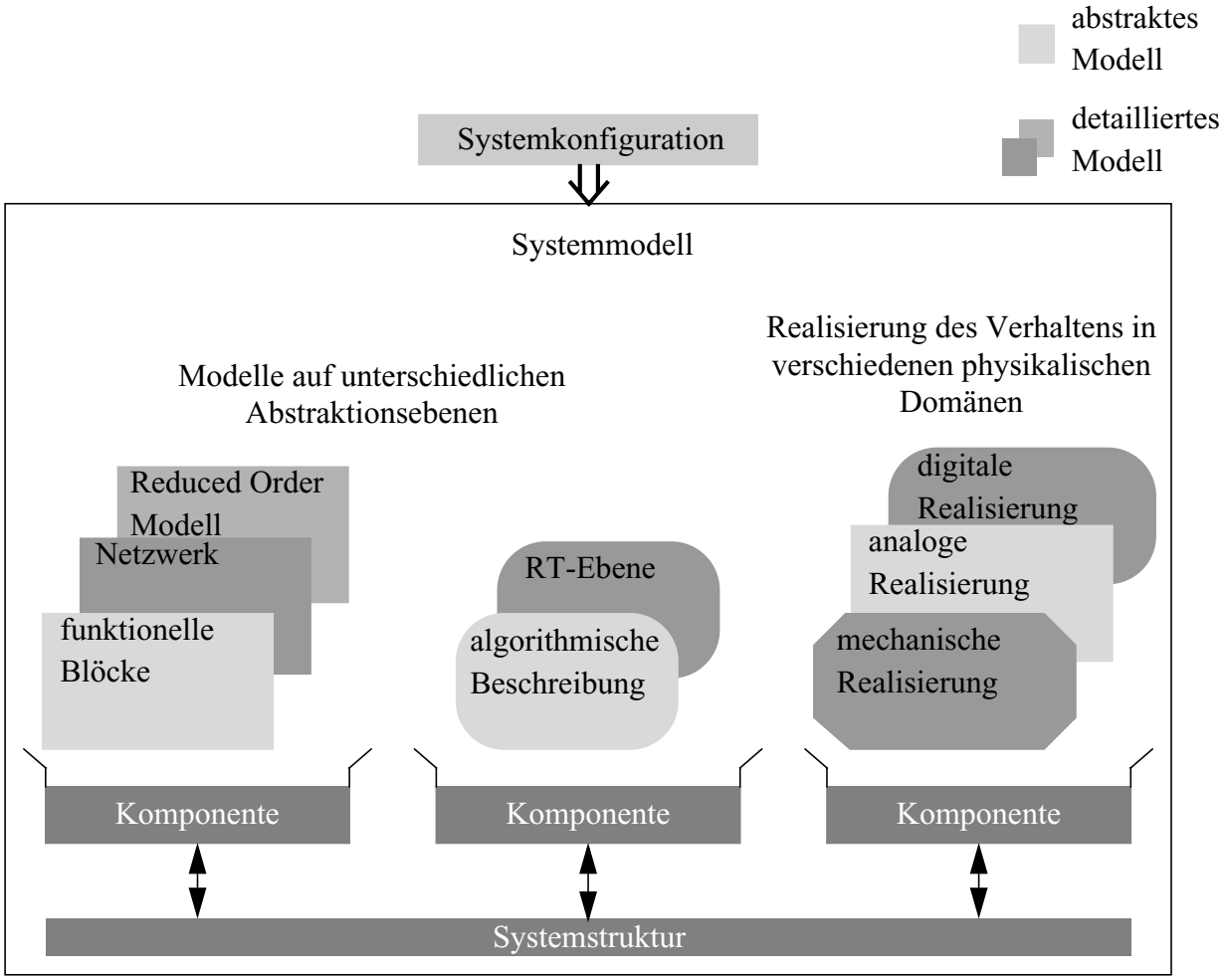


Bild 1-6 Verschiedene Konfigurationen eines heterogenen Systems

1.6 Abgrenzung zu alternativen Ansätzen

Die Arbeiten zur Multi-Architecture-Modellierung (MAM) begannen im Jahr 2000 auf der Basis von Publikationen wie [35]. In der Zwischenzeit wurden weitere Lösungsansätze für das Problem der Schnittstellengestaltung in Systemen mit verschieden abstrakten Komponenten erarbeitet. So verwies man z. B. 2004 im Rahmen eines Vortrages von Infineon [79] auf die Problematik unterschiedlich abstrakter Schnittstellen im Rahmen von Entwicklungen in der Automobilindustrie. In

diesem Zusammenhang wurde der für die Erstellung einheitlicher Schnittstellen notwendige Mehraufwand als „gut angelegte Zeit“ bezeichnet. Im folgenden wird eine Auswahl an Veröffentlichungen vorgestellt, die ähnliche Probleme wie in dieser Arbeit erörtern. Bild 1-7 zeigt eine Abgrenzung dieser einzelnen Ansätze zur Multi-Level-Simulation untereinander und zur MAM.

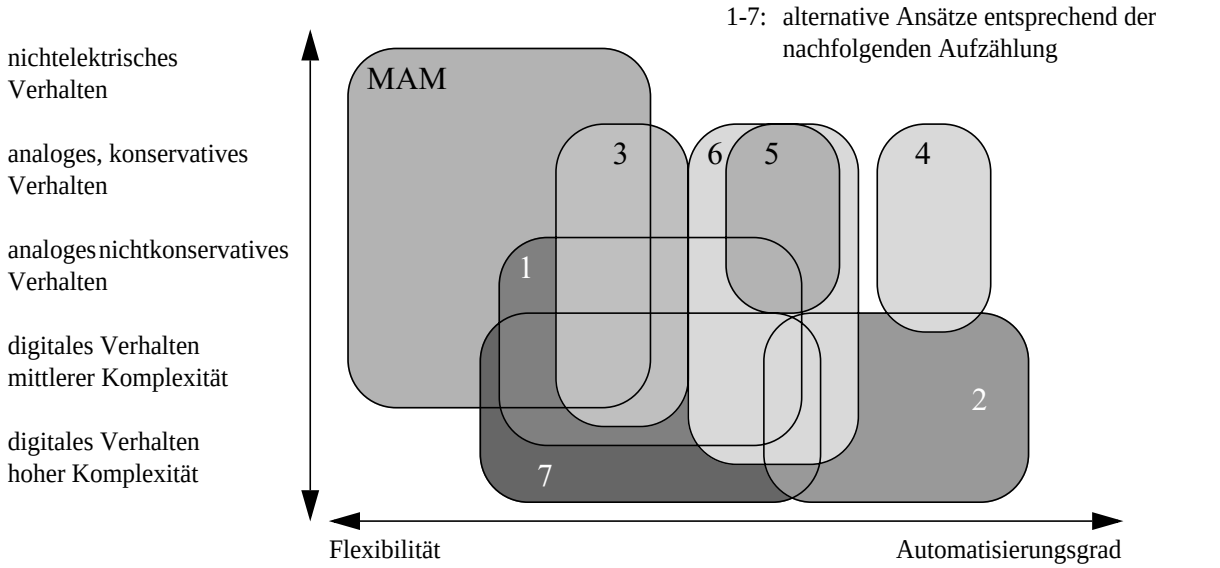


Bild 1-7 Abgrenzung anderer Multi-Level-Ansätze zur MAM

1. Während die hier vorgestellte MAM die verfeinerte Schnittstelle bereits auf hoher Abstraktionsebene verwendet, wurde 2002 in [30] ein gegenteiliges Konzept vorgestellt. Diese Konzept beruht auf der Anwendung der abstrakten Schnittstelle der obersten Abstraktionsebene auch auf die verfeinerten Modelle. Dabei kommen im Rahmen eines Systementwurfs neben analogen Erweiterungen von SystemC die objektorientierten Mechanismen von C++ zum Einsatz. Dieses Verfahren wird in Abschnitt 5.4, S. 111 ff. noch näher diskutiert.

2. Ein Verfahren zur automatischen Generierung digitaler Kommunikationskanäle auf der Basis von SystemC wurde im Rahmen einer Dissertation [80] an der Professur Schaltungs- und Systementwurf der TU Chemnitz entwickelt. Die entstandene Protokollbeschreibungssprache SVE [80] bietet unter anderem die Möglichkeit, Kommunikationsprotokolle abstrakt zu beschreiben und daraus Beschreibungen für Kommunikationskanäle zu erzeugen. Diese Kommunikationskanäle bieten Zugriffsmöglichkeiten für unterschiedlich abstrakte Modelle. In Abschnitt 5.3 wird der Ansatz noch näher erläutert und es werden Kombinationsmöglichkeiten mit der MAM aufgezeigt.

3. 2003 stellte die Firma Motorola im Rahmen eines Vortrags [81] ein der MAM vergleichbares Verfahren auf der Basis von Verilog-AMS vor. Dabei wird für digitale und analoge Schnittstellen ebenfalls diejenige Implementierungsvariante benutzt, die auf niedriger Abstraktionsebene notwendig ist, woraus sich auch eine moderate Verlängerung der Simulationszeit ergeben kann. Der wesentliche Unterschied im Vergleich zum hier vorgestellten Ansatz besteht darin, daß im Rahmen dieser Arbeit ein allgemein anwendbarer Ansatz auf

der Basis von VHDL-AMS Packages entstand, während der Ansatz von Motorola eng an hauseigene Design-Flows gekoppelt ist.

4. Die Firma Mentor Graphics bietet in ihrem Software-Paket „Architect-IC“ die Möglichkeit, verschiedene Sichten auf Modelle zu hinterlegen. Die Verbindung unterschiedlich abstrakter Modelle erfolgt mittels Konvertern, die von der Software bei Bedarf eingefügt oder entfernt werden. Der Entwerfer hat dabei allerdings keinen Einfluß auf die Art und den Funktionsumfang der Konverter.

5. In [82] wurde 2003 ein Ansatz zur generischen Modellierung analoger, mehrphasig arbeitender Systeme vorgestellt. Der Schwerpunkt dieser Arbeit liegt auf der Verhaltensmodellierung. Für die Schnittstellengestaltung analoger Systeme im Rahmen von Template-Modellen werden jedoch ähnliche Empfehlungen getroffen wie im Rahmen dieser Arbeit.

6. In [83] wird ein Design-Flow für heterogene Systeme vorgestellt, bei dem man die Verfeinerung von Modellen auf der Basis einer Code Transformation mittels einer *refactoring methodology* durchführt. Der Schwerpunkt liegt dabei auf der Transformation des Modellverhaltens von VHDL-AMS Modellen, wobei das zunächst abstrakt beschriebene Modellverhalten sukzessive um Aspekte des realen Verhaltens, z. B. nichtlineare Effekte, erweitert wird. Ziel des Verfahrens ist das Schaffen einer Verbindung von der Konzept-Phase zum Architektur- und Schaltungsentwurf bei gesicherter Konsistenz der Modell-Transformationen und somit die Schaffung validierter ausführbarer Spezifikationen und validierter Transitionen zwischen den Abstraktionsebenen. Die sich daraus ergebenden Probleme mit unterschiedlich abstrakten Schnittstellen werden teils konventionell durch Hinzufügen von Ports, teils durch der MAM ähnliche Prinzipien behandelt.

7. 2004 erfolgte die Vorstellung eines neuen orthogonalen Schemas für die Klassifikation der Modellierungsabstraktion digitaler Systeme [71]. Ziel dieses Ansatzes ist die Schaffung einer Multi-Level-Testbench für Hardware/Software-Systeme. Dabei wird ein vergleichbarer Weg eingeschlagen, wie er im Rahmen der MAM in Abschnitt 3.2.2, S. 48 beschrieben ist. Da sich der Ansatz auf digitale Systeme ausrichtet, ist die Umwandlung digitaler Protokolle stärker ausgebaut als bei der MAM. Dafür finden Belange der Analog- und Mixed-Signal-Modellierung keine Beachtung.

Eine weitere interessante Alternative stellt die Verwendung von Präprozessoren dar (z. B. der C-Präprozessor). Dabei beschreibt man sowohl abstrakte als auch verfeinerte Schnittstellen in einem Modell und berücksichtigt oder ignoriert diese mittels Präprozessoranweisung beim Kompilieren des Modells. Damit lassen sich sowohl Modellteile, wie z. B. Schnittstellen, als auch ganze Modelle zwischen abstrakt und detailliert konfigurieren. Einschränkend wirken die in Abschnitt 3.2, S. 46 ff. erläuterten Restriktionen bei der Abbildung detaillierter Schnittstellen auf abstrakte Schnittstellen. Weitere Nachteile dieses Verfahrens sind die mögliche Unübersichtlichkeit der resultierenden Modelle, mögliche zusätzliche Compile-Schritte sowie die Notwendigkeit des Vorhandenseins eines geeigneten Präprozessors für das Modell sowohl beim Entwerfer als auch beim potentiellen Kunden.

2 Einführung zur Hardwarebeschreibungssprache VHDL-AMS

Dieses Kapitel gibt einen kurzen Überblick über die Hardwarebeschreibungssprache VHDL-AMS, ohne dabei alle Details der Sprache und ihrer Simulation ausführlich zu behandeln. Da sich viele Aussagen in dieser Arbeit direkt auf Eigenschaften von VHDL-AMS beziehen, soll dieses Kapitel dazu dienen, das Verständnis für die vorliegende Arbeit zu erleichtern. Für einen weitergehenden Einblick zur Sprache VHDL-AMS und zum ASIC-Entwurf mit VHDL sei [62], [84], [85] und [86] empfohlen.

2.1 Historisches

Die Entwicklung von VHDL geht bis in die frühen 1980er Jahre auf das amerikanische Verteidigungsministerium (*Department of Defense*, DoD) zurück. Das DoD war auf der Suche nach einer einheitlichen Sprache zur Dokumentation technischer Systeme, da die bis dahin gebräuchliche Vielzahl von Beschreibungssprachen enorme Kosten verursachte. Ziel war die Schaffung einer klaren Beschreibungsform komplexer technischer System, die außerdem eine Austauschbarkeit von Modellen zwischen Entwicklergruppen ermöglichen sollte. Als Ergebnis dieser Entwicklung entstand die Hardwarebeschreibungssprache VHDL, die sich an die Programmiersprache ADA anlehnte und die 1987 durch das IEEE als Standard 1076-1987 verabschiedet wurde. 1993 erfolgte dann die Verabschiedung eines überarbeiteten Standards 1076-1993, der einige Erweiterungen und Vereinheitlichungen der Sprache bietet.

Die Arbeiten zur Erweiterung von VHDL für analoge und heterogene Systeme begannen 1993. Diese Arbeiten wurden 1999 nach langen Diskussionen abgeschlossen und als IEEE Standard 1076.1-1999 veröffentlicht. Der als VHDL-AMS (manchmal auch nur als VHDL-A) bezeichnete Standard ist eine Erweiterung des Standards 1076-1993, d. h., alle aus dem digitalen VHDL bekannten syntaktischen und semantischen Strukturen können auch in VHDL-AMS verwendet werden. Zusätzlich ermöglicht VHDL-AMS die Modellierung und Simulation analoger Komponenten, nichtelektrischer Komponenten und eine Verbindung dieser mit digitalen Komponenten zu mixed-signal und heterogenen Systemen.

2.2 Struktur eines VHDL-AMS Modells

Ein VHDL-AMS Modell besteht stets aus mindestens zwei Bestandteilen: der Entity und der Architecture.

Entity. Die Entity (engl. Entität, Einheit) repräsentiert das Modell an sich, d. h., der Entity-Name ist der Modellname. Weiterhin enthält die Entity die Schnittstelle des Modells zu seiner Umgebung. Die Schnittstelle wird aus Generics – den Modellparametern – und/oder aus Ports – den Ein- und Ausgängen des Modells – gebildet. Die Generics können nur gelesen werden und dienen der Parametrisierung des Modells. Die Kommunikation des Modells findet über die Ports statt. Dazu gibt es Signal-, Quantity- und Terminal-Ports. Bei Signal- und Quantity-Ports wird zusätzlich ein Richtungsattribut angegeben. Damit wird festgelegt, ob der Anschluß Ein- oder Ausgang ist. Für

Informationen zu den Eigenschaften von Signalen, Quantities und Terminal siehe Abschnitt 2.3.

Darüber hinaus besteht in der Entity die Möglichkeit, Datentypen und Datenobjekte zu deklarieren, Funktionen und Prozeduren zu definieren, sowie passive Anweisungen (Anweisungen, die keinen Einfluß auf Datenobjekte haben) zu beschreiben.

Architecture. In VHDL hat der Nutzer die Möglichkeit, für eine Entity verschiedene Architekturalternativen, z. B. für Design- oder Technologiealternativen, zu beschreiben.

In der Architecture erfolgt die Beschreibung der Modellhierarchie und des Modellverhaltens. Zum Aufbau der Modellhierarchie können hier Submodelle instantiiert bzw. modellinterne Blöcke definiert werden. Zur Verhaltensbeschreibung dienen nebenläufige, simultane und sequentiellen Anweisungen (siehe Abschnitt 2.4). Darüber hinaus besteht auch hier die Möglichkeit, Datentypen und Datenobjekte zu deklarieren und Funktionen und Prozeduren zu definieren.

Package und **Package Body.** Werden dieselben Datentypen, Funktionen und Prozeduren in mehreren Modellen benötigt, bietet es sich an, diese innerhalb von Packages zu beschreiben. Diese können dann in den Modellen geladen und die Datentypen, Funktionen etc. somit eingebunden werden. Ein Package wird dabei in der Regel in Package und Package Body unterteilt. Während ein Package beim Kompilieren der Modelle, die das Package benutzen, bereits kompiliert vorliegen muß, wird der Package Body erst beim Elaborieren (Aufbau der Modellhierarchie vor der Simulation) des Modells hinzugelinkt. Dies hat zur Folge, daß bei Änderungen an einem Package alle Modelle, die dieses Package benutzen, neu kompiliert werden müssen. Daher kann man in VHDL im Package die Konstanten und Funktionen nur deklarieren und ihnen erst im Package Body den Wert zuweisen bzw. den eigentlichen Funktionskörper definieren. Ändert man nun einen Defaultwert oder funktionale Teile einer Funktion (aber nicht deren Interface), muß man außer dem Package Body kein Modell neu kompilieren.

Configuration. Beim Anlegen einer Instanz von Submodellen hat der Nutzer die Möglichkeit, Components zu deklarieren, die einer Fassung entsprechen, in die das zu instantiierende Modell später eingesetzt wird. In der Configuration kann der Nutzer nun entscheiden, welche Architekuralternative (siehe „Architecture“) für die Simulation verwendet oder welches Submodell in welche Fassung „gesteckt“ werden soll. Damit lassen sich verschiedene Designalternativen simulieren, ohne das Model selbst neu kompilieren zu müssen. Außerdem können in der Configuration die an das Modell zu übergebenden Parameter (Generics) modifiziert werden

2.3 Datentypen und Datenobjekte

Datentypen. Ähnlich wie in höheren Programmiersprachen kann der Nutzer auch in VHDL Datentypen definieren. Dabei unterscheidet man zwischen skalaren und zusammengesetzten Typen. Zu den skalaren Typen zählen Enumeration-, Integer-, Real- und physikalische Typen. Zusammengesetzte Typen sind Arrays und Records. Der Inhalt eines Arrays kann wieder ein skalarer oder ein zusammengesetzter Typ sein. Der Inhalt eines Records kann dagegen aus verschiedenen skalaren und zusammengesetzten Typen bestehen. Im stets verfügbaren `standard`-Package werden u. a. die Enumeration-Typen `bit` und `boolean`, die Typen `integer` und `real`, der physikalische Typ

`time` sowie die Arrays `bit_vector` und `real_vector` (ab 1076.1) definiert. Für die Simulation digitaler Systeme stehen außerdem im Package `std_logic_1164` die Datentypen `std_ulogic` und `std_logic` für eine neunwertige Logik (Zustände mit starkem Treiber `'0'`, `'1'`, `'X'`, Zustände mit schwachem Treiber `'L'`, `'H'`, `'W'`, hochohmig `'Z'`, uninitialisiert `'U'` und beliebig `'-'`) zur Verfügung, wobei der Typ `std_logic` mit einer *resolution function* ausgestattet ist (siehe „Signale“).

Bei der Deklaration von Datentypen muß der Nutzer beachten, daß z. B. ein vordefinierter mathematischer Operator nicht immer auf ein Datenobjekt mit selbstdefiniertem Typ anwendbar ist. Alternativ hat der Nutzer die Möglichkeit, Subtypen von einem Datentyp abzuleiten, wobei er den Wertebereich eines skalaren Typs und den Indexbereich eines zusammengesetzten Typs einschränken kann.

Ein Datentyp kann auf Konstanten, Signale, Variablen, Quantities, Funktions- und Prozedurparameter sowie Funktions-Rückgabewerte angewandt werden.

Natures. Der Begriff Nature repräsentiert eine physikalische Domäne und stellt den Datentyp des Terminals dar. Die Definition einer Nature enthält die Deklaration der Datentypen der Zustands- und Flußgröße sowie die Deklaration des zu der entsprechenden Domäne gehörende Referenzknotens (Masseknoten).

Konstanten. Konstanten, Signale, Variablen, Quantities und Terminals werden in der Literatur zu VHDL als Objekte bezeichnet [62]. Dieser Begriff führt aber leicht zu Verwechslungen mit Objekten einer objektorientierten Sprache wie C++ oder Modelica. Da VHDL nicht unmittelbar zu den objektorientierten Sprachen gezählt werden kann, soll hier für diese Elemente von VHDL bzw. VHDL-AMS der Begriff Datenobjekt verwendet werden.

Konstanten werden mit einem Defaultwert deklariert (Ausnahme siehe „Package“) und können während der Simulation nur gelesen werden. Der Defaultwert kann entweder ein konstanter Ausdruck sein oder sich aus Generics errechnen.

Signale. Signale repräsentieren einen zeit- und ereignisdiskreten Werteverlauf. Ihr Wertebereich kann diskret (Integer- oder Enumeration-Typ) oder kontinuierlich (Real-Typ) sein. Signale sind zeitdiskret und werden nur beim Auftreten digitaler Ereignisse, den *events*, berechnet. Werden Signale als Port verwendet, so kann es in Abhängigkeit des Richtungsattributes nur gelesen (`in`), nur geschrieben (`out`) oder gelesen und geschrieben (`inout`, `buffer`, `linkage`) werden. Der einem Signal zugewiesene Wert wird dabei erst am Ende des jeweiligen Delta-Zyklus (siehe Abschnitt 2.5) sichtbar.

Existieren mehrere Treiber für ein Signal, so kann es zu Konflikten kommen, wenn z. B. ein Treiber eine logische `'1'` schreibt, der andere zur selben Zeit eine logische `'0'`. In diesen Fällen kann man Datentypen verwenden, zu denen eine *resolution function* angegeben ist. Man spricht dann von *resolved* Datentypen. Treten bei einem Signal mit einem *resolved* Typ Konflikte auf, berechnet die *resolution function* einen resultierenden Zustand auf dem Signal. Existieren mehrere Treiber auf ein *unresolved* Signal, erzeugt der Compiler eine Fehlermeldung.

Variablen. Variablen dienen als allgemeine Rechengröße in Funktionen und Prozeduren, in digitalen Prozessen oder analogen Proceduralen. Ihr Wertebereich kann wie bei den Signalen diskret (Integer- und Enumeration-Typen) oder kontinuierlich (Real-Typ) sein. Im Gegensatz zu Signalen werden auf Variablen zugewiesene Werte sofort sichtbar.

Quantities. Quantities repräsentieren einen wert- und zeitkontinuierlichen Werteverlauf. Sie müssen immer mit einem Real-Typ deklariert sein. Quantities werden oft auch als nichtkonservative Knoten bezeichnet. Man unterscheidet Port-Quantities, freie Quantities und Branch-Quantities. Port-Quantities dienen der Informationsübermittlung zur Umgebung des Modells, freie Quantities repräsentieren innere „analoge Zustände“ und Branch-Quantities ermöglichen den Zugriff auf Differenz- (`ACROSS QUANTITY`) und Fluß- (`THROUGH QUANTITY`) Größe zwischen zwei konservativen Knoten, den Terminals. Quantities sind die unbekannten Größen im analogen Gleichungssystem. Die Anzahl der Gleichungen muß dabei gleich der Summe der freien Quantities, der Port-Quantities mit dem Richtungsattribut `out` sowie der Branch-Quantities in `THROUGH`-Zweigen sein.

Terminals. Terminals stellen konservative Kirchhoffsche Knoten dar. Ein Terminal repräsentiert immer eine Zustands- bzw. Differenzgröße (z. B. die elektrische Spannung) und eine Flußgröße (z. B. den elektrischen Strom). Auf Differenzgröße und Flußgröße kann jedoch nicht direkt zugegriffen werden. Es müssen erst Branch-Quantities zwischen zwei Terminals deklariert werden, um z. B. auf die Spannung oder den Strom zwischen zwei Knoten zugreifen zu können.

2.4 Verhaltens- und Strukturbeschreibung

VHDL-AMS erlaubt die Betrachtung von verschiedenen Sichtweisen auf ein Modell. So ist es zum einen möglich, das Verhalten der Komponente direkt durch verschiedene Zuweisungen und Differentialgleichungen zu beschreiben. Beschreibt man dabei das physikalische Verhalten, oft auch vereinfacht, so spricht man von physikalischen Verhaltensmodellen. Wird dagegen das Klemmenverhalten ohne Bezug zu tatsächlich ablaufenden physikalischen Prozessen beschrieben, so spricht man von empirischen Verhaltensmodellen.

Zum anderen besteht die Möglichkeit, eine Komponente auf der Basis ihrer Struktur zu beschreiben. Man spricht dabei von Strukturmodellen, wobei der Begriff Struktur hier im makroskopischen Sinne als Modellhierarchie zu verstehen ist. In VHDL werden in der Regel keine geometrischen Strukturen auf dem Chip beschrieben sondern Substrukturen und Hierarchien, z. B. Gatternetzlisten, Funktionsblöcke oder IPs, aus denen eine Komponente aufgebaut ist (vgl. auch Abschnitt 1.3.1). Eine gemeinsame Beschreibung von Verhalten und Struktur in einem Modell ist ebenfalls möglich.

Für die Modellierung von Strukturen gibt es in VHDL verschiedene Syntax-Alternativen. Üblicherweise wird dabei zuerst ein Sockel definiert (`COMPONENT`), in die später das zu instantiierende Modell eingesetzt wird. Dieser Sockel wird mittels Generic Map und Port Map Statement mit dem Modell verbunden. Zu diesem Zeitpunkt muß die zu instantiierende Komponente noch nicht kompiliert vorliegen. Mit Hilfe der Configuration kann dann am Ende des Modellierungsprozesses

entschieden werden, welches Submodell in den Sockel „gesteckt“ wird. Alternativ kann man auch die Subkomponente direkt instantiieren, d. h. gewissermaßen „direkt einlöten“, wodurch keine Component und keine Configuration notwendig ist.

2.5 Simulationsablauf

Die Simulation gliedert sich in drei Schritte. Im ersten Schritt, der Elaboration, wird das Simulationsmodell aus allen kompilierten VHDL-Modellen aufgebaut (gelinkt). Der zweite Schritt ist die Initialisierung. Hier werden alle Default-Werte zugewiesen und anschließend alle digitalen Signalzuweisungen ausgeführt und alle Prozesse einmal gestartet, bis diese durch ein wait-Statement angehalten werden. Danach wird der Arbeitspunkt der analogen Komponenten bestimmt. Im dritten Schritt, der Execution-Phase, werden alle Zuweisungen, Prozesse und das Differentialgleichungssystem wiederholt berechnet, bis das Ende der Simulation erreicht ist. Im Detail gestaltet sich die Execution-Phase wie folgt:

- Berechnung des Differentialgleichungssystems,
- Setzen der aktuellen Simulationszeit auf die im vorherigen Schritt bestimmte nächste Simulationszeit,
- Berechnen der Signalzuweisung von Signalen, deren Treiber zur aktuellen Simulationszeit aktiv sind. Der aktuelle Signalwerte wird aber noch nicht geändert. Ergibt sich eine Wertänderung, findet auf diesem Signal ein *event* statt.
- Ausführen von Prozessen, die auf Ereignisse auf Signalen reagieren sollen,
- Änderung der Signalwerte, auf denen ein *event* stattgefunden hat,
- Setzen des nächsten geplanten Simulationszeitpunktes auf den Zeitpunkt, zu dem der erste Signaltreiber im System erneut aktiv wird. Ist dieser Zeitpunkt derselbe Zeitpunkt wie die aktuelle Simulationszeit, so wird ein sog. Delta-Zyklus ausgeführt. Bei einem Delta-Zyklus wird die Simulationszeit um ein infinitesimal kleines Inkrement erhöht. Zusammen mit der getrennten Berechnung und Zuweisung von Signalen bildet der Delta-Zyklus die Grundlage zur Simulation der hardwarespezifischen Eigenschaft der parallelen Signalberechnung.

Aus dieser Simulationsreihenfolge ergibt sich die Besonderheit, daß analoger Gleichungslöser und der digitale *event*-basierte Simulator unterschiedliche Zeitachsen benutzen und damit zu unterschiedlichen Zeitpunkten rechnen können. Dies muß in der Modellierung wie folgt beachtet werden:

- Eine Quantity kann nicht in einem *concurrent* Statement einem Signal zugewiesen werden, da eine Quantity keine *events* besitzt und somit kein Treiber für ein Signal im Sinne von VHDL-AMS sein kann. Man muß statt dessen entweder die Quantity in einem Prozeß abtasten oder durch Verwendung des `'Above` Attributs im Sinne eines Schwellwertschalters digitale *events* erzeugen.
- Die Verwendung eines Signals in einem *simultaneous* Statement ist problematisch, da Analog- und Digitalsimulator zu unterschiedlichen Zeitpunkten rechnen und somit Zustandswechsel auf dem Signal verzögert oder gar nicht vom Analogsimulator erfaßt werden. Wird

eine solche Modellierung benötigt, ist mit Hilfe des BREAK-Statements dem analogen Simulator die durch den Pegelwechsel auf dem Signal erzeugte Diskontinuität anzuzeigen. Das BREAK-Statement ist dabei eine leistungsfähige Anweisung, um mit VHDL-AMS Diskontinuitäten in analogen Systemen, wie z. B. mechanische Anschläge u. a., beschreiben und simulieren zu können. Alternativ hat man die Möglichkeit, eine Quantity einem Signal mittels des `'Ramp` oder `'Slew` Attributs mit definierten Anstiegs- und Abfallzeiten bzw. definierten Flankensteilheiten folgen zu lassen.

3 Multi-Architecture-Modellierung

3.1 Ausgangsproblem

Gegeben sei ein Meßsystem, bei dem die zu erfassende physikalische Größe von einem Sensor aufgenommen und in ein elektrisches Signal umgewandelt wird. Dieses elektrische Signal durchläuft eine analoge Signalvorverarbeitung, wird digitalisiert und anschließend von einer digitalen Signalverarbeitung klassifiziert. Im Entwurfsprozeß soll nun der Einfluß von Sensorparametern und von Eigenschaften der analogen Signalverarbeitung auf das Klassifikationsergebnis untersucht werden. Dazu ist ein Modell des kompletten Systems notwendig. Ein System, das nur aus detaillierten Komponentenmodellen besteht, wird einen hohen Rechenaufwand hervorrufen, ein System aus abstrakten Modellen die Wirklichkeit zu ungenau widerspiegeln. Wie bereits in Abschnitt 1.5 angedeutet, lassen sich die geplanten Verifikationsaufgaben aber auch ausführen, indem man z. B. nur für diejenigen Komponenten das detaillierte Modell einsetzt, deren Einfluß auf das Systemverhalten bestimmt werden soll. Will man die Kommunikation zwischen Komponenten untersuchen, so kann man für eine oder mehrere der beteiligten Komponenten das detaillierte Modell verwenden. Für die anderen Komponenten des Systems können deren abstrakte Modelle eingesetzt werden.

In Bild 3-1 besteht die Verifikationsaufgabe darin, den Einfluß von Sensorparametern auf das Simulationsergebnis zu bestimmen. Für den Sensor wird ein detailliertes, genaues Modell verwendet, für analoge und digitale Signalverarbeitung ein abstraktes, schnelles Modell.

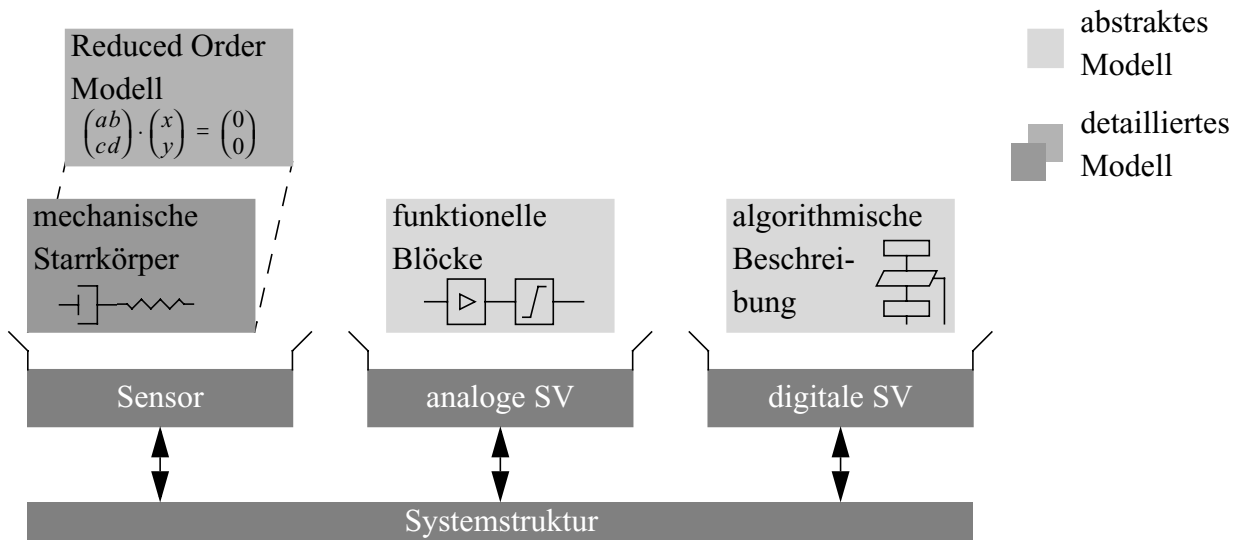


Bild 3-1 Modellkonfiguration zur Bestimmung des Einflusses von Sensorparametern

Eine mögliche Konfiguration zur Untersuchung von Eigenschaften der analogen Signalverarbeitung (wie z. B. Signal-zu-Rauschverhältnis oder Klirrfaktor) und deren Einflüsse auf das Systemverhalten ist in Bild 3-2 dargestellt. Dazu konfiguriert man das Modell so, daß für die analoge Signalverarbeitung ein genaueres Modell, für Sensor und digitale Signalverarbeitung ein abstrakteres Modell verwendet wird.

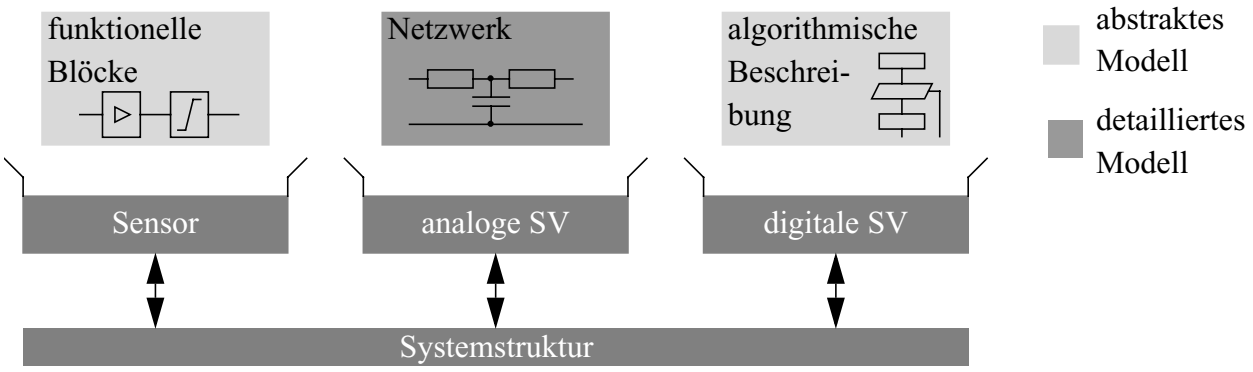


Bild 3-2 Modellkonfiguration zur Bestimmung des Einflusses von Eigenschaften der analogen Signalverarbeitung

Ist dagegen zu untersuchen, wie mehrere Komponenten interagieren, so sind für diese Komponenten deren detaillierte Modelle einzusetzen. Bild 3-3 zeigt die Konfiguration des Beispielsystems, wenn die Kommunikation zwischen analoger und digitaler Signalverarbeitung verifiziert werden soll.

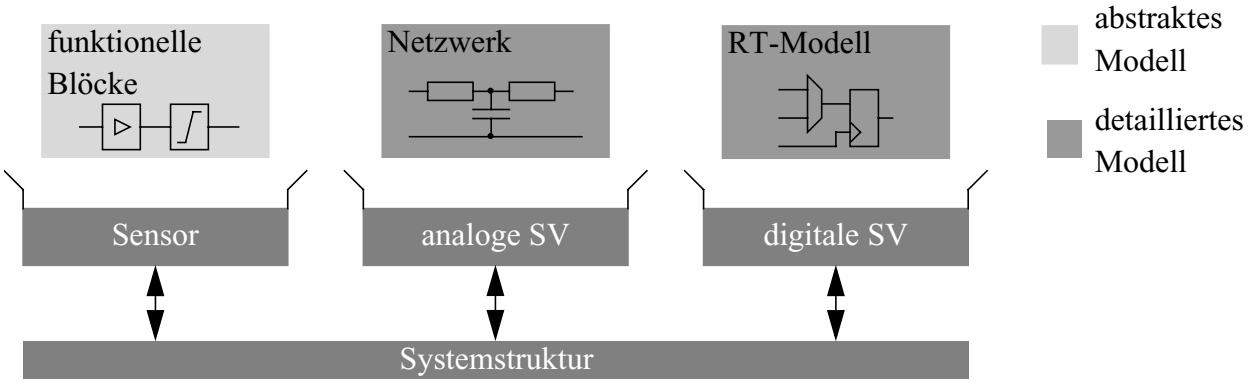


Bild 3-3 Modellkonfiguration zur Verifikation der Kommunikation zwischen analoger und digitaler Signalverarbeitung

Je mehr detaillierte Komponenten-Modelle in der Simulation Verwendung finden, um so länger wird die Simulation dauern. Mit der Möglichkeit, für die Komponenten abstraktere oder weniger abstrakte Modelle einzusetzen, kann man das System so konfigurieren, daß immer nur für diejenigen Komponenten genauere Modelle eingefügt werden, die für die Simulation eines bestimmten Sachverhaltes notwendig sind. Somit kann man das Systemmodell zwischen Genauigkeit und Geschwindigkeit skalieren.

Nun bestimmt aber in der Regel der Abstraktionsgrad eines Modells den Abstraktionsgrad der Modellschnittstellen. So entstehen in einem Top-Down-basierten Systementwurf zu Beginn Modelle auf einem hohen Abstraktionsniveau. Das Abstraktionsniveau der Modelle verringert sich mit Fortschreiten des Entwurfsprozesses, was dazu führen kann, daß die Schnittstellen der Modelle mit jedem Entwicklungsschritt auf Grund der damit verbundenen Verringerung des Abstraktionsniveaus angepaßt werden müssen. Dies ist ein zeitaufwendiger und fehleranfälliger Prozeß, da bei

der Modifikation einer Schnittstelle alle anderen mit dieser Schnittstelle verbundenen Modellschnittstellen ebenfalls angepaßt werden müssen.

Bild 3-4 verdeutlicht das Problem anhand eines Beispiels, bei dem zunächst analoge Modelle auf der Basis funktioneller Blöcke beschrieben werden. In VHDL-AMS kommen hier als Interface nichtkonservative Knoten (`QUANTITY`) zum Einsatz. Während des Entwurfsprozesses wird die rechte Komponente zu einer Netzwerkbeschreibung weiterentwickelt, die linke dagegen nicht. Die weiterentwickelte, verfeinerte Komponente benutzt nun als Interface konservative Knoten (`TERMINAL`) während die unveränderte Komponente nach wie vor Quantities benutzt. Beide Modelle können dadurch nicht mehr miteinander verbunden werden. Ähnliche Probleme treten auch bei digitalen oder nichtelektrischen Modellen auf.

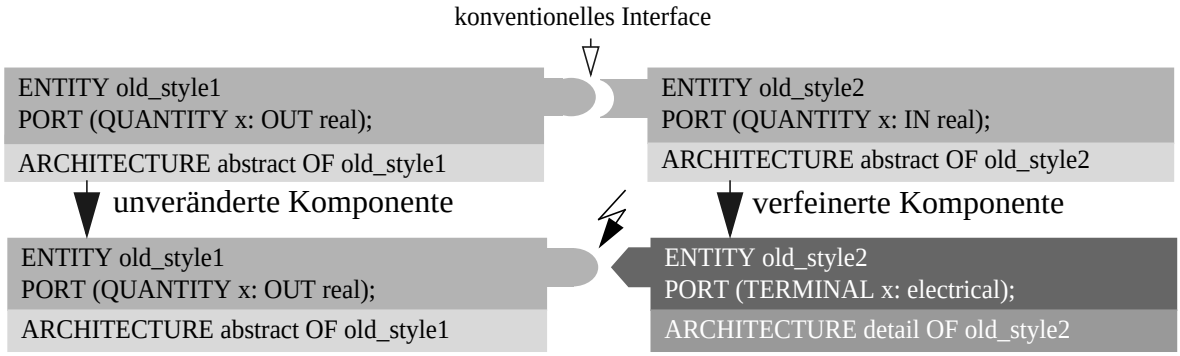


Bild 3-4 Problem beim Verbinden von Modellen auf unterschiedlichen Abstraktionsebenen

Im Rahmen dieser Arbeit entstand daher zunächst auf der Basis der Hardwarebeschreibungssprache VHDL-AMS ein neuer methodischer Ansatz – die Multi-Architecture-Modellierung (MAM). Ziel dieser Methode ist die Schaffung einer einheitlichen Schnittstelle über Abstraktionsebenen hinweg, so daß sich in einem hierarchisch höherliegenden Modell (z. B. dem Systemmodell) Submodelle (z. B. Komponentenmodelle) auf unterschiedlichen Abstraktionsebenen instantiieren lassen und diese gegen abstraktere oder weniger abstrakte Alternativ-Modelle – die multiplen Architekturen – ohne Änderungen der Schnittstellen ausgetauscht werden können.

Als Randbedingungen sollen dabei gelten:

- Der entstandene Ansatz muß leicht anwendbar sein.
- Er darf nur einen geringen Mehraufwand bei der Modellierung und Simulation hervorrufen, und
- er muß 100%ig konform zum VHDL-AMS-Standard sein.

Der Einsatz dieser Methode ist dabei primär für den Einsatz im Entwurf heterogener Systeme vorgesehen und weniger für den Entwurf rein digitaler Systeme, da bei digitalen Systeme durch den Einsatz von Synthesewerkzeugen der Übergang zwischen Abstraktionsebenen weitgehend automatisiert ist.

3.2 Grundlagen der Multi-Architecture-Modellierung

Beim Entwurf digitaler Systeme erlaubt VHDL die Beschreibung mehrerer Architekturen (`ARCHITECTURE`) für eine Schnittstellenbeschreibung (`ENTITY`). Mit der Erweiterung von VHDL zu VHDL-AMS steht diese Eigenschaft von VHDL auch für die Beschreibung analoger und heterogener Systeme zur Verfügung. Wurde diese Eigenschaft bislang für Entwurfsalternativen digitaler Komponenten oder eines digitalen Teilsystems angewandt, so bietet sich diese Eigenschaft nun auch an, mehrere Architekturen für ein Komponentenmodell eines heterogenen Systems auf unterschiedlichen Abstraktionsebenen zu hinterlegen. Als Voraussetzung dafür ist allerdings die im vorigen Abschnitt beschriebene Schnittstellenproblematik zu lösen. Erste Lösungsansätze wurden 2001 vom Verfasser in [8] veröffentlicht, ein umfassenderer Überblick zur Schnittstellenproblematik der MAM ist in [7] dargelegt.

3.2.1 Digitale Schnittstellen

Für digitale Schnittstellen finden Signale Verwendung. Der Abstraktionsgrad hat dabei einen Einfluß auf den verwendeten Datentyp. So können z. B. auf funktioneller oder algorithmischer Ebene Datentypen wie `integer` oder `real` verwendet werden. Im Gegensatz dazu sind auf Register-Transfer-Ebene (RT-Ebene) Vektoren der Datentypen `bit` oder `std_logic` bzw. Vektoren dieser Datentypen üblich.

Für die MAM sind nun zwei Ansätze denkbar: Entweder man verwendet die Schnittstelle, die das Modell auf hohem Abstraktionsniveau besitzt auch für die weiteren Modelle auf niedrigem Abstraktionsniveau, oder man benutzt die Schnittstelle des detaillierteren Modells bereits auf hoher Abstraktionsebene.

Da man bei einem Top-Down-Entwurf die Schnittstelle des detaillierten Modells normalerweise auf hoher Abstraktionsebene noch nicht kennt, wurde zunächst versucht, die Schnittstelle des abstrakteren Modells im detaillierten Modell beizubehalten. Dies führt zu

- einem geringen Modellierungs-Mehraufwand,
- u. U. zu zusätzlichen Delta-Zyklen während der Simulation durch Datentypkonvertierungen und
- einem unvermeidbaren Verlust von Zustandsinformationen (z. B. der Zustände `'X'` und `'Z'`) bei mehrwertigen Logik-Datentypen.

Der Verlust der Zustandsinformationen im Modell auf RT-Ebene ist unumgänglich, da Ein- und Ausgangsdaten auf der Basis einer mehrwertigen Logik die abstrakte Schnittstelle, die z. B. den Datentyp `integer` oder `real` besitzt, passieren müßten. Dies stellt einen entscheidenden Nachteil bei der Anwendung der abstrakten Schnittstellen im detaillierten Modell dar.

Eine Alternative, die Verwendung der detaillierten Schnittstelle auf hoher Abstraktionsebene, setzt die Bekanntheit dieser Informationen am Anfang des Entwurfs voraus. Dafür müßte der Standard-Top-Down-Entwurfsablauf modifiziert werden. Betrachtet man diesen Ansatz näher, so bleiben die Aspekte Modellierungs-Mehraufwand und zusätzlichen Delta-Zyklen gegenüber dem vorherigen Ansatz unverändert. Im Gegensatz zum ersten Ansatz entsteht der Verlust von Zustandsinformationen nicht mehr zwangsweise, sondern nur noch, wenn die zu verarbeitende Information ein

noch im Systemmodell vorhandenes abstraktes Modell durchläuft. Dies liegt allerdings in der Natur der Abstraktion und ist nicht dem Ansatz selbst geschuldet.

Die folgende Tabelle mit Quellcode soll den zweiten Modellierungsansatz im Vergleich zur konventionellen Modellierung noch einmal verdeutlichen:

Ansatz mit MAM	konventioneller Ansatz

`PORT( SIGNAL in1: IN bit_vector`
`  SIGNAL out1: OUT bit_vector);`
`SIGNAL s1,s2: integer;`
`s1 <=bit_vector2int(in1);`
`s2 <=s1 AND 1;` -- Modellfunktion
`out1 <=int2bit_vector(s2);`
	`PORT( SIGNAL in1: IN integer;`
	`  SIGNAL out1: OUT integer);`
	
	`out1 <=in1 AND 1;` -- Modellfunktion

Tabelle 3-1 Quellcodefragmente einer digitalen Komponente auf hoher Abstraktionsebene

Ansatz mit MAM	konventioneller Ansatz

`PORT( SIGNAL in1: IN bit_vector;`
`  SIGNAL out1: OUT bit_vector);`
`out1 <=in1 AND "0001";` -- Modellfunktion
	`PORT( SIGNAL in1: IN bit_vector;`
	`  SIGNAL out1: OUT bit_vector);`
	`out1 <=in1 AND "0001";` -- Modellfunktion

Tabelle 3-2 Quellcodefragmente einer digitalen Komponente auf niedriger Abstraktionsebene

Wie zu erkennen ist, erweist sich dieser Ansatz als anwendbar und bedeutet für digitale Schnittstellen die Verwendung der Datentypen `bit_vector` oder eines Vektors einer mehrwertigen Logik (z. B. `std_logic_vector`) bereits im ersten abstrakten Modell. Gegenüber der Anwendung der Datentypen `integer` oder `real` erzeugt dies nur einen sehr geringen Mehraufwand in den abstrakten Modellen. Zustände wie `'X'`, `'Z'` usw. werden außerhalb der Modelle korrekt propagiert, im abstrakten Modell kann die Handhabung entsprechend der Anforderungen an das Modell implementiert werden.

Ein dritter zu diskutierender Ansatz ist die gemeinsame Verwendung der abstrakten und der detaillierten Schnittstelle. Dabei besitzt jede Komponente am Eingang die für sie typische Schnittstelle, am Ausgang stellt sie Schnittstellen sowohl in abstrakter als auch in detaillierter Form bereit Bild 3-5.

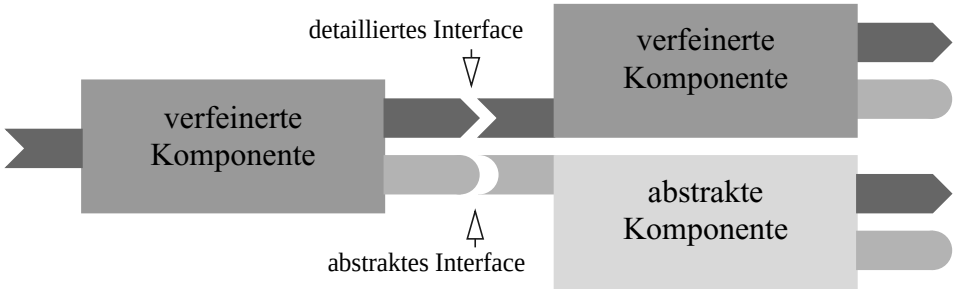


Bild 3-5 Gemeinsame Verwendung von abstrakten und detaillierten Schnittstellen

Auf diese Art und Weise kann im System jede Komponente gegen eine abstraktere oder weniger abstrakte Komponente ausgetauscht werden. Allerdings zeigen sich bei diesem Verfahren die folgenden Probleme:

- Es ist sehr aufwendig, da immer verschieden abstrakte Ausgänge eingefügt werden müssen.
- Es läßt sich bei inout-Ports nur mit erheblichem Mehraufwand benutzen.
- Die Modellierung von *wired OR* oder *wired AND* Verknüpfungen über *resolution functions* ist nicht möglich.

Darüber hinaus müssen wie beim Ansatz des Vorziehens der abstrakten Schnittstelle die detaillierten Schnittstellen bereits auf hoher Abstraktionsebene bekannt sein, so daß dieser Ansatz keine brauchbare Lösung des Ausgangsproblems darstellt.

Ein Ableiten der detaillierten aus der abstrakten Schnittstelle, wie im Abschnitt 5.3, S. 104 ff. für SystemC beschrieben, funktioniert in VHDL-AMS nicht, da VHDL-AMS keine objektorientierte Sprache im Sinne der Informatik ist. Das Konzept der Kanäle (*Channels*) und die Verwendung von *Interface-Method-Calls* existieren im Gegensatz zu SystemC in VHDL ebenfalls nicht.

Nach der Diskussion obiger drei Lösungsansätze wird für digitale Schnittstellen die Verwendung der detaillierten Schnittstelle bereits auf hoher Abstraktionsebene favorisiert, wobei eine Modifikation des Top-Down-Entwurfes notwendig wird.

3.2.2 Protokolle digitaler Schnittstellen

Digitale Schnittstellen werden nicht nur durch die Art der über sie übertragenen Daten charakterisiert, sondern auch dadurch, wie diese Daten übertragen werden, d. h., es spielt eine entscheidende Rolle wie Sender und Empfänger der Daten miteinander kommunizieren. So kann im Modell auf hoher Abstraktionsebene der Sender ein Datenwort oder -paket an die Empfängerkomponente übergeben, während auf niedriger Abstraktionsebene die Kommunikation aus einer speziellen Abfolge von übertragenen Bits besteht. Auch diesem Aspekt muß der Ansatz der Multi-Architecture-Modellierung gerecht werden.

Dazu müssen Konverter zur Übersetzung der Kommunikationsprotokolle – ähnlich wie in [87] – in die Modelle eingefügt werden. Dabei besteht prinzipiell die Möglichkeit, auf abstrakter Ebene im Sender Konverter einzufügen, die abstrakte Datenworte oder -pakete auf Bit-Level umsetzen und diesen Bitstrom an den Empfänger übermitteln, wo ein entsprechender Konverter den Bitstrom wieder zu Datenworten umsetzt. Alternativ besteht die Möglichkeit, die verfeinerten Modelle mit Konvertern auszustatten, die den Bitsrom zu Datenworten zusammenfassen. Dieser wird in abstrakter Form übertragen und anschließend im Empfänger von einem Konverter wieder in einen Bitstrom umgewandelt.

Ausgehend von

- den Problemen und Lösungsansätzen aus Abschnitt 3.2.1,
- dem Ziel, mit der MAM durch das Systemmodell eine Testbench für verfeinerte Modelle zu schaffen und

- der möglichen Gefahr eines fehlerhaften Designs falls der Konverter unbeabsichtigte Wechselwirkungen mit der RT-Beschreibung hervorrufen sollte,

ergibt sich, daß die Kommunikation auf hoher Abstraktionsebene bereits so zu erfolgen hat, wie dies später auf niedriger Abstraktionsebene der Fall sein soll.

Da es dem Entwerfer nicht zuzumuten ist, auf hoher Abstraktionsebene ein komplexes digitales Protokoll bit- und taktgenau zu modellieren, müssen entsprechende Prozeduren in Form von Abstraktionswandlern eingefügt werden, die die Protokolle zwischen unterschiedlichen Abstraktionsebenen übersetzen (Bild 3-6).

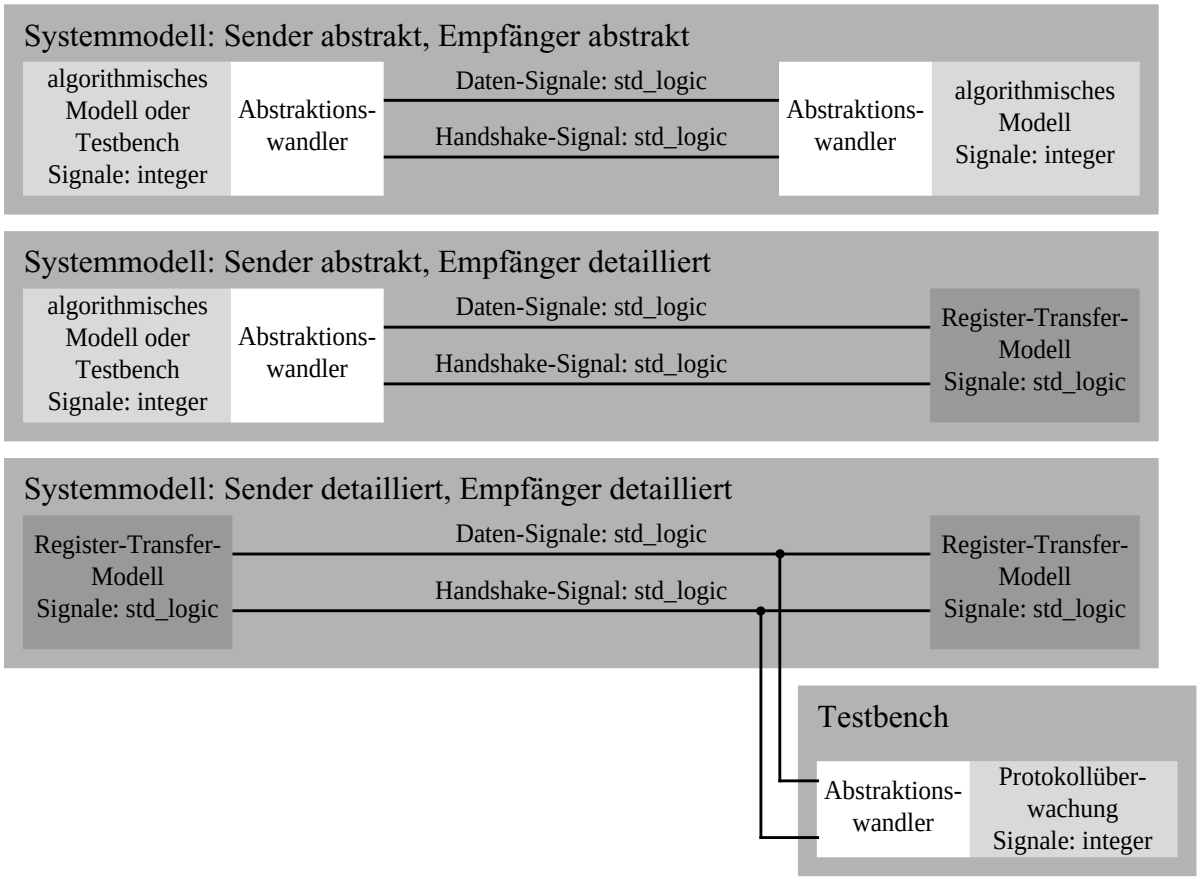


Bild 3-6 Mögliche Anwendungen von Abstraktionswandlern zur Übertragung digitaler Protokolle

Das Erstellen der Prozeduren erfolgt manuell und bedeutet einen Mehraufwand im Entwurfsprozeß. Allerdings kann man diese Prozeduren in Bibliotheken ablegen, so daß beim nächsten Entwurf mit einem standardisierten Übertragungsprotokoll die passenden Prozeduren zur Verfügung stehen und keine zusätzliche Arbeit erforderlich ist.

Die Protokoll-Abstraktionswandler-Prozeduren übersetzen abstrakte, für den Entwerfer direkt interpretierbare Stimuli-Datenworte in den der jeweiligen Schnittstelle entsprechenden Datenstrom, bzw. sie übersetzen den Datenstrom einer Schnittstelle in vom Entwerfer direkt auswertbare Datenworte. Daher eignen sie sich ebenfalls hervorragend zur Testmustergenerierung in einer Testbench und zur Überwachung der Datenübertragung auf niedriger Abstraktionsebene. Des weiteren

besteht die Möglichkeit, diese Prozeduren mit den in Abschnitt 3.2.4 beschriebenen speziellen A/D- und D/A-Wandlern zu kombinieren und so z. B. den Einfluß von Leitungslängen, Dämpfungen und Übersprechen bereits auf hoher Abstraktionsebene mit zu berücksichtigen.

Die automatische Generierung dieser Protokoll-Abstraktionswandler-Prozeduren ist nicht Bestandteil dieser Arbeit. Eine Kombination mit bestehenden Ansätzen zur automatische Generierung ist jedoch prinzipiell möglich. In Abschnitt 5.2.2, S. 95 wird dies für die Protokollbeschreibungssprache SVE [80] kurz erläutert. Auch die Kombination z. B. mit der Spezifikationsbeschreibungssprache ESTELLE [88] in Kombination mit einem Übersetzungsverfahren nach VHDL [89] ist vorstellbar. In [90] und [91] werden Verfahren zur Kommunikationsbeschreibung und -synthese im Rahmen des Hardware/Software-Codesigns vorgestellt, die ebenfalls VHDL-Code generieren.

Der Einsatz der manuell erstellten Protokoll-Abstraktionswandler-Prozeduren eignet sich primär für Übertragungsprotokolle mit geringer bis mittlerer Komplexität wie RS232-Schnittstelle oder I^2C-Bus. Zum Testen des Ansatzes wurden die Abstraktionswandler-Prozeduren für diese Schnittstellen implementiert. Für komplexe Protokolle wie z. B. USB oder Ethernet eignet sich dieser Ansatz weniger, da hier die Erstellung der Abstraktionswandler-Prozeduren einen erheblichen Mehraufwand bedeutet. Außerdem werden solche komplexen Transceiver in der Regel als verifizierte IP-Komponenten eingekauft, so daß mehrere Iterationszyklen über Abstraktionsebenen hinweg hier in der Regel nicht notwendig sind und der Einsatz der MAM somit nicht zur Steigerung der Entwurfseffizienz beiträgt.

Wie bereits in Abschnitt 3.1, S. 43 angedeutet, ist der Einsatz der MAM weniger für den Entwurf rein digitaler Systeme konzipiert, sondern für den Entwurf heterogener Systeme. Bei solchen Systemen finden gerade die oben beschriebenen einfacheren Schnittstellen nach wie vor häufig Anwendung, während z. B. USB oder Ethernet innerhalb solcher Systeme eher selten anzutreffen sind. Wenn derartige Schnittstellen benutzt werden, dann in der Regel als Interface zu einem externen Rechner, wobei entweder fertige Transceiver-IC, Ethernet- bzw. USB-fähige Mikrocontroller oder die zuvor erwähnten IP-Komponenten zum Einsatz kommen. Für die RS232-Schnittstelle soll der Aufbau der Prozeduren und deren Anwendung im folgenden demonstriert werden, für den I^2C-Bus befinden sich die Prozeduren im Anhang A.

Quelltext 3-1 zeigt die Prozeduren `send_data` und `receive_data` des Abstraktionswandlers für die RS232-Schnittstelle. Diese benutzen ihrerseits Funktionen aus dem Package `MAM_Support` welches in Abschnitt 3.2.4 noch näher erörtert wird. Die Anwendung dieser Prozeduren in einem komplexen System ist im Abschnitt 4.2, S. 79 ff. beschrieben.

Die Abstraktionswandler-Prozeduren für eine serielle RS232-Schnittstelle sind in einem eigenen Package „serial“ beschrieben. Dabei kommen VHDL-Prozeduren zum Einsatz, da diese gegenüber Funktionen den Vorteil haben, daß sie direkt auf Signale schreibend zugreifen können und sie die Abarbeitung des Prozesses, in dem die Prozeduren gerufen werden, unterbrechen können. Dies ermöglicht es dem Entwerfer, innerhalb von Prozessen eine effektive und im Rahmen von VHDL komfortable Kommunikation auf algorithmischer Ebene zu beschreiben, ohne sich um taktbezogene Timings etc. kümmern zu müssen.

```
use work.mam_support.all;
package body serial is
   --Definition der Schnittstellenparameter
   constant baud: integer:=9600;
   constant parity: parity_type:=none;
   constant stopbits: integer range 1 to 2:=1;
   constant delay:time:=1 sec / baud;
   --Procedure zum Senden von Daten
   procedure send_data (   constant data2send: in integer range 0 to 255;
                           signal   TxD: out bit;
                           signal   CTS: in bit;
                           constant nowait: in boolean:=true) is
      variable data:bit_vector(7 downto 0);
      variable one_count:integer:=0;
   begin
      one_count:=0;
      if CTS='0' and not nowait then  --Wenn bei Vierdrahtverbindung CTS auf '0' ist und
         wait on cts;                 --nowait nicht gesetzt ist, dann darauf warten, daß der
      end if;                         --Receiver bereit ist (CTS='1')
      if CTS='1' then
         TxD<='0';                    --Startbit senden
         data:=int2bv(data2send);     --Daten von integer nach bit_vector wandeln
         wait for delay;
         for i in 0 to 7 loop
            TXD<=data(i);             --Übermitteln der Daten
            …
            wait for delay;           --Entsprechend der Baudrate warten
         end loop;
         …                            --Parity und Stop-Bits übermitteln
      else
         report "Receiver not ready" severity note;
      end if;
   end;
   …
   --Procedure zum Empfangen von Daten
   procedure receive_data (signal data_received: out integer range 0 to 255;
                           signal RxD: in bit;
                           signal RTS: out bit) is
      variable data:bit_vector(7 downto 0);
      variable one_count:integer:=0;
      variable parity_bit:bit;
   begin
      RTS<='1';                       --Bereitschaft zum Empfangen von Daten anzeigen
      wait on RxD until RxD='0';      --auf ankommendes Startbit warten
      wait for delay / 2;
      for i in 0 to 7 loop
         wait for delay;
         data(i):=RxD;                --ankommende Datenbits sammeln
         …
      end loop;
      …                               --Parity und Stopbits empfangen
      RTS<='0';                       --keine Bereitschaft zum Empfangen von Daten
      data_received:=bv2int(data);    --empfangenen bit_vector nach integer wandeln
      wait for delay/2;
   end;
end;
```

Quelltext 3-1 Ausschnitt aus den Prozeduren „send_data“ und „receive_data“ für die RS232-Schnittstelle

Das Package „serial“ enthält darüber hinaus die Definitionen der Parameter einer seriellen Schnittstelle wie Baudrate, Parity-Bit usw. Für die Kommunikation sorgen drei Prozeduren `send_data`, `receive_data` und `init`. Letztere Prozedur dient lediglich der Initialisierung des Datenausgangs TxD (Transmit Data) und der Steuerleitung RTS (Ready to Send). Die Prozeduren unterstützen sowohl eine Zweidrahtkommunikation als auch eine Vierdrahtverbindung mittels RTS/CTS Hardware-*Handshake* [92]. Dabei arbeitet das *Handshake* auf der Basis einer bidirektionalen Kommunikation zweier Datenendeinrichtungen.

3.2.3 Analoge elektrische und nichtelektrische Schnittstellen

Abstrakte analoge Komponenten werden oft als funktionelle Blöcke beschrieben [93]. Als Interface-Objekt kommen dabei in VHDL-AMS sog. Quantities (`QUANTITY`) zum Einsatz. Diese repräsentieren einen zeit- und wertkontinuierlichen Informationsverlauf. Da auf sie die Kirchhoffschen Gesetzte nicht angewandt werden können, bezeichnet man Quantities auch als nichtkonservative Knoten. Die Informationsübertragung ist rückwirkungsfrei. Im Gegensatz dazu benutzt ein Modell auf niedrigerem Abstraktionsniveau Terminals (`TERMINAL`) als Interface. Terminals enthalten eine `ACROSS` und eine `THROUGH QUANTITY` (z. B. Spannung und Strom) und erfüllen die Kirchhoffschen Gesetze. Man bezeichnet sie deshalb auch als konservative Knoten. Mehrere konservative Knoten bilden zusammen ein Netzwerk, auf dem jeder Knoten eine Wirkung auf andere Knoten ausüben kann.

Ein gemeinsames Interface, das sowohl funktionelle Blöcke als auch Netzwerke miteinander verbinden soll, muß demnach entweder aus Terminals oder aus zwei Quantities (je eine für die `ACROSS` und `THROUGH` Größe) bestehen. Da man im Rahmen eines Top-Down-Entwurfes analoger Systeme in der Regel mit funktionellen Blöcken beginnt, wurde zunächst ein Ansatz auf der Basis von zwei Quantities untersucht. Die entsprechenden Modellierungs-Varianten (konventionell und mit vereinheitlichter Schnittstelle) sind in Tabelle 3-3 und Tabelle 3-4 gegenüber gestellt.

Ansatz mit MAM	konventioneller Ansatz

`PORT( QUANTITY q1i,q1u: IN real;`
`  QUANTITY q2i,q2u: OUT real);`
`q2u==F(q1u, q1u'dot, t);  -- Modellfunktion`
`q2i==0.0;`
	`PORT( QUANTITY q1: IN real;`
	`  QUANTITY q2: OUT real);`
	`q2==F(q1, q1'dot, t);  -- Modellfunktion`

Tabelle 3-3 Quellcodefragmente einer analogen Komponente auf hoher Abstraktionsebene

Ansatz mit MAM	konventioneller Ansatz

`PORT( QUANTITY q1i,q1u: IN real;`
`  QUANTITY q2i,q2u: OUT real);`
 	
`q1i + q2i==0.0;`
`q1u - q2u==q1i * R;  -- Modellfunktion`
	`PORT( TERMINAL t1, t2: electrical);`
	
	`QUANTITY u ACROSS i THROUGH t1 TO t2;`
	`i==u / R;  -- Modellfunktion`

Tabelle 3-4 Quellcodefragmente einer analogen Komponente auf niedriger Abstraktionsebene

Aus diesem Ansatz ergeben sich die folgenden Eigenschaften:

- Im abstrakten Modell wird für jede ursprüngliche Ausgangs-Quantity ein zusätzliches *simultaneous statement* notwendig.
- Im Netzwerkmodell muß der Entwerfer das Kirchhoffsche Gesetz explizit beschreiben.

Dies verursacht einen nicht unerheblichen Mehraufwand in Modellierung und Simulation, ist fehleranfällig und kann zur Instabilität und Verlangsamung der Simulation führen.

Beim zweiten Ansatz, der Verwendung von Terminals auf hoher Abstraktionsebene, wird eine zu übertragende Quantity als ideale Spannungs- oder Stromquelle zwischen dem Terminal und dem dazugehörigen Masseknoten beschrieben. In der Empfängerkomponente wird dann die übertragene Quantity als Spannung (bei Bedarf auch als Strom) zwischen dem Terminal und dem jeweiligen Masseknoten ermittelt. Ein entsprechendes Beispiel verdeutlichen Tabelle 3-5 und Tabelle 3-6.

Ansatz mit MAM	konventioneller Ansatz

`PORT( TERMINAL t1, t2: electrical);`
 	
`QUANTITY q1 ACROSS t1;`
`QUANTITY q2 ACROSS i2 THROUGH t2;`
`q2==F(q1, q1'dot, t);` -- Modellfunktion
	`PORT( QUANTITY q1: IN real;`
	`QUANTITY q2: OUT real);`
	
	`q2==F(q1, q1'dot, t);` -- Modellfunktion

Tabelle 3-5 Quellcodefragmente einer analogen Komponente auf hoher Abstraktionsebene

Ansatz mit MAM	konventioneller Ansatz

`PORT( TERMINAL t1, t2: electrical);`
`QUANTITY u ACROSS i THROUGH t1 TO t2;`
`i==u / R;` -- Modellfunktion
	`PORT( TERMINAL t1, t2: electrical);`
	`QUANTITY u ACROSS i THROUGH t1 TO t2;`
	`i==u / R;` -- Modellfunktion

Tabelle 3-6 Quellcodefragmente einer analogen Komponente auf niedriger Abstraktionsebene

Im Gegensatz zum Ansatz mit zwei Quantities ergeben sich hier

- nur ein geringer Mehraufwand,
- eine leichte Anwendbarkeit und
- eine geringe Fehleranfälligkeit.

Somit hat es sich auch bei analogen Schnittstellen als ungünstig erwiesen, die auf hohem Abstraktionsniveau verwendete Schnittstelle in dem Modell auf niedrigem Abstraktionsniveau weiter zu benutzen. Die Verwendung von Terminals für analoge Schnittstellen auf allen Abstraktionsebenen dagegen ist möglich. Bild 3-7 verdeutlicht nochmals die Anwendung dieses Ansatzes im Rahmen der MAM beim Entwurf analoger Schnittstellen.

Die z. B. als funktionelle Blöcke beschriebenen Modelle erhalten als Interface im Gegensatz zu Bild 3-4 keine Quantity sondern ein Terminal. Die Anpassung innerhalb des Modells an dieses Interface besteht lediglich aus der Definition von Branch-Quantities (`ACROSS` und `THROUGH QUANTITY`). Wird nun eine Komponente verfeinert, die andere aber nicht, so können beide Modelle immer noch problemlos miteinander verbunden werden.

Das in diesem Abschnitt beschriebene Verfahren läßt sich unter Verwendung der entsprechenden Nature auch auf andere, mit VHDL-AMS beschreibbare physikalische Domänen abbilden.

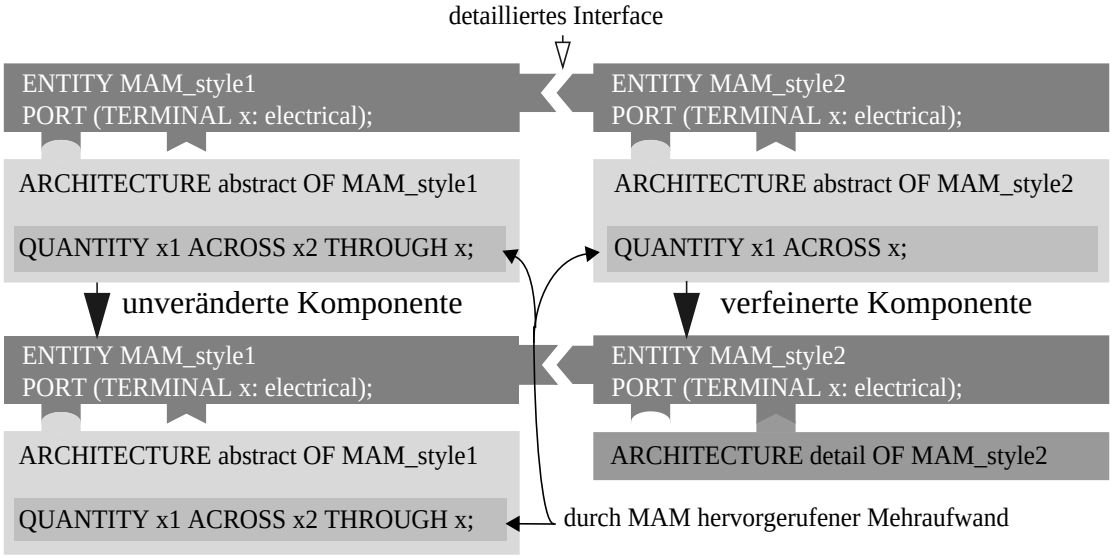


Bild 3-7 Verbindung analoger Modelle mittels MAM

3.2.4 Digital/Analoge Schnittstellen

Im Rahmen einer Variantendiskussion (z. B. Ausführung eines Filters analog, digital oder mechanisch) oder in zeitkritischen digitalen Systemen kann es erforderlich sein, daß Schnittstellen auf hohem Abstraktionsniveau digital, auf niedrigem Niveau aber analog beschrieben werden müssen. Da digitale Schnittstellen Signale benutzen und analoge Schnittstellen Terminals, muß man ein gemeinsames Interface-Objekt finden, das die Eigenschaften beider Schnittstellenobjekte nachbilden kann. In [94] wird ein Verfahren vorgestellt, analoges konservatives Verhalten unter Verwendung von *resolved signals* zu modellieren. Untersuchungen zur Anwendung ereignisdiskreter Beschreibungen zur Simulation analogen Verhaltens [95] sowie eigene Untersuchungen zur Simulation elektrischer Schaltungen mittels Relaxationsalgorithmen im Projekt UNISIM [96] haben jedoch gezeigt, daß insbesondere bei größeren oder nichtlinearen Schaltungen eine Simulation ohne nichtlinearen Gleichungssystemlöser wenig erfolgversprechend ist. Da folglich die Abbildung der Eigenschaften von Terminals auf Signale wenig erfolgreich erscheint, soll hier ein Terminal als gemeinsames Objekt Anwendung finden.

Für die Verbindung von Signalen und Terminals finden spezielle D/A- und A/D-Wandler Anwendung. Diese ermöglichen in den abstrakten Modellen die Abbildung der Zustände einer mehrwertigen Logik als Spannung auf ein Terminal. Eine geeignete Beschreibung der D/A- und A/D-Wandler erlaubt außerdem die Rückgewinnung der *events* auf den digitalen Signalen.

Diese D/A- und A/D-Wandler werden in den abstrakten Modellen instantiiert (Bild 3-8). Allerdings haben Untersuchungen zur Simulationszeit gezeigt, daß diese Vorgehensweise die Simulationszeiten deutlich verlängert. Wenn diejenige Komponente, die von einer digitalen auf eine analoge Beschreibung verfeinert werden soll, keine Verbindung zu einer anderen analogen Kom-

ponente oder einer weiteren gleichermaßen zu verfeinernden Komponente besitzt, so kann hier auch das abstrakte Interface auf hoher Abstraktionsebene verwendet werden. Die D/A- und A/D-Wandler sind dann in die einzelne, zum Netzwerk verfeinerte Komponente einzufügen (Bild 3-9).

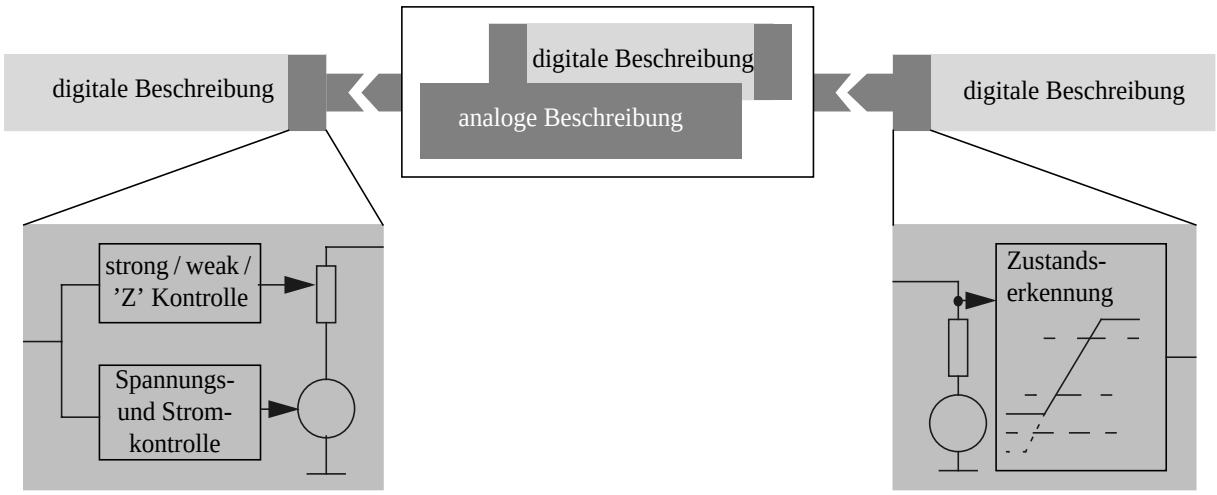


Bild 3-8 Instantiieren der D/A- und A/D-Wandler in den abstrakten Modellen

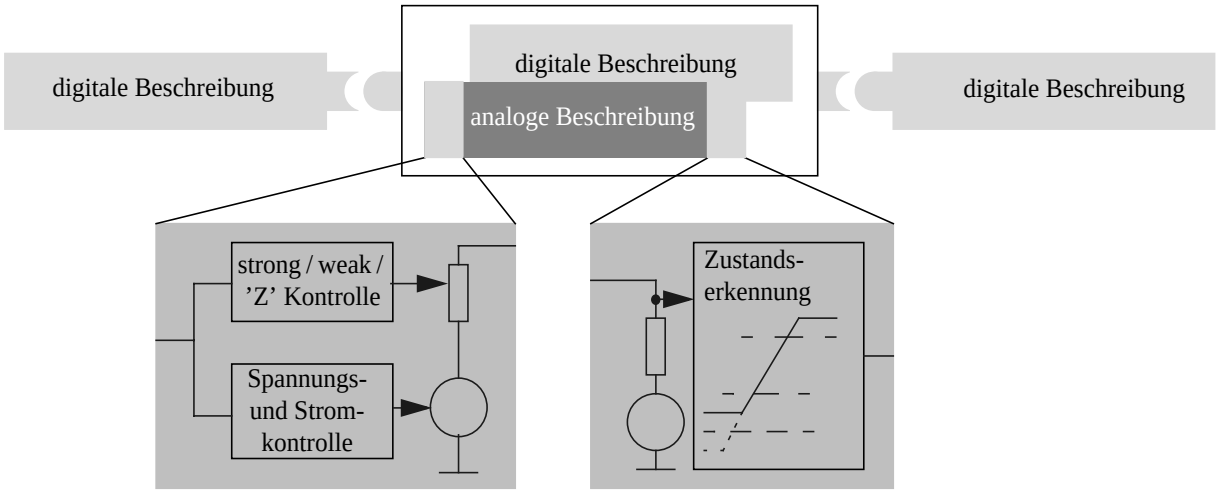


Bild 3-9 Instantiieren der D/A- und A/D-Wandler im verfeinerten Modelle

Verbindung von Signalen mit Terminals. Der 1 Bit D/A-Wandler für die Umsetzung eines Signals vom Typ `std_ulogic` oder `std_logic` auf einen elektrischen Knoten ist in Quelltext 3-2 zu sehen. Die Konstante R_s beschreibt den Ausgangswiderstand einer starken Treiberstufe, R_w den Ausgangswiderstand einer schwachen Treiberstufe und U_1, U_0, U_x die Ausgangsspannungen der jeweiligen Zustände. Diese sind zusammen mit dem A/D- und D/A-Wandler in einem Package `MAM_Support` beschrieben und können auf technologiebezogene oder technologieunabhängige Werte gesetzt werden. Verwendet man technologieunabhängige Werte, so besteht die Möglichkeit, zusammen mit dem entsprechenden A/D-Wandler eine VHDL *resolution function* nachzubilden.

```
ENTITY MAM_S2T IS
   PORT (SIGNAL s_in:IN std_ulogic; TERMINAL t1:electrical);
END;

ARCHITECTURE behav OF MAM_S2T IS
   TERMINAL t2:electrical;
   QUANTITY u_r ACROSS i_r THROUGH t2 TO t1;
   QUANTITY u_out ACROSS i_out THROUGH t2;

BEGIN
   IF s_in='U' OR s_in='-' OR s_in='X' USE
      u_out==Ux; u_r==i_r*Rs;
   ELSIF s_in='1' USE
      u_out==U1; u_r==i_r*Rs;
   ELSIF s_in='0' USE
      u_out==U0; u_r==i_r*Rs;
   ELSIF s_in='W' USE
      u_out==Ux; u_r==i_r*Rw;
   ELSIF s_in='L' USE
      u_out==U0; u_r==i_r*Rw;
   ELSIF s_in='H' USE
      u_out==U1; u_r==i_r*Rw;
   ELSE
      i_out==0.0; u_r==0.0;
   END USE;

   BREAK ON s_in;
END;
```

Quelltext 3-2 Umsetzung eines Signals vom Typ std_ulogic oder std_logic auf einen elektrischen Knoten (s. Bild 3-9)

Die Zustände `'U'`,`'X'`,`'W'` und `'-'` werden als Zustand `'X'` in den Ausgang geschrieben. Eine Rückerkennung im nachfolgenden Eingang ist daher nicht mehr möglich. Dies stellt aber keinen Nachteil dar, da der Zustand `'-'` auf einem Signal nicht vorkommen sollte und – da abgesehen vom Zustand `'U'` – die logischen Funktionen des IEEE Packages `std_logic_1164` `'X'` und `'W'` gleich behandeln.

Für abstrakte Schnittstellen vom Typ `bit` entstand ein ähnlicher Treiber. Soll das abstrakte Modell Schnittstellen vom Typ `integer` oder `real` anstelle von `std_logic` oder `std_logic_vector` besitzen, so kann der hier beschriebene Ansatz mit der Verfahrensweise nach Abschnitt 3.2.1 kombiniert werden. Alternativ besteht die Möglichkeit einer direkten Zuweisung dieser Signale auf das Ausgangs-Terminal mittels der in Quelltext 3-3 beschriebenen Syntax.

```
...
QUANTITY u_out ACROSS i_out THROUGH t1;
...
u_out==signal_real;
BREAK ON signal_real;
--oder
u_out==signal_real'RAMP(tr, tf);      -- Anstiegs- und Abfallzeit tr und tf hinreichend klein wählen
                                      -- oder auf technologiespezifische Werte setzen
```

Quelltext 3-3 Direktes Zuweisen von Signalen des Typs real auf ein Ausgangs-Terminal

Verbindung von Terminals mit Signalen. Das Gegenstück zu dem zuvor beschrieben D/A-Wandler ist der folgende 1 Bit A/D-Wandler zur Verbindung von Terminals mit Signalen vom Typ `std_ulogic`. Dieser besitzt folgende Struktur:

```
ENTITY MAM_T2S IS
   PORT (  TERMINAL t1:electrical;
           SIGNAL s_out: OUT std_ulogic);
END;

ARCHITECTURE behav OF MAM_T2S IS
   TERMINAL t2:electrical;
   QUANTITY u_in ACROSS t1;
   QUANTITY u_r ACROSS i_r THROUGH t1 TO t2;
   QUANTITY u_s ACROSS i_s THROUGH t2;
   SIGNAL s1,s0,sZ:boolean;
BEGIN
   i_r==u_r/Rl;
   u_s==Uz;

   s1<=u_in'ABOVE(Ui1);
   s0<=NOT u_in'ABOVE(Ui0);
   sZ<=NOT u_in'ABOVE(0.9*Uz);
   s_out<= '1' WHEN s1 ELSE
           '0' WHEN s0 AND NOT sZ ELSE
           'Z' WHEN sZ ELSE
           'X';
END;
```

Quelltext 3-4 Verbindung von Terminals mit Signalen vom Typ std_ulogic (s. Bild 3-9)

Für eine Unterscheidung der Zustände `'L'` und `'0'` bzw. `'H'` und `'1'` am Eingang müßte vorausgesetzt werden, daß das Interface nur einen Treiber besitzt. Dies hätte aber zur Folge, daß dieser Ansatz nicht auf Bus-Strukturen angewandt werden könnte. Da jedoch die logischen Funktionen aus dem `std_logic_1164` Package nicht zwischen starken und schwachen Treibern unterscheiden, wurde darauf verzichtet, am Eingang des A/D-Wandlers zwischen starken und schwachen Treiber-Zuständen zu unterscheiden. Damit kann dieser Eingang mit mehreren Ausgängen verbunden sein, und er kann entsprechend einer *resolution function* Konflikte der Treiber auf der Busstruktur auflösen.

Eine Besonderheit beim A/D-Wandler stellt die Spannungsquelle U_z dar. Sind alle Treiber im Zustand `'Z'`, so würde am Eingang des A/D-Wandlers eine Spannung von 0 V anliegen. Der Zustand `'Z'` könnte so nicht von `'0'` unterschieden werden. Das Erkennen des Zustandes `'Z'` ist aber insbesondere bei der Simulation von Bus-Systemen notwendig, um bei der späteren praktischen Implementierung undefinierte Pegel zu vermeiden. Mit Hilfe der Quelle U_z speist der A/D-Wandler über R_l einen kleinen Strom in das Terminal ein. Ist nur ein Treiber nicht im Zustand `'Z'`, so wird der minimale Teststrom überlagert und trägt praktisch nicht zum Verhalten auf dem Terminal bei. Sind dagegen alle Treiber im Zustand `'Z'`, so stellt sich am Eingang des Wandlers U_z ein. Möchte man den Zustand `'Z'` explizit erkennen, so muß U_z entweder über U_{i1} oder unter U_{i0} liegen. Soll dagegen z. B. ein TTL-System betrachtet werden, so kann man U_z auf 4 V einstellen und der A/D-Wandler erkennt eine `'1'`, wenn alle Treiber auf `'Z'` sind. Bei einem 5 V-CMOS-System kann U_z auf 2,5 V gelegt werden und der A/D-Wandler erkennt in diesem Fall ein `'X'`.

Für eine Zuweisung auf abstrakte Schnittstellen vom Typ `bit` wurde ebenfalls ein entsprechender Wandler geschaffen. Besitzt das abstrakte Modell Schnittstellen vom Typ `integer` oder `real`, so kann der hier beschriebene Ansatz mit der Verfahrensweise nach Abschnitt 3.2.1 kombiniert werden. Alternativ bestehen die folgenden Möglichkeiten der Verbindung des Interface-Terminals mit einem Signal:

```
ENTITY MAM_T2Sreal IS
   PORT (TERMINAL in1:electrical);
END ENTITY;

ARCHITECTURE behav OF MAM_T2Sreal IS
   QUANTITY u_in ACROSS in1;
   SIGNAL s_in_r:real:=0.0;
   CONSTANT delta: real:=0.1;
   SIGNAL x1,x2: boolean;
BEGIN
   x1<=u_in'ABOVE(s_in_r+delta);
   x2<=u_in'ABOVE(s_in_r-delta);
   s_in_r<=u_in WHEN x1 OR (NOT x2);
END ARCHITECTURE;
```

Quelltext 3-5 Alternative Verbindung des Interface-Terminals mit einem Signal

Dabei muß allerdings beachtet werden, daß

$$h_{max} \leq \varepsilon / \frac{du_{in}}{dt} \qquad \text{(GL 3-1)}$$

ist, wobei $h_{\max}$ den größten zugelassenen Zeitschritt des *analog solvers* der eingesetzten Software darstellt. Mit Hilfe des Parameters ε kann man die Genauigkeit dieses Konverters beeinflussen. Eine exakte Reproduktion der digitalen *events* des vorhergehenden Ausgangs in diesem Eingang ist allerdings nicht möglich. Abhängig von der erforderlichen Genauigkeit und der maximalen zeitlichen Ableitung der zu wandelnden Quantity können sich sehr kleine Werte für $h_{\max}$ ergeben. Dies führt zu einer extremen Verlangsamung der Simulation, so daß diese Art der Verbindung nur für geringe Genauigkeiten und geringe zeitliche Ableitungen der Eingangs-Quantity empfohlen werden kann. Das Verhältnis aus Simulations-Endzeit und $h_{\max}$, das der vom analogen Simulator minimal zu berechnenden Zeitschritte entspricht, sollte die Größenordnung von 10^5 nicht überschreiten.

Handelt es sich bei der zu beschreibenden abstrakten Komponente um ein getaktetes System, so läßt sich die Konvertierung von einem Terminal auf ein Signal mit Hilfe eines Prozesses erheblich vereinfachen (Quelltext 3-6).

```
PROCESS(clk)
BEGIN
   IF clk'event AND clk'last_value='0' THEN      --L-H Flanke auf clk
      s_in_r<=u_in;
   END IF;
END PROCESS;
```

Quelltext 3-6 Konvertierung vom Terminal auf das Signal mit Hilfe eines getakteten Prozesses

Bild 3-10 zeigt die Anwendung gemischt analog/digitaler Schnittstellen im Rahmen der Multi-Architecture-Modellierung.

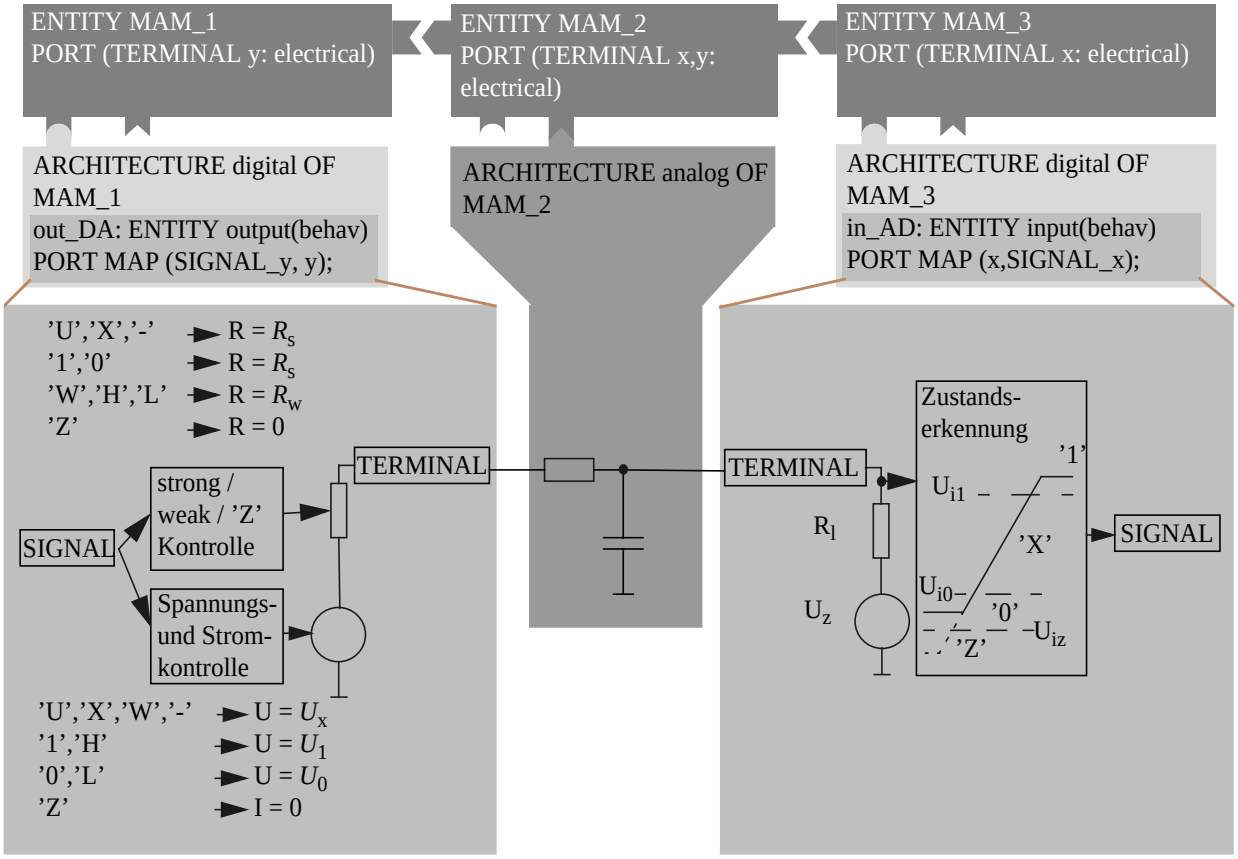


Bild 3-10 Verbindung von analogen mit digitalen Ports

Für die Beschreibung eines technologienahen Verhaltens oder die Nachbildung einer *resolution function* müssen für die Parametern R_s, R_w, R_l, U_x, U_1, U_0, U_z, U_{i1} und U_{i0} der D/A- und A/D-Wandler die folgenden Zusammenhänge gelten:

- technologienahe Beschreibung
  - U_{i1} ist die minimale Eingangsspannung, so daß der Zustand `'1'` erkannt wird, wenn kein Treiber den Zustand `'X'` oder `'0'` hat.
  - U_{i0} ist die maximale Eingangsspannung, so daß der Zustand `'0'` erkannt wird, wenn kein Treiber den Zustand `'X'` oder `'1'` hat. U_{i1} und U_{i0} sind hier durch die Technologie vorgegeben.
  - R_s und R_w sind die Ausgangswiderstände für starke bzw. schwache Treiber und R_l ist der Eingangswiderstand eines Empfängers. R_s, R_w und R_l sind durch die Technologie vorgegeben. Es muß dabei gelten:

$$0 < R_s < R_w \ll R_l \qquad \text{(GL 3-2)}$$

- Für das maximale Verhältnis von Treibern mit den Zuständen `'W'`, `'L'` oder `'H'` zu Treibern mit den Zuständen `'1'` oder `'0'`, bei dem am Eingang die Zustände `'1'` und `'0'` wiedererkannt werden, ergibt sich hier:

$$\frac{N_{w,l,h}}{N_{1,0}} = \frac{R_w}{R_s} \cdot \frac{U_{i1,0} - U_{1,0}}{U_{x,0,1} - U_{i1,0}} \qquad \text{(GL 3-3)}$$

- technologieunabhängige Beschreibung zur Nachbildung einer *resolution function:*
  - Für U_{i1} und U_{i0} gilt:

$$\frac{U_x + U_1 \cdot (n-1)}{n} < U_{i1} < U_1 \text{ und} \qquad \text{(GL 3-4)}$$

$$\frac{U_x + U_0 \cdot (n-1)}{n} > U_{i0} > U_0 \qquad \text{(GL 3-5)}$$

    wobei *n* die maximale Anzahl von Treibern für ein Terminal beschreibt.

  - Für das maximale Verhältnis von Treibern mit den Zuständen `'W'`, `'L'` oder `'H'` zu Treibern mit den Zuständen `'1'` oder `'0'`, bei dem am Eingang die Zustände `'1'` und `'0'` wiedererkannt werden, muß in diesem Fall gelten:

$$0 < R_s < R_w \ll R_l \qquad \text{(GL 3-6)}$$

$$R_w \geq max\left[(n-1) \cdot R_s \cdot \frac{U_0 - U_{i1}}{U_{i1} - U_1}, (n-1) \cdot R_s \cdot \frac{U_1 - U_{i0}}{U_{i0} - U_0}\right] \qquad \text{(GL 3-7)}$$

  - Das maximale Verhältnis von Treibern mit den Zuständen `'W'`, `'L'` oder `'H'` zu Treibern mit den Zuständen `'1'` oder `'0'`, bei dem am Eingang die Zustände `'1'` und `'0'` wiedererkannt werden, bestimmt sich hier aus der zu realisierenden *resolution function.*

3.2.5 Abstrakte analoge Modelle und digitale Implementierungen

Ein Übergang von einer höheren Abstraktionsebene auf eine niedrigere kann auch vorliegen, wenn ein Modell abstrakt als analoger Signalfluß beschrieben wird und man das Modell zu einer digitalen, synthetisierbaren RT-Implementierung verfeinern will. Dabei besteht grundsätzlich die Möglichkeit, Quantities oder Signale als gemeinsame Schnittstelle zu verwenden. Für die Kommunikation zwischen digitalen RT-Modellen kommt oft ein *Handshake* zwischen diesen Komponenten zum Einsatz. Für dieses *Handshake* sind auf hoher Abstraktionsebene entsprechende Leitungen vorzusehen. Bild 3-11 zeigt eine mögliche Verfeinerung von abstrakten analogen Modellen zu digitalen RT-Modellen unter Beibehaltung der Interface-Quantities. In Bild 3-12 ist derselbe Vorgang unter Verwendung von Signalen auf hoher Abstraktionsebene dargestellt. Für das *Handshake* ist ein Enable-Signal vorgesehen, das dem Empfänger anzeigt, wann ein Datenwort gültig ist. Wie zu erkennen ist, kann stets dasselbe Interface verwendet werden, unabhängig davon ob für eine Komponente das abstrakte Signalflußmodell oder das digitale RT-Modell zum Einsatz kommt.

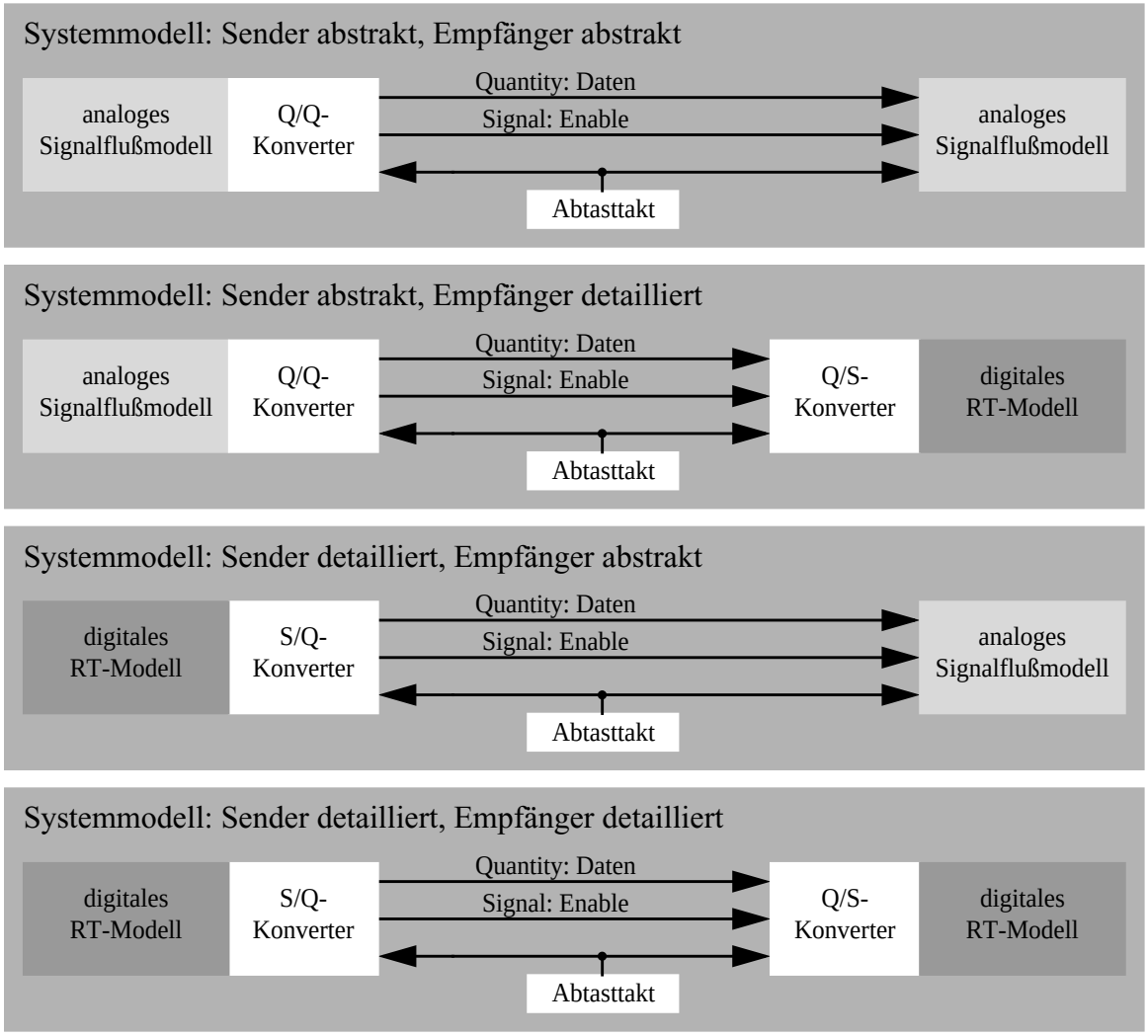


Bild 3-11 Beibehalten der Interface-Quantity bei der Entwicklung digitaler RT-Modelle aus analogen Signalflußmodellen

Für die Anpassung der Schnittstellen wurden die folgenden Konverter implementiert:

- Quantity/Quantity-Konverter (Q/Q-Konverter):
  Der Q/Q-Konverter übergibt die Ausgangs-Quantity des analogen Signalflußmodells unverändert an das für die MAM notwendige Interface. Aus dem Abtasttakt erzeugt der Q/Q-Konverter ein Enable-Signal für potentiell angeschlossene RT-Modelle.
- Quantity/Signal-Konverter (Q/S-Konverter):
  Der Q/S-Konverter wandelt die Eingangs-Quantity des für die MAM notwendigen Interfaces in ein digitales Signal. Dieses Signal benutzt den Datentyp `std_logic_vector`. Das so entstehende digitale Datenwort wird als Festkommazahl im Zweierkomplement interpretiert. Bitbreite und Kommaposition sind per Generic definierbar. Die Abtastung der Quantity erfolgt mit jeder positiven Flanke des Enable-Signals.

- Signal/Quantity-Konverter (S/Q-Konverter):
  Der S/Q-Konverter wandelt das Ausgangs-Signal des RT-Modells in eine für das MAM-Interface notwendige Quantity. Das digitale Datenwort wird wie beim Q/S-Konverter als Festkommazahl im Zweierkomplement interpretiert. Da die hier verwendeten digitalen Signale zeit- und wertdiskret sind, muß der Werteverlauf der Quantity zwischen den Signaländerungen mittels des `'RAMP`-Attributes linear approximiert werden. Anstiegs- und Abfallzeiten sind dabei auf $t_r = t_f = 1/f_{clk}$ zu setzen (f_{clk} ist die Frequenz des Abtasttaktes). Dies gewährleistet einen stetigen Werteverlauf der Quantity und damit eine sichere erste Ableitung dieser nach der Zeit. Der zeitliche Verlauf dieser Ableitung hat die Form einer Stufenfunktion. Eine weitere Ableitung dieser Stufenfunktion erzeugt eine Reihe steiler Impulse. Diese Ableitungen zweiter Ordnungen verursachen noch keine numerische Instabilität, sie sind aber nicht mehr direkt interpretierbar. Eine weitere Ableitung dieser steilen Impulse kann zu numerischen Instabilitäten führen. Ableitungen höherer Ordnung der Eingangsquantity sollten daher im nachfolgenden Modell nicht vorkommen. Verhalten, das die Eingangsquantity integriert, verursacht keine Probleme. Entsprechend ist bei der Verwendung von Laplace-Übertragungsfunktionen darauf zu achten, daß das Zählerpolynom maximal eine Ordnung größer sein darf als das Nennerpolynom.

Die Konverter sind Bestandteil des Packages „MAM_Support“. Ihr Quellcode ist im Anhang B beschrieben.

Die Vorgehensweise nach Bild 3-11 erlaubt auf hoher Abstraktionsebene die Durchführung einer AC-Simulation. Es ergeben sich allerdings auch einige Nachteile:

- Besteht das Systemmodell nur aus abstrakten analogen Modellen, so werden die Systemgleichungen in **einem** Gleichungssystem gelöst. Sobald jedoch ein digitales Modell eingefügt wird, ergibt sich ein Simulationsschrittversatz bei der Berechnung. Dieser kann das Simulationsergebnis insbesondere bei rückgekoppelten Modellen verändern. Auf diese Problematik wird in Abschnitt 5.2.3.2, S. 99 im Zusammenhang mit den MOC von SystemC-AMS noch näher eingegangen.
- Das Verifizieren der digitale Kommunikation der RT-Modelle ist nur bedingt möglich, da durch die Wandlung in eine Quantity und zurück eine Verzögerung des Signals von einer Taktperiode hervorgerufen wird.
- Die Wandler in den RT-Modellen behindern die Synthese einer Gatternetzliste aus der RT-Beschreibung.

Im Gegensatz dazu wird in Bild 3-12 bereits auf hoher Abstraktionsebene – in Form von Signalen – das Interface benutzt, das auf RT-Ebene notwendig ist. Dafür wurden die folgenden Konverter entwickelt:

- Quantity/Signal-Konverter (Q/S-Konverter):
  Diese Variante des Q/S-Konverters wandelt die Ausgangs-Quantity des analogen Modells in ein für das MAM-Interface notwendiges digitales Signal. Dieses Signal benutzt ebenfalls den Datentyp `std_logic_vector`. Das so entstehende digitale Datenwort wird als Festkommazahl im Zweierkomplement interpretiert. Bitbreite und Kommaposition sind per

Generic definierbar. Aus dem Taktsignal wird ein Enable-Signal gewonnen. Die Abtastung der Quantity erfolgt mit jeder positiven Flanke dieses Enable-Signals.

- Signal/Quantity-Konverter (S/Q-Konverter):
  Der S/Q-Konverter ist identisch mit dem Konverter nach Bild 3-11.

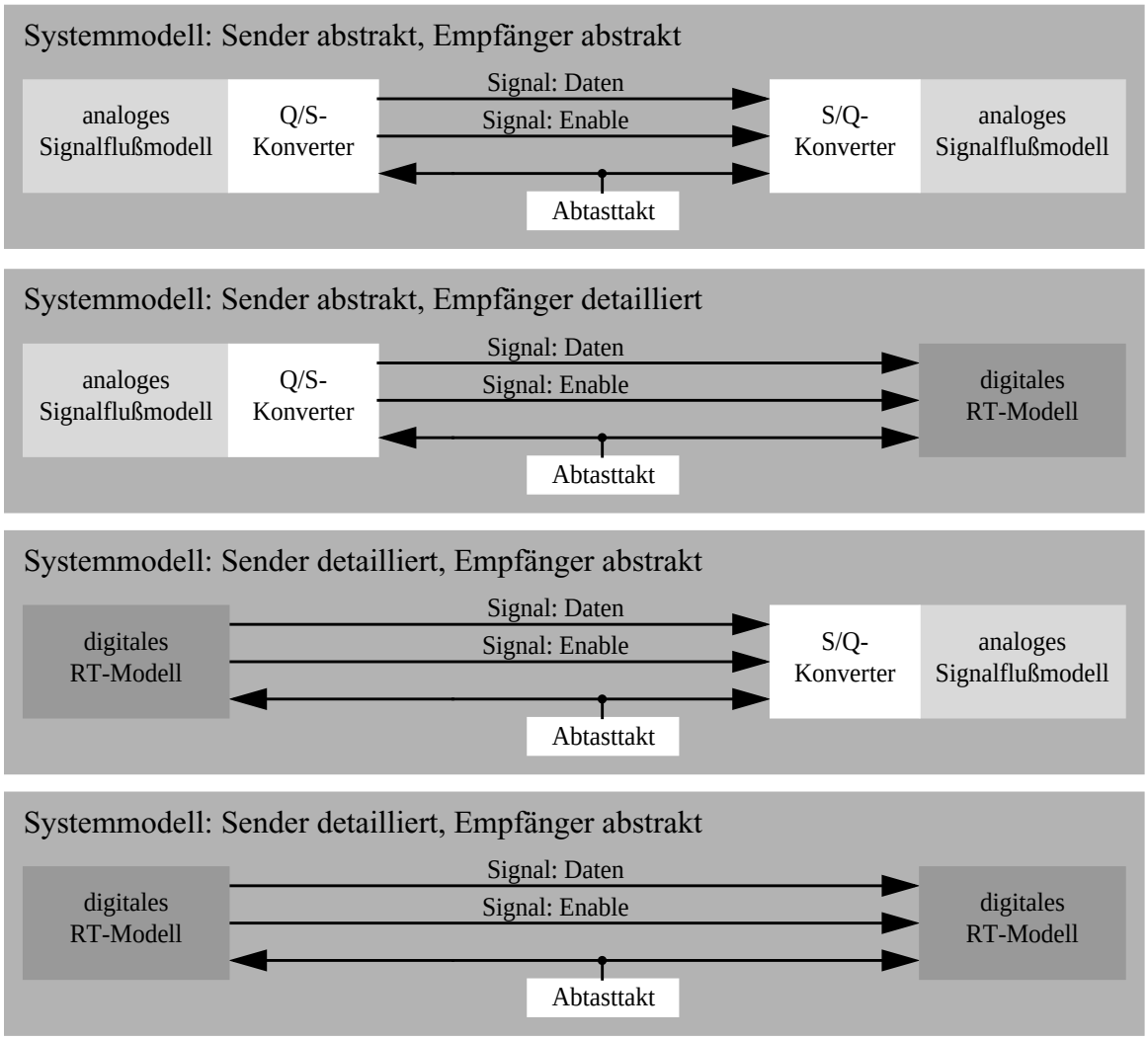


Bild 3-12 Verwendung von Signalen auf hoher Abstraktionsebene bei der Entwicklung digitaler RT-Modelle aus analogen Signalflußmodellen

Dieser Ansatz hat den Nachteil, daß auf hoher Abstraktionsebene keine AC-Simulation der analogen Komponenten mehr durchgeführt werden kann. Dieser Nachteil ist insofern zu relativieren, da nach dem Austausch des ersten analogen Komponentenmodells gegen eine digitale Modellbeschreibung die Möglichkeit zur AC-Simulation ohnehin entfällt.

Die Nachteile des Ansatzes aus Bild 3-11 entfallen dagegen:

- Durch die Verwendung von Signalen zur Datenübertragung entsteht der Schrittversatz bereits bei der Simulation einer Konfiguration aus rein analogen Modellen. Dadurch ändert sich das prinzipielle Systemverhalten beim Austausch der analogen Modelle gegen digitale Modelle nicht mehr.

- Da die digitalen Modelle ohne Konverter miteinander kommunizieren, tritt eine Signalverzögerung bei der digitalen Kommunikation nicht mehr auf. Die Synthese wird ebenfalls nicht mehr behindert.

Wenn auf hoher Abstraktionsebene keine AC-Simulation durchgeführt werden soll, so ist der Ansatz nach Bild 3-12 – die Verwendung von Signalen auf hoher Abstraktionsebene anstelle von Quantities – zu bevorzugen. Bei diesem Verfahren wird, wie in den Ansätzen nach Abschnitt 3.2.1 bis Abschnitt 3.2.4, die auf niedriger Abstraktionsebene notwendige Schnittstelle bereits auf hoher Abstraktionsebene benutzt.

Will man dagegen auf hoher Abstraktionsebene eine AC-Simulation durchführen, so ist das Verfahren nach Bild 3-11 – die Beibehaltung der Quantities der hohen Abstraktionsebene auch auf niedrigeren Abstraktionsebenen – anzuwenden und die Nachteile dieses Verfahrens gegen die Vorteile der MAM abzuwägen. Alternativ kann man auch den Ansatz nach Bild 3-12 einsetzen und die AC-Simulation mit einem separaten Modell ohne MAM-Schnittstellen durchführen. Der Mehraufwand dafür hält sich in Grenzen. Es ist aber darauf zu achten, daß keine Inkonsistenzen zwischen den Modellen für AC-Simulation und den Modellen für den Systementwurf mit MAM entstehen.

Ein Lösung, die die Vorteile beider Verfahren ohne deren Nachteile auf sich vereint, konnte im Rahmen dieser Arbeit für VHDL-AMS nicht gefunden werden, da weder Sprachsyntax noch Sprachsemantik geeignete Mittel bereitstellt.

3.2.6 Steuersignale und Versorgungsspannungen

Durch die Verringerung des Abstraktionsgrades ändern sich nicht nur die Art oder der Datentyp einer Schnittstelle, es kommen in der Regel auch neue Anschlüsse, z. B. für Steuersignale oder Versorgungsspannungen hinzu. Wenn im Rahmen des in Abschnitt 3.4 erläuterten Systementwurfs mit MAM die notwendige Schnittstelle auf niedrigem Abstraktionsniveau ermittelt wird, so erhält man in der Regel auch Informationen für zusätzlich notwendige Anschlüsse auf niedrigem Abstraktionsniveau, so daß diese Anschlüsse bereits auf hoher Abstraktion vorgesehen werden können. Dabei sind im zu instantiierenden abstrakten Modell diese Anschlüsse wie folgt zu beschalten, um Auswirkungen auf die Simulationszeit zu vermeiden bzw. zu minimieren:

- Parametereingänge (Generics), Signal- und Quantity-Eingänge sind offen zu lassen.
- Signal- und Quantity-Ausgänge müssen mit einem konstanten, möglichst sinnvollen Wert (z. B. 0 V für Quantities und `'L'` oder `'H'` für Signale) getrieben werden.
- Für Terminals sollte man keine Branch-Quantity anlegen. Führt dies zu Fehlermeldungen im Simulator, so sind die Terminals mit einem Widerstand gegen den Referenz-Knoten der jeweiligen Nature zu beschalten.

Wurde auf hoher Abstraktionsebene ein Anschluß nicht berücksichtigt, so muß wie beim konventionellen Entwurf dieser in das Systemmodell eingefügt werden, sobald er benötigt wird. Entfernt man das verfeinerte Modell aus dem Systemmodell und setzt statt dessen das abstrakte ein, so ist das abstrakte Modell mit diesen Anschlüssen nachzurüsten, anstatt die Anschlüsse aus dem Systemmodell wieder zu entfernen.

3.3 Package MAM_Support

Zur komfortablen Anwendung der MAM im Entwurf entstand ein VHDL-AMS-Package, das die im Rahmen dieser Arbeit entstandenen Abstraktionswandler, deren Parameter sowie weitere Hilfsfunktionen enthält. Dabei handelt es sich insbesondere um:

- einfachen Funktionen zur Konvertierung zwischen Datentypen

```
FUNCTION bit_vector2int(in1: bit_vector) RETURN integer;
FUNCTION int2bit_vector(in1: in integer; size: in integer:=8) RETURN bit_vector;
FUNCTION bit2real(in1:bit; low_level,high_level:real) RETURN real;
FUNCTION real2bit(in1:real; threshold:real) RETURN bit;
```

- Funktionen zur Konvertierung zwischen den Datentypen `std_logic_vector` (Festkommarepräsentation) und `real`

```
FUNCTION fix2real( CONSTANT data_in:std_logic_vector; CONSTANT sig_wide:positive;
                   CONSTANT fix_point:natural) RETURN real;
FUNCTION real2fix( CONSTANT data_in:real; CONSTANT sig_wide:positive;
                   CONSTANT fix_point:natural) RETURN std_logic_vector;
```

- A/D- und D/A-Wandler nach Abschnitt 3.2.4

```
COMPONENT MAM_Sstdulogic2T    -- Treiber Signal(std_ulogic)->Terminal(electrical)
COMPONENT MAM_Sbit2T          -- Treiber Signal(bit)->Terminal(electrical)
COMPONENT MAM_Sreal2T         -- Treiber Signal(real)->Terminal(electrical)
COMPONENT MAM_T2Sstdulogic    -- Treiber Terminal(electrical)->Signal(std_ulogic)
COMPONENT MAM_T2Sbit          -- Treiber Terminal(electrical)->Signal(bit)
COMPONENT MAM_T2Sreal         -- Treiber Terminal(electrical)->Signal(real)
COMPONENT MAM_T2Sreal_clk     -- Treiber Terminal(electrical)->Signal(real) für getaktete Systeme
```

- Parameter für die A/D- und D/A-Wandler nach Abschnitt 3.2.4
- Abstraktionswandler nach Abschnitt 3.2.5

```
COMPONENT MAM_s2q             -- Treiber Signal(real)->Quantity
COMPONENT MAM_q2s             -- Treiber Quantity->Signal(real)
COMPONENT MAM_q_clk2s         -- Treiber Quantity->Signal(real), Erzeugung eines Enable-Signals
                              -- aus Abtasttakt
COMPONENT MAM_q_clk2q         -- Treiber Quantity->Quantity, Erzeugung eines Enable-Signals
                              -- aus Abtasttakt
```

Die Funktionen und Modelle des VHDL-AMS Packages „MAM-Support“ wurden mit dem Simulator „AdvanceMS“ von Mentor Graphics in der Version v2.1_1.1 kompiliert und erfolgreich getestet. Die Abstraktionswandler-Prozeduren nach Abschnitt 3.2.2 für die RS232-Schnittstelle und den I^2C-Bus sind in separaten Packages definiert. Diese beiden Packages lassen sich mit der vorliegenden Simulatorversion nicht kompilieren, da die Version v2.1_1.1 verschiedene verwendete Statements noch nicht unterstützt. Die Funktionalität der Packages wurde statt dessen mit dem VHDL-Simulator „Modelsim“ in der Version SE Plus 5.7c verifiziert.

Das Package „MAM-Support“ sowie der dazu gehörende Package Body sind im Anhang B abgebildet. Der Quellcode des Packages ist frei verfügbar (*open source*), so daß der Anwender das Pakkage jederzeit um weitere, für einen bestimmten Einsatzfall benötigte Elemente erweitern kann.

3.4 Entwurfsablauf mit MAM

Im Vergleich zu einem herkömmlichen Top-Down-Entwurf muß bei Anwendung der MAM die auf der niedrigsten zu berücksichtigenden Abstraktionsebene notwendige Schnittstelle mit Beginn der Modellierung der abstrakten Komponenten bekannt sein. Deshalb ist der Entwurfsablauf mit MAM gegenüber dem Top-Down-Entwurf entsprechend Bild 3-13 anzupassen.

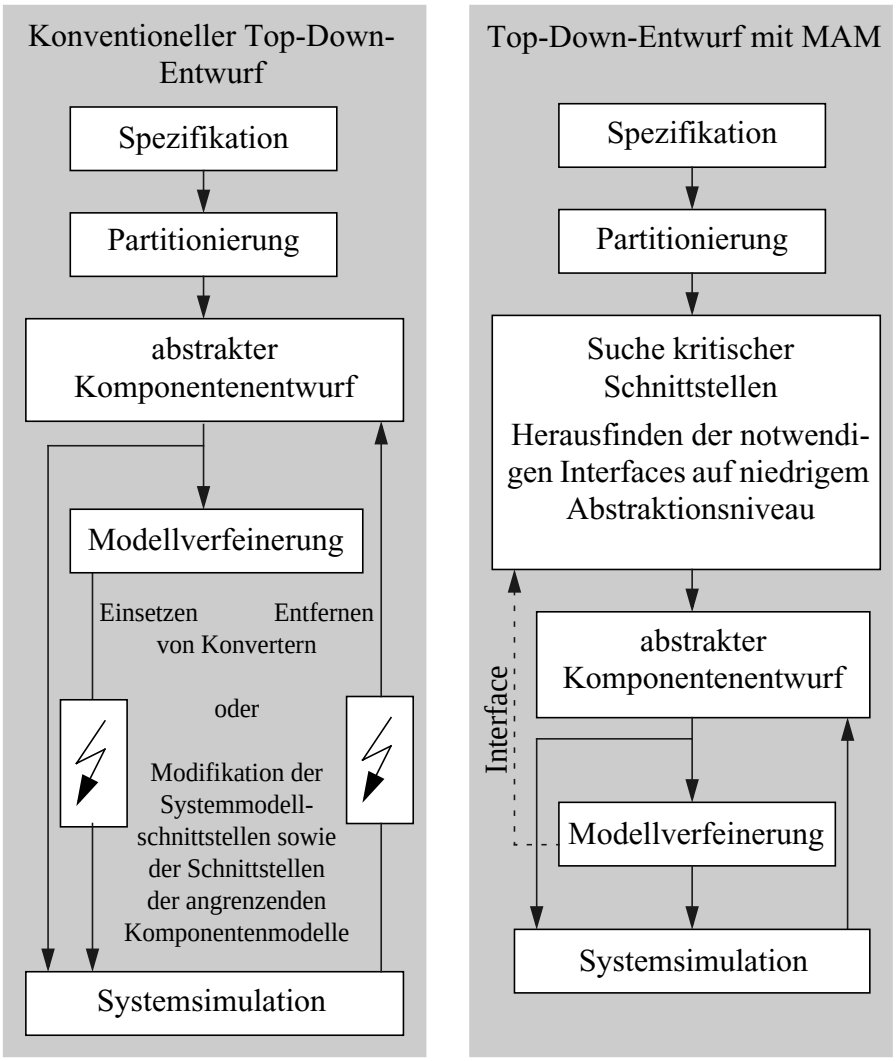


Bild 3-13 Top-Down-Entwurf ohne und mit MAM

Eine Schnittstelle einer Komponente gilt Bild 3-13 zufolge als „kritisch“, wenn für die Komponente eines oder mehrere der folgenden Kriterien gelten:

- Eine digitale Komponente wird zunächst algorithmisch beschrieben, soll später aber durch ein synthetisierbares oder synthetisiertes Modell ersetzt werden bzw. die Komponente steht in Verbindung mit einer digitalen Komponente auf niedrigem Abstraktionsniveau.
- Eine analoge Komponente wird zunächst als funktioneller Block beschrieben, soll aber später gegen eine Netzwerkbeschreibung ausgetauscht werden bzw. die Komponente steht in Verbindung mit einem Netzwerk.
- Die Realisierungsart einer Komponente unterliegt einer Variantendiskussion.

- Das exakte zeitliche Verhalten der Komponente oder die Kommunikation zwischen zwei Komponenten ist bedeutsam und kann nicht z. B. per *Backannotation* in das abstrakte Modell implementiert werden.
- Für die Verbindung von Komponenten erscheint ein Leitungsmodell notwendig.
- Eine Schnittstelle ist ebenfalls kritisch, wenn sie mit einer der zuvor genannten Komponenten in Verbindung steht.

Bei der Evaluierung der MAM haben sich einige Modellierungsregeln ergeben, deren Einhaltung für einen effizienten Entwurf empfohlen wird:

- Das Abbilden einer digitalen Schnittstelle auf Terminals sollte man nur für kritische Schnittstellen durchführen. Da für jedes Signal ein Terminal angelegt werden und für jede `THROUGH QUANTITY` eines Terminals eine Gleichung vorhanden sein muß, ergibt sich bei einer Vielzahl so abgebildeter Signale ein großes vom analogen Simulator zu lösendes Gleichungssystem, was die Simulation verlangsamt.
- Schnittstellen, die man sowohl im abstrakten als auch im detaillierten Modell mit Quantities beschreiben kann, sind keine kritischen Schnittstellen und sollten ebenfalls nicht als Terminal beschrieben werden. Informationen auf Quantities kann ein Simulator u. U. von Komponente zu Komponente ohne Rückwirkung propagieren, während für ein Terminal stets ein Eintrag im Gleichungssystem zu lösen ist. Daraus kann je nach Modellstruktur und verwendetem Simulator ein Verlust an Simulationsgeschwindigkeit und -stabilität resultieren.
- Das Abbilden einer Quantity erfolgt üblicherweise auf den `ACROSS`-Zweig eines Terminals. Dies bedeutet, daß der Eingangswiderstand des nächsten Modells größer als Null sein muß. Ist der Eingangswiderstand gleich Null, so muß die Quantity im sendenden Modell den `THROUGH`-Zweig des Terminals benutzen.
- Werden abstrakte und detaillierte Modelle miteinander verbunden, so ist zu beachten, daß die abstrakten Modelle auf den Terminals ein ideales Verhalten zeigen und bezüglich Rückwirkungen aus der detaillierten Komponente unempfindlich sind.
- Weiterhin ist zu beachten, daß in abstrakten Komponenten Informationen aus vorhergehenden detaillierten Komponenten verloren gehen können (z. B. digitale Zustände wie `'U'`, `'X'`)

Werden diese Modellierungsregeln beachtet, ist nur mit einer geringfügigen bis moderaten Verringerung der Simulationsgeschwindigkeit zu rechnen, wie noch in Abschnitt 4, S. 69 ff. gezeigt werden wird. Dafür erspart man sich beim Entwurf die zeitaufwendige und fehleranfällige Konvertierung der Schnittstellen beim Einfügen von Modellen oder Architekturen auf unterschiedlichen Abstraktionsebenen. Dies ist insbesondere dann von Vorteil, wenn z. B. für eine oder mehrere Komponenten Verhaltensmodelle und *Reduced Order Modelle* [36], [46] zur Verfügung stehen. Während der Simulation kann so für jede Komponente, für die unterschiedlich abstrakte Architekturen existieren, entweder eine abstrakte oder eine detaillierte Version eingesetzt werden, je nach Erfordernissen bezüglich Simulationsgeschwindigkeit und Genauigkeit.

Beim Entwurf heterogener Systeme wird in der Regel ein Top-Down-Entwurf, wie beim Entwurf digitaler Systeme, angestrebt. Durch technologische und physikalische Grenzen ergeben sich

dabei aber oft Probleme hinsichtlich der Realisierbarkeit der entworfenen Schaltungen und Strukturen. Daher kommen beim Entwurf solcher Systeme nach wie vor auch Bottom-Up- bzw. Meet-In-The-Middle-Strategien [21] zum Einsatz. Wie Bild 3-14 zeigt, ist eine Anwendung der MAM auch im Rahmen solcher Entwurfsabläufe möglich. Beim Bottom-Up-Entwurf mit MAM wird nur das Verhalten der Modelle abstrahiert, aber nicht deren Schnittstellen. Beim Meet-In-The-Middle-Ansatz können die im Rahmen der Bottom-Up-Schritte beim Komponentenentwurf definierten Schnittstellen, ähnlich wie in Bild 3-13, bei den Top-Down-Schritten des Systementwurfes einfließen. Für die Anwendung der MAM beim Bottom-Up- oder Meet-In-The-Middle-Entwurf gelten die gleichen Entwurfsregeln wie beim Top-Down-Entwurf mit MAM.

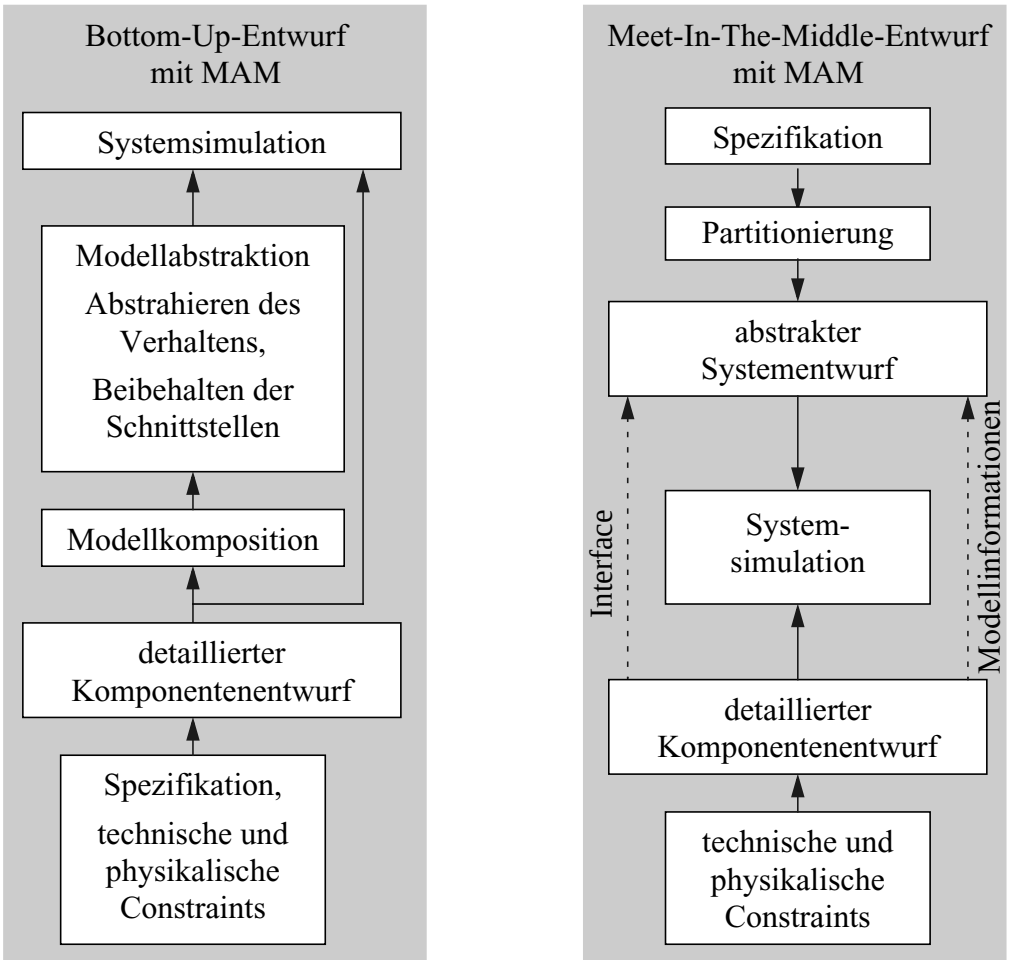


Bild 3-14 Anwendung der MAM bei Bottom-Up- bzw. Meet-In-The-Middle-Entwurfsstrategie

Die MAM sollte man bei rein digitalen Systemen oder Systemen, bei denen Modelle oder Schnittstellen automatisch generiert werden, nicht anwenden, da bei solchen Systemen keine Steigerung der Entwurfseffizienz durch die MAM zu erwarten ist. Besteht jedoch nicht die Möglichkeit, Modelle durch Synthese zu erzeugen, wie dies bei heterogenen Systemen in der Regel der Fall ist, so ist der Einsatz der MAM sinnvoll. Da solche Entwürfe durch ein hohes Maß an manuellen Arbeiten geprägt sind, wurde für die MAM auf die Realisierung von Werkzeugen zur Automatisierung verzichtet. Weitere Aspekte dazu werden auch im Ausblick in Abschnitt 8, S. 127 erörtert.

4 Anwendung der Multi-Architecture-Modellierung

4.1 Vergleich von Simulationszeiten mit und ohne MAM

Mit der Verringerung der Abstraktion der Schnittstellen ist in der Regel eine Verringerung der Simulationsgeschwindigkeit verbunden. Beim Entwickeln der MAM wurde große Sorgfalt darauf gelegt, diese Verlangsamung der Simulation so gering wie möglich zu halten. Die nachfolgenden Untersuchungen sollen anhand einfacher Beispiele zeigen, wie groß der Einfluß der verfeinerten Schnittstellen im Rahmen der MAM tatsächlich ist.

4.1.1 Konservative Schnittstellen statt nichtkonservativer Schnittstellen

Die einfachste Anwendung der MAM bei analogen elektrischen und nichtelektrischen Systemen ist die Verwendung von Terminals anstelle von Quantities. Zur Durchführung des Simulationszeitvergleichs sei folgende Struktur gegeben:

Bild 4-1 Testmodell „Terminal statt Quantity“

Auf Systemebene soll die Quelle aus einer Sinus-Spannungsquelle bestehen, die Übertragungsfunktion ist ein Bandpaß zweiter Ordnung und die Senke möge einen unendlich hohen Eingangswiderstand und keine weitere Funktionalität besitzen.

Betrachtet man nun diese Komponenten als Bestandteil eines komplexeren Systems und möchte man z. B. den Einfluß des Quellenwiderstandes auf das Verhalten der realen Implementierung einer Übertragungsfunktion berücksichtigen, so müssen ohne MAM alle drei Komponentenmodelle und das Systemmodel entsprechend geändert werden. Mit MAM sind, wie in Abschnitt 3, S. 43 beschrieben, nur die Modelle zu modifizieren, in denen es Veränderungen des Verhaltens gibt. Der Vergleich in Quelltext 4-1 zeigt eine mögliche Realisierung dieses Verhaltens ohne und mit MAM.

Wie man sieht, ist der Mehraufwand bei der Modellierung sehr begrenzt. Den Einfluß auf die Simulationszeit zeigt Tabelle 4-1. Die Zeiten wurden auf einer Sun UltraSparc-III mit 900 MHz unter „Sun Solaris 8“ ermittelt. Als Simulator kam dabei „AdvanceMS“ in der Version 2.1 der Firma Mentor Graphics zum Einsatz. Es wurde auch eine Simulation auf einem PC mit dem Simulator „Simplorer“ (keine AC-Simulation) der Firma Ansoft durchgeführt. Da „Simplorer“ aber die absolute Rechenzeit, „AdvanceMS“ dagegen die CPU-Zeit angibt, werden hier nur die CPU-Zeiten der Simulation mit „AdvanceMS“ angegeben, da diese besser reproduzierbar sind. Es wurden jeweils drei Simulationsläufe für eine Transient- und eine AC-Simulation durchgeführt.

konventionelle Beschreibung:

```
… --Library und Use Statements
ENTITY t4q_src IS
   PORT( QUANTITY q_out:OUT real);
END;
ARCHITECTURE a OF t4q_src IS
   QUANTITY q_ac: real SPECTRUM 1.0,0.0;

BEGIN
   q_out== sin(100.0e3*math_2_pi*now)
           + q_ac; --1V, 100kHz
END;


ENTITY t4q_trans IS
   PORT( QUANTITY q_in:IN real;
         QUANTITY q_out:OUT real);
END;
ARCHITECTURE a OF t4q_trans IS
   CONSTANT a:real:=1.0;
   CONSTANT rc:real:=1.0e-6;
   --Zähler- und Nennerpolynom
   CONSTANT num:real_vector:=(0.0,
              rc*(1.0+a));
   CONSTANT den:real_vector:=(2.0,
              rc*(3.0-a), rc**2);


BEGIN
   q_out==q_in'LTF(num,den);
END;


ENTITY t4q_dest IS
   PORT( QUANTITY q_in:IN real);
END;
ARCHITECTURE a OF t4q_dest IS

BEGIN
END;


ENTITY t4q IS
END;
ARCHITECTURE a OF t4q IS
   QUANTITY q1,q2:real;
   … --Components
BEGIN
   src:      t4q_src    PORT MAP(q1);
   trans:    t4q_trans  PORT MAP(q1,q2);
   dest:     t4q_dest   PORT MAP(q2);
END;
```

Beschreibung mit MAM:

```
…  --Library und Use Statements
ENTITY MAMt4q_src IS
   PORT( TERMINAL t_out:electrical);
END;
ARCHITECTURE a OF MAMt4q_src IS
   QUANTITY q_ac: real SPECTRUM 1.0,0.0;
   QUANTITY u ACROSS i THROUGH t_out;
BEGIN
   u==  sin(100.0e3*math_2_pi*now)
        + q_ac; --1V, 100kHz
END;

… --Library und Use Statements
ENTITY MAMt4q_trans IS
   PORT( TERMINAL t_in:electrical;
         TERMINAL t_out:electrical);
END;
ARCHITECTURE a OF MAMt4q_trans IS
   CONSTANT a:real:=1.0;
   CONSTANT rc:real:=1.0e-6;
   --Zähler- und Nennerpolynom
   CONSTANT num:real_vector:=(0.0,
              rc*(1.0+a));
   CONSTANT den:real_vector:=(2.0,
              rc*(3.0-a), rc**2);
   QUANTITY u1 ACROSS t_in;
   QUANTITY u2 ACROSS i2 THROUGH t_out;
BEGIN
   u2==u1'LTF(num,den);
END;

… --Library und Use Statements
ENTITY MAMt4q_dest IS
   PORT( TERMINAL t_in :electrical);
END;
ARCHITECTURE a OF MAMt4q_dest IS
   QUANTITY u1 ACROSS t_in;
BEGIN
END;

…--Library und Use Statements
ENTITY MAMt4q IS
END;
ARCHITECTURE a OF MAMt4q IS
   TERMINAL t1,t2:electrical;
   … --Components
BEGIN
   src:    MAMt4q_src   PORT MAP(t1);
   trans:  MAMt4q_trans PORT MAP(t1,t2);
   dest:   MAMt4q_dest  PORT MAP(t2);
END;
```

Quelltext 4-1 Vergleich einer möglichen Realisierung von Komponentenmodellen ohne und mit MAM

Um den Einfluß der Schnittstelle auf die Simulationszeit besser beurteilen zu können, wurde zusätzlich die Simulation für ein Testsystem mit einer Konfiguration aus 100 in Serie geschalteten Modellen mit einer Übertragungsfunktion (Bild 4-2) sowie mit einer Konfiguration aus 100 Übertragungsfunktionen in einem Modell (Bild 4-3) durchgeführt.

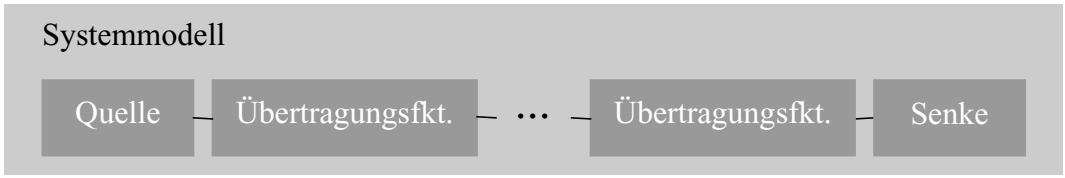


Bild 4-2 Konfiguration aus 100 in Serie geschalteten Modellen

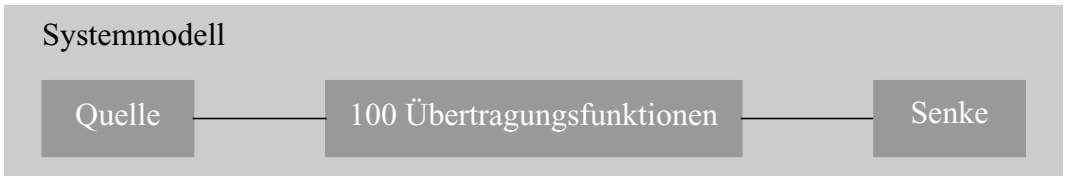


Bild 4-3 Konfiguration aus 100 Übertragungsfunktionen in einem Modell

	Mittelwert von drei Simulationsläufen ohne MAM	Mittelwert von drei Simulationsläufen mit MAM
Ausgangsmodell, Transient-Simulation	12,98 s	13,10 s
Ausgangsmodell, AC-Simulation	163 ms	173 ms
100 Modelle mit einer Übertragungsfunktion, Transient-Simulation	2 min 55 s	2 min 49 s
100 Modelle mit einer Übertragungsfunktion, AC-Simulation	3,22 s	4,07 s
1 Modell mit 100 Übertragungsfunktionen, Transient -Simulation	41,45 s	40,39 s
1 Modell mit 100 Übertragungsfunktionen, AC-Simulation	1,60 s	1,38 s

Tabelle 4-1 Simulationszeiten „Terminal statt Quantity“

Die Simulationsparameter waren:

- AC-Simulation:
  - Startfrequenz: 1 Hz
  - Endfrequenz: 1 MHz
  - Frequenzschritte: 1000 pro Dekade

- Transient-Simulation
    - Simulations-Zeit: 100 µs
    - minimale Schrittweite: 10^{-12} s
    - maximale Schrittweite: 10^{-9} s

Es ist zu erkennen, daß es keinen signifikanten Unterschied in den Simulationszeiten zwischen Modell ohne MAM und mit MAM gibt. Es gibt sogar Konstellationen, in denen die Modellierung mit MAM schneller als ohne MAM ist. Die Ursachen dafür sind in der Implementierung des Simulators zu suchen. Da der Programmcode des kommerziellen Simulators nicht verfügbar ist, können keine weiteren Aussagen diesbezüglich getroffen werden.

Der deutliche Unterschied in der Simulationszeit zwischen der Konfiguration mit 100 Modellen mit einer Übertragungsfunktion und der Konfiguration mit einem Modell mit 100 Übertragungsfunktionen ist von der MAM unabhängig und deutet auf eine interne Optimierung der Modelle seitens des Simulators „AdvanceMS“ hin.

Die Simulation mit „Simplorer“ führt bezüglich Rechenzeit mit MAM und ohne MAM dagegen zu einer Verlängerung der Simulationszeit um ca. 25 %. Die Simulation der Konfiguration aus 100 Modellen wurde nach ca. einer Stunde abgebrochen, da kein Simulationsfortschritt erkennbar war und von einem Fehler im Simulator des „Simplorer“ ausgegangen werden mußte. Dieses Verhalten war sowohl mit als auch ohne MAM reproduzierbar.

Beim Kompilieren und in der Elaborationsphase konnte in keinem Fall ein nennenswerter Unterschied bei der benötigten Rechenzeit beobachtet werden.

In einem weiteren Modell wurde untersucht, ob es einen Einfluß auf die Simulationszeit gibt, wenn die Zuweisung auf die Ausgangs-Quantity unidirektional erfolgt. Da das *simultaneous procedural* Statement mit der zur Verfügung stehenden Version von „AdvanceMS“ nicht funktioniert, wurde auf den äquivalenten Funktionsaufruf zurückgegriffen. Als Modell wurde ein PID-Regler gewählt. Die Simulationszeiten zwischen Simulation mit und ohne MAM unterscheiden sich, wie schon in Tabelle 4-1, ebenfalls nicht.

4.1.2 Konservative Schnittstellen statt digitaler Schnittstellen

Die nächste Möglichkeit, die die MAM bietet, ist die Beschreibung eines digitalen Signal-Ports mit analogen Terminals. Dadurch wird es möglich, digitale Komponenten auf hoher Abstraktionsebene gegen elektrische Netzwerke auf niedriger Abstraktionsebene auszutauschen. Außerdem kann diese Funktionalität auch der Verbindung digitaler Logikbeschreibungen mit analogen Verhaltensbeschreibungen, z. B. parasitären Elementen, dienen. Das Modell zum Test einer solchen Konfiguration soll den Aufbau entsprechend Bild 4-4 besitzen. Das Simulationsergebnis ist in Bild 4-5 dargestellt. Für die beiden MOS-Transistoren kommt dabei ein einfaches Modell mit je einer Gleichung für Sperr-, Linear- und Sättigungsbereich sowie konstanten Gate-Source- und Gate-Drain-Kapazitäten zum Einsatz. Parasitäre Effekte wie verschliffene Signalflanken (Terminal `t2`) und daraus folgende Signalverzögerungen (Signal `s2`) kann man deutlich erkennen.

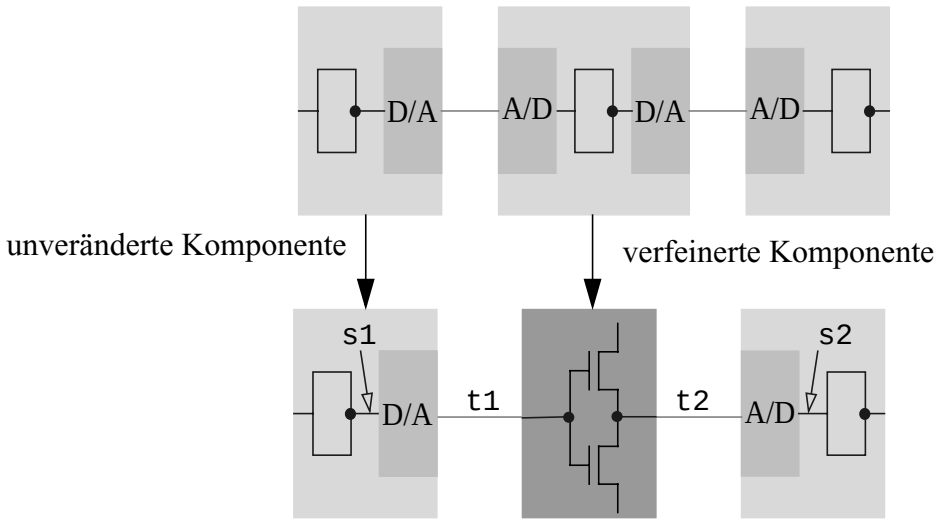


Bild 4-4 Testmodell „Terminal statt Signal“

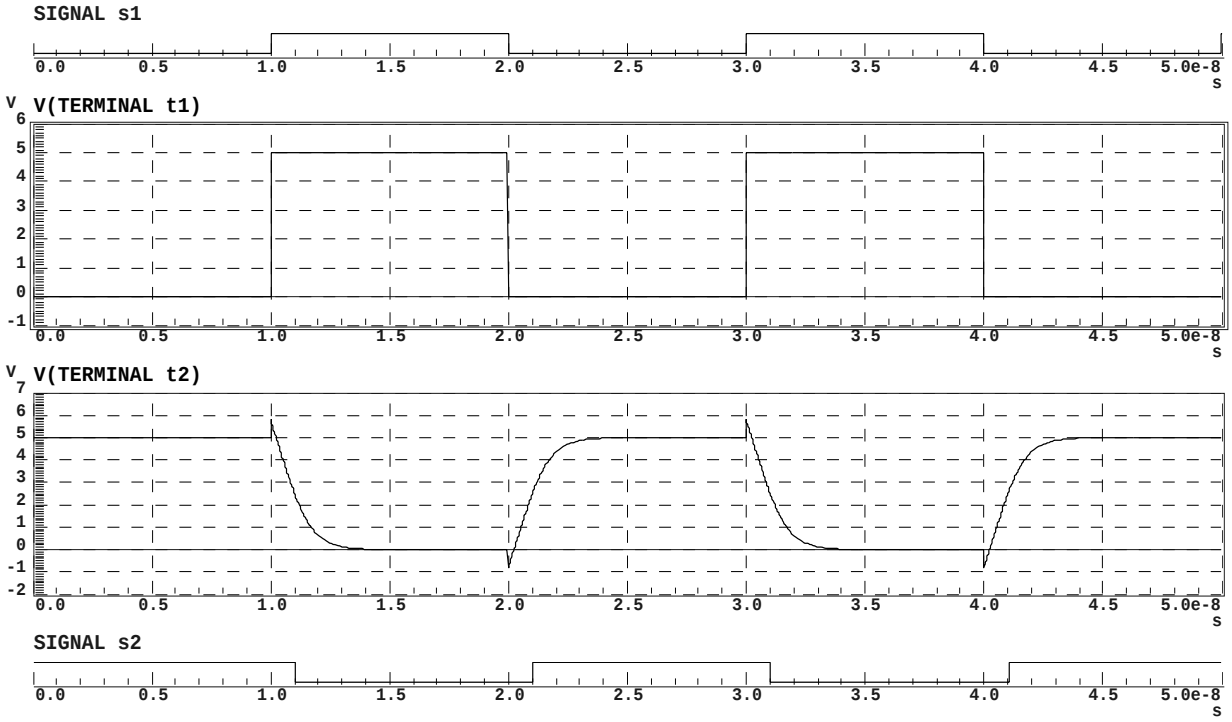


Bild 4-5 Simulationsergebnis des Testmodells

Zur Bestimmung des Einflusses der Schnittstelle wurde darüber hinaus eine Konfiguration aus 100 abstrakten Modellen simuliert. Die sich ergebenden Simulationszeiten sind in Tabelle 4-2 zu finden. Für die Simulationsparameter der Transient-Simulation kommen bis auf die Einstellung für die maximale Schrittweite die in Abschnitt 4.1.1 angegebenen zum Einsatz. Die maximale Schrittweite wurde auf 1 ms gesetzt, da in diesem Fall eine maximale analoge Schrittweite von 1 ns die Simulationszeiten zu stark vergrößert hätte. Eine AC-Simulation ist aufgrund der digitalen Komponenten nicht möglich.

	Mittelwert von drei Simulationsläufen ohne MAM	Mittelwert von drei Simulationsläufen mit MAM
Modell nach Bild 4-4, abstrakte Komponente	226 ms	510 ms
Modell nach Bild 4-4, verfeinerte Komponente	ohne Änderungen am Systemmodell nicht möglich	1,76 s
100 Modelle mit je einem Negator	400 ms	1 min, 40 s
1 Modell mit 100 abstrakten Negatoren	310 ms	580 ms

Tabelle 4-2 Simulationszeiten „Terminal statt Signal“

Wie zu erwarten war, benötigt das Modell, welches die MAM benutzt, eine längere Simulationszeit als das rein digitale Modell. Eine Anwendung ist daher nur selektiv sinnvoll. Dies bedeutet,

- daß nur diejenigen abstrakten Modelle ein Interface nach MAM erhalten sollten, bei denen ein Austausch gegen ein analoges Modell absehbar ist und
- keinesfalls pauschal alle Modelle mit einer Schnittstellen entsprechend der MAM ausgerüstet werden sollten.

Die Zeiten in Tabelle 4-2 stellen bereits *worst case* Szenarien für die Anwendung der MAM dar, da bei diesen Modellen wenig Verhalten beschrieben ist. Dadurch hebt sich der Einfluß der Schnittstelle besonders hervor. Da in den Modellen realer Systeme die Simulation der eigentlichen Modellbeschreibung mehr Rechenzeit benötigt als in den Testmodellen, werden die Differenzen der Simulationszeiten geringer ausfallen. Im Hinblick auf die sich ergebenden Möglichkeiten zur Modellkonfiguration ist die selektive Anwendung der MAM auch für diesen Anwendungsfall sinnvoll. Alternativ besteht unter bestimmten Umständen (wie in Abschnitt 3.2.4 beschrieben) die Möglichkeit, auch die Signale der hohen Abstraktionsebene auf niedriger Abstraktionsebene zu verwenden, was auf hoher Abstraktionsebene dann keine Verlängerung der Simulationszeiten hervorruft.

Das hier gewählte Beispiel dient lediglich der Veranschaulichung des Sachverhaltes. Wenn man die Möglichkeit hat, Ergebnisse der Simulation des detaillierten Modells in das abstraktere Modell zu integrieren, z. B. im Rahmen einer *Backannotation,* so ist ein solches Verfahren der MAM in der Regel vorzuziehen.

Bild 4-6 zeigt eine weitere Anwendungsmöglichkeit der Terminal-zu-Signal- und Signal-zu-Terminalkonverter. Mit diesen Konvertern können analoge Beschreibungen digitaler Ein- und Ausgänge (z. B. TTL- oder CMOS-Technologie) und digitale Modelle unter Nachbildung z. B. der *resolution function* für den Datentyp `std_logic` aus den Package `std_logic_1164` miteinander verbunden werden. Ein entsprechendes Simulationsergebnis einer Konfiguration aus abstrakten Modellen zeigt Bild 4-7.

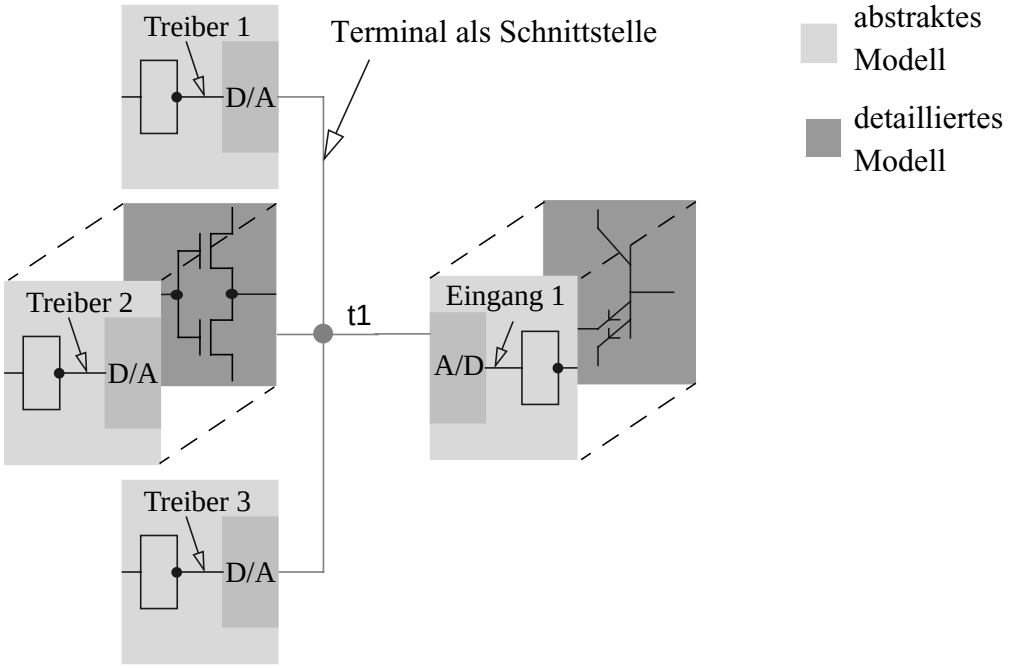


Bild 4-6 Testmodell zur Nachbildung einer resolution function auf einem Terminal

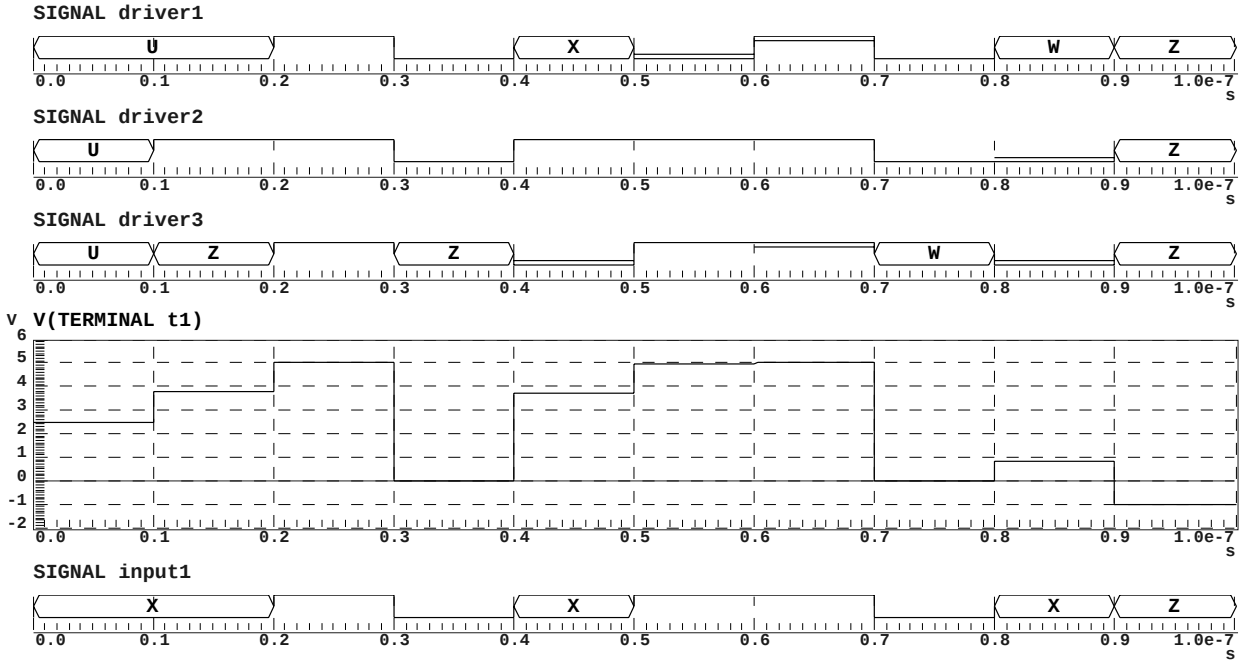


Bild 4-7 Simulationsergebnis der Nachbildung einer resolution function auf einem Terminal

4.1.3 Digitale Schnittstellen statt nichtkonservativer Schnittstellen

Erfolgt die Beschreibung eines Systems auf hoher Abstraktionsebene mit analogen Signalflußmodellen, und werden diese dann durch digitale Implementierungen ersetzt, so kann man den Ansatz nach Abschnitt 3.2.5 anwenden. Da bei der Anwendung dieses Ansatzes die Verwendung digitaler

Signale auf hoher Abstraktionsebene anstelle von Quantities empfohlen wird, soll hier zur Bestimmung des Einflusses auf die Simulationszeit das Beispiel in Bild 4-8 betrachtet werden.

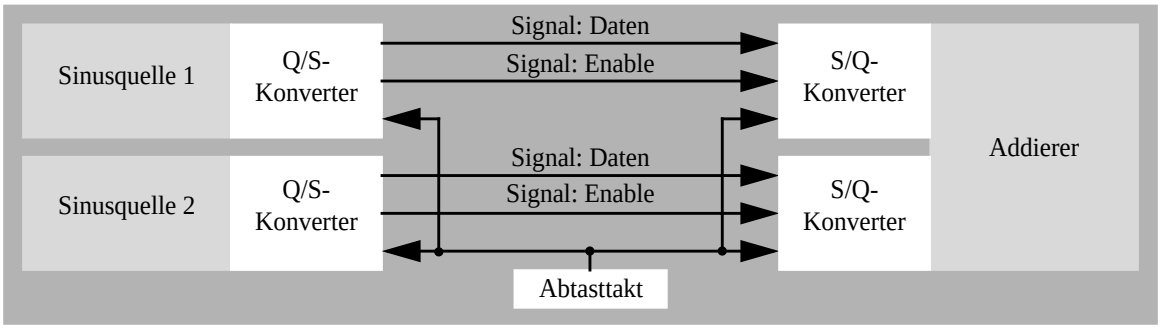


Bild 4-8 Testmodell „Signal statt Quantity“

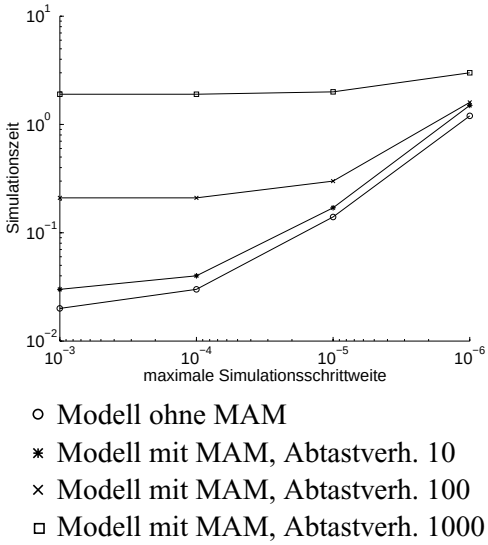
Als wesentlicher Faktor für die Simulationsgeschwindigkeit erweist sich dabei der Abtasttakt, mit dem die Quantity/Signal- und Signal/Quantity-Konverter angesteuert werden, bzw. das Verhältnis aus Abtasttakt zur höchsten im System vorkommenden Frequenz. Laut Abtasttheorem muß dieses Verhältnis gelten: $f_{clk} / f_{Signal} \geq 2$. Für eine direkte graphische Auswertbarkeit der Simulationsergebnisse empfiehlt sich aber ein Verhältnis von mindestens 10. Die Simulationszeiten wurden für Verhältnisse von 10, 100 und 1000 bestimmt. Die maximal zugelassene Simulationsschrittweite stellt einen weiteren zu berücksichtigenden Parameter dar. Die Simulationsparameter sind:

- Simulations-Zeit: 10 ms
- minimale Schrittweite: 10^{-12} s
- maximale Frequenz im System 320 Hz

Die ermittelten Simulationszeiten zeigt Tabelle 4-3.

	Simulationszeiten bei einer maximal zugelassenen Simulationsschrittweite von			
	10^{-3} s	10^{-4} s	10^{-5} s	10^{-6} s
Modell ohne MAM	0,02	0,03 s	0,14 s	1,2 s
Modell mit MAM, Abtastverhältnis 10	0,03	0,04 s	0,17 s	1,5 s
Modell mit MAM, Abtastverhältnis 100	0,21	0,21 s	0,30 s	1,6 s
Modell mit MAM, Abtastverhältnis 1000	1,9	1,9 s	2,0 s	3,0 s

Tabelle 4-3 Simulationszeiten „Signal statt Quantity“



Die Simulationsgeschwindigkeit verringert sich mit MAM bei einem Abtastverhältnis von 10 um ca. 30 %. Bei einer maximalen Simulationsschrittweite von 1 ms verlängert sich die Simulation zwar um 50 %, die Simulation ohne MAM zeigt aber mit dieser Einstellungen bereits deutliche Verzerrungen der Kurvenform. Je höher die Abtastrate gewählt wird, desto deutlicher verlangsamt sich die Simulation. Erfordern die Modelle kleinere maximale Simulationsschrittweiten, desto größer wird der Raum zur Festlegung des Abtastverhältnisses.

Die hier beschriebene Vorgehensweise hat einen Einfluß darauf, wie oft das analoge Gleichungssystem berechnet wird, aber nur in geringem Maße auf das Gleichungssystem selbst. Daher ist mit zunehmender Komplexität der Modelle nicht mit einer Verschlechterung der Simulationszeitverhältnisse zu rechnen. Einflüsse auf die Simulatorkonvergenz wurden in dem einfachen Testbeispiel nicht beobachtet, können aber insbesondere in mehrfach differenzierenden und nichtlinearen Komponentenbeschreibungen nicht ausgeschlossen werden.

4.1.4 Einsatz digitaler Abstraktionswandler

Die einfachste Form der digitalen Schnittstellenanpassung stellen Funktionen zur Datentypskonvertierung dar. Diese wandeln ein zu übertragendes Signal vom Typ `real` oder `integer` in einen Signal vom Typ `bit_vektor` um und in der Empfängerkomponente entsprechend zurück. Soll die Kommunikation der Komponenten ein bestimmtes Protokoll einhalten, so müssen die in Abschnitt 3.2.2 beschriebenen Abstraktionswandler zum Einsatz kommen. Die sich ergebenden Simulationszeitunterschiede für eine Umwandlung einer Sequenz von 100.000 8 Bit Integerwerten in Bitvektoren und zurück sowie die Transformation dieser Sequenz ins RS232-Protokoll und zurück enthält Tabelle 4-4. Dabei wurden im ersten Fall Empfänger- und Senderkomponente ohne weiteres Verhalten beschrieben, so daß die sich ergebenden Simulationszeiten ebenfalls einem *worst case* Szenario entsprechen, bei dem sich die Simulationszeit der Komponente aus der Simulationszeit ihrer Schnittstelle ergibt. Im zweiten und dritten Fall wurde die Komplexität der Komponenten auf 1000 bzw. 10000 Zuweisungen erhöht, so daß die Simulationszeit der Komponente primär von ihrer Verhaltensbeschreibung bestimmt wird.

Die Simulation des Systems mit RS232-Protokoll konnte bis zum gegenwärtigen Zeitpunkt nicht mit VHDL-AMS erfolgen, da „AdvanceMS“ einige zur Beschreibung der Abstraktionswandler-Prozeduren notwendigen Statements nicht beherrscht. Statt dessen wurden Konfigurationen aus Datenübertragung ohne MAM, mit einfacher Datentypskonvertierung und mit RS232-Abstraktionswandler (auf der Basis von rein digitalem VHDL) mittels „Modelsim“ simuliert. Leider zeigt „Modelsim“ nicht die benötigte Simulationszeit an, so daß die Simulationszeit manuell erfaßt werden mußte. Die Simulationszeiten für die Übertragung von 1 MByte sind in Tabelle 4-5 zusammengefaßt. Auch hier wurden, wie in Tabelle 4-4, die zwei Fälle mit schnittstellen- und verhaltensbestimmter Simulationszeit untersucht.

Wie zu erwarten war, steigt die benötigte Simulationszeit mit dem Detaillierungsgrad der Schnittstelle. Da hierbei der Anstieg der Rechenzeit konstant bleibt, gewinnt die MAM mit steigendem Umfang an Verhaltensbeschreibungen im Modell an Effizienz.

	Mittelwert von drei Simulationsläufen ohne MAM	Mittelwert von drei Simulationsläufen mit MAM (einfache Datentypskonvertierung)	Mittelwert von drei Simulationsläufen mit MAM (RS232-Protokoll)
Modell ohne Verhalten in Sender und Empfänger: Simulationszeit	1,9 s	4,5 s	Simulation bisher nicht möglich
Modell mit Komplexität von 1000 Zuweisungen: Simulationszeit	23 s	26 s	Simulation bisher nicht möglich
Modell mit Komplexität von 10000 Zuweisungen: Simulationszeit	3 min, 25 s	3 min, 29 s	Simulation bisher nicht möglich

Tabelle 4-4 „AdvanceMS“ Simulationszeiten „Bit und Bitvektor statt Integer“

Als positiver Nebeneffekt erhält der Entwerfer bei Benutzung der Abstraktionswandler bereits auf hoher Abstraktionsebene – und damit zeitig im Entwurfsablauf – eine Information, ob sein favorisiertes Übertragungsverfahren überhaupt in der Lage ist, die anfallenden Datenmengen zu bewältigen.

	Mittelwert von drei Simulationsläufen ohne MAM	Mittelwert von drei Simulationsläufen mit MAM (einfache Datentypskonvertierung)	Mittelwert von drei Simulationsläufen mit MAM (RS232-Protokoll)
Modell ohne Verhalten in Sender und Empfänger: Simulationszeit	7 s	9 s	2 min
Modell mit Komplexität von 1000 Zuweisungen: Simulationszeit	44 s	46 s	2 min, 40 s
Modell mit Komplexität von 10000 Zuweisungen: Simulationszeit	6 min, 30 s	6 min, 30 s	8 min, 20 s

Tabelle 4-5 „Modelsim“ Simulationszeiten „Bit und Bitvektor statt Integer“

4.1.5 Fazit

Die in diesem Abschnitt vorgestellten Szenarien zeigen anhand einfacher Beispiele, daß die Simulationszeiten durch die Verwendung der detaillierten Schnittstellen ansteigen können. Dieser Anstieg fällt in den *worst case* Szenarien (viel Schnittstelle, wenig Verhalten) unter Umständen

deutlich, in *typical case* Szenarien eher moderat aus. Es ist daher ratsam, die Methode der MAM nicht pauschal auf alle Schnittstellen des Systems anzuwenden, sondern selektiv für kritische Komponenten, wie in Bild 3-13 in Abschnitt 3.4 dargestellt.

Die Untersuchungen zur Simulationszeit wurden primär mit dem Simulator „AdvanceMS“ von Mentor Graphics durchgeführt. Vergleichende Untersuchungen mit dem VHDL-AMS Simulator des „Simplorer“ von Ansoft haben die prinzipielle Anwendbarkeit der MAM auch mit anderen VHDL-AMS Simulatoren gezeigt. Die Simulationszeitdifferenzen zwischen konventionellem Entwurf und Entwurf mit MAM können je nach verwendetem Werkzeug jedoch unterschiedlich ausfallen.

4.2 Anwendung im System „Demonstrator Vibrationssensor-Array“

Dieser Abschnitt soll die Anwendbarkeit der MAM in der Modellierung und Simulation komplexer mikromechanischer und mikroelektronischer Systeme demonstrieren. Als Demonstrationsobjekt kommt dabei das im Rahmen des SFB 379 entstandene Meß- und Klassifikationssystem zur Verschleißüberwachung mittels Vibrationsanalyse zum Einsatz [6]. In [1], [4] und [5] werden Aspekte der Anwendung der MAM im Entwurf dieses Systems betrachtet.

4.2.1 Vorstellung des Meßsystems

Die Verschleißerkennung an Werkzeugmaschinen spielt bei der Sicherung der Produktqualität eine bedeutende Rolle. Bislang kommen dazu vorrangig Messungen der Kräfte an den Schneidwerkzeugen zum Einsatz. Eine alternative Methode ist die Auswertung vom Maschinen-Vibrationen. Ansätze dazu existieren seit vielen Jahren [97]. In der Regel verwenden solche Ansätze jedoch teure breitbandige piezo-keramische Beschleunigungssensoren [98], [99].

Ein neuartiges, an der TU Chemnitz entwickeltes Sensorarray [100] arbeitet dagegen schmalbandig in Resonanz zur jeweils zu messenden Frequenzkomponente. Dies ermöglicht eine einfachere Signalverarbeitung bei besserem Signal zu Rauschverhältnis und niedrigeren Kosten.

Das Meßsystem enthält ein Sensor-Array, das aus acht einzelnen Feder-Masse-Resonatoren besteht, deren Resonanzfrequenz elektrostatisch abgestimmt werden kann. Das vom Sensor gelieferten Signal wird zunächst analog und anschließend digital in einem Mikrocontroller weiterverarbeitet. Ein Verstärker erzeugt die zur Abstimmung notwendigen Spannungen. Am Ende entscheidet ein Fuzzy-Pattern-Klassifikator [101], [102], ob die gemessenen Daten z. B. einem scharfen oder einem verschlissenen Drehmeißel zuzuordnen sind. Bild 4-9 zeigt das Blockschaltbild des Systems.

4.2.2 Modelle des Meßsystems

Die Stimuli der Simulation werden von einer virtuellen Werkzeugmaschine als Umgebungsmodell erzeugt. Dieses Modell ist in der Lage, gemessene Daten im Zeit- oder Frequenzbereich wiederzugeben oder fiktive Daten zu erzeugen. Das Modell dieser Komponente steht nur in abstrakter Form zur Verfügung.

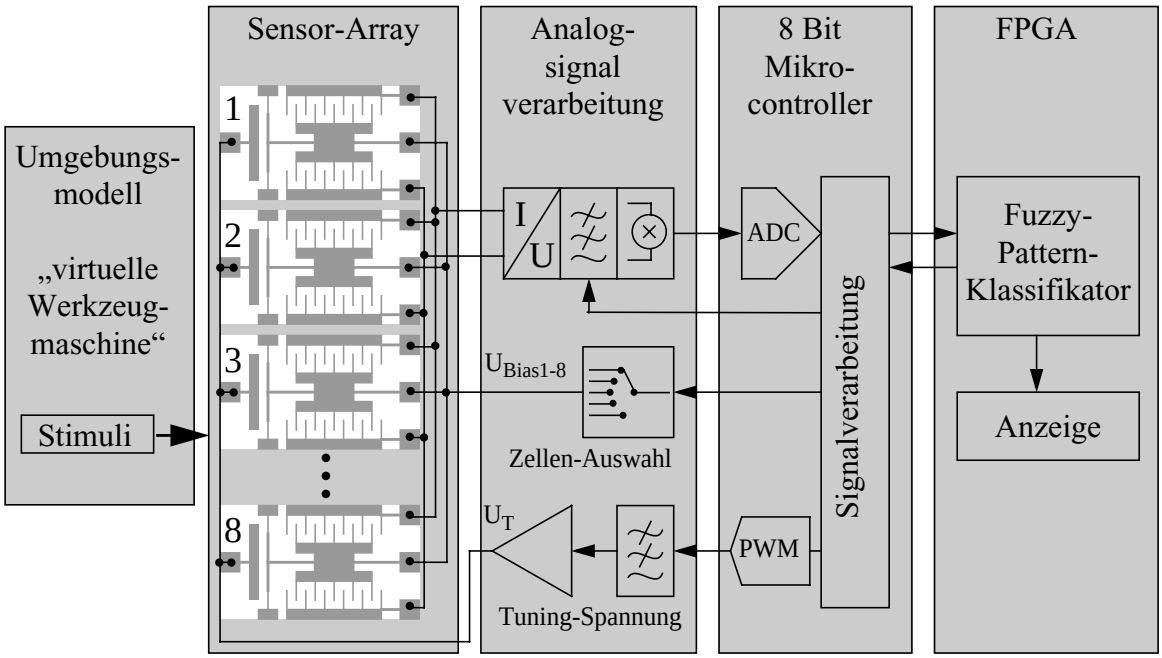


Bild 4-9 Blockschaltbild des Meßsystems

Das Sensor-Array wandelt die Vibrationen in ein elektrisches Signal um. Da die resultierenden Ströme sehr gering sind, befindet sich direkt am Sensor ein Strom-Spannungs-Wandler mit Verstärker. Für die Modellierung wurde zunächst ein abstraktes Sensormodell auf der Basis von konzentrierten Starrkörperelementen [103] beschrieben [6], das im Verlauf der Entwicklungsarbeiten gegen ein wesentlich genaueres, automatisch erzeugtes *Reduced Order Modell* [104] ausgetauscht wurde [1].

Die Analogsignalverarbeitung verstärkt und filtert das Meßsignal mittels eines Lock-In-Verstärkers. Die zur Abstimmung der Sensoren notwendigen Spannungen (30 V) erzeugt ein Spannungsverstärker. Das Modell der Analogsignalverarbeitung enthält verschieden abstrakte Modelle der einzelnen Bauteile, so daß die MAM hier auf das Modell der Komponente an sich sowie auf deren Submodelle angewendet werden muß.

Der Mikrocontroller steuert den Meßablauf, führt eine Selbstkalibrierung durch und digitalisiert das Meßsignal. Da eine Nachbildung des Mikrocontrollers auf Register-Transfer-Ebene den Modellierungs- und Simulationsaufwand extrem erhöhen würde, entstand ein Modell, das nur die zur Steuerung des Meßablaufs und die zur Meßwertverarbeitung notwendige Algorithmen abstrakt nachbildet.

Am Ende der Signalverarbeitungskette ordnet der Fuzzy-Pattern-Klassifikator die gemessenen Daten verschiedenen Klassen von Zuständen zu. Für diese Aufgabe wurde auf der Basis eines „Virtex“-FPGAs der Firma Xilinx ein Spezialprozessor entwickelt, dessen Befehlssatz und Schnittstellen auf die Belange der Fuzzy-Pattern-Klassifikation zugeschnitten sind. Für erste Simulationen entstand mit VHDL ein abstraktes Modell, das lediglich den Algorithmus der Klassifikation abarbeitet. Anschließend erfolgte die Entwicklung des Prozessors auf RT-Ebene, so daß

einerseits daraus das Konfigurationsfile des FPGAs synthetisiert werden konnte und er andererseits gemeinsam mit den weiteren Komponenten das Systems in VHDL-AMS simuliert werden kann.

Die sich ergebenden Systemkonfigurationen zeigt Bild 4-10.

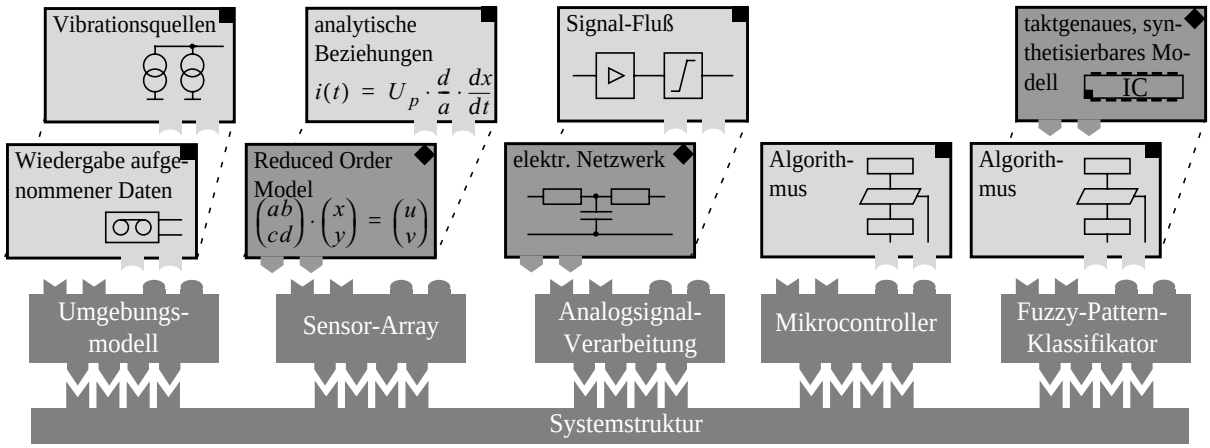


Bild 4-10 Konfigurationsmöglichkeiten des Meßsystems

4.2.3 Schnittstellengestaltung

Die in den Abschnitten 4.2.3.1 bis 4.2.3.3 getroffenen Aussagen beschreiben die Überlegungen des im Rahmen des Top-Down-Entwurfs mit MAM notwendigen Arbeitsschrittes „Suche kritischer Schnittstellen“ entsprechend Bild 3-13 in Abschnitt 3.4.

4.2.3.1 Schnittstellen Umgebung – Sensor – Elektronik

Da es sich bei dem Sensor um einen Vibrationssensor handelt, liegt es nahe, über die Schnittstelle Beschleunigungswerte zu übertragen. Benötigt man Informationen zu Geschwindigkeit und Auslenkung, so muß die Beschleunigung ein- bzw. zweimal integriert werden. Dabei entsteht in VHDL-AMS das Problem, daß die sich bei einer unbestimmten Integration ergebenden Integrationskonstanten nicht explizit gesetzt werden können. Dies hat zur Folge, daß sich ein Offset einstellen kann und der Sensor in der Simulation, bildlich gesprochen, die Drehmaschine hinter sich herzieht. Deshalb werden über diese Schnittstelle Auslenkungen anstelle von Beschleunigungen übertragen. Die zugrunde liegenden Geschwindigkeiten und Beschleunigungen lassen sich dann durch Ableitung eindeutig bestimmen. Die Anfangswerte für nach der Zeit abgeleitete Größen werden vom Simulator auf Null gesetzt, was in der Regel kein Hindernis für die Simulation darstellt.

Man hat nun die Möglichkeit, diese Auslenkung rückwirkungsfrei über eine Quantity zu übertragen. Da aber der Sensor selbst eine Masse besitzt und dadurch u. U. das Schwingungsverhalten des zu überwachenden Objekts beeinflußt, wurde als Schnittstelle ein Terminal gewählt. Weiterhin ist bei diesem Meßsystem zu beachten, daß der Sensor an stark beanspruchten Maschinenteilen befestigt werden soll, die sich im Betrieb beträchtlich erwärmen können. Daher ist ein weiteres Terminal für die Übertragung von Temperaturen vorzusehen.

Der Ausgang des Sensors liefert ein sehr kleines differentielles Stromsignal. Da sich zur Verringerung der Störeinflüsse im späteren Aufbau eine erste Verstärkerstufe direkt am Sensor befindet, soll diese auch im Sensormodell ein Bestandteil des Sensors sein. Da Rückwirkungen aus der nachfolgenden Signalverarbeitungsstufe nicht ausgeschlossen werden können, ist das Sensormodell mit einem Terminal für die resultierende Ausgangsspannung sowie mit Terminals für positive und negative Betriebsspannung und elektrischer Masse auszustatten.

Weiterhin verfügt der Sensor über Anschlüsse für eine Polarisationsspannung zur Erzeugung der Ausgangsströme sowie für die Tuningspannung zur Abstimmung der Sensoren. Da hier ebenfalls mit Rückwirkungen in Form von fließenden Strömen zu rechnen ist, müssen auch für diese Anschlüsse Terminals vorgesehen werden.

4.2.3.2 Schnittstellen der Analogelektronik

Die Anforderungen an die Schnittstellen zum Sensor wurde bereits im vorherigen Abschnitt beschrieben. Die Analogelektronik selbst besteht aus einem Netzwerk elektrischer Bauelemente, die wiederum verschieden abstrakt vorliegen. Wenn auch hier ein Austausch zwischen verschieden abstrakten Bauteilen möglich sein soll, muß die MAM auch innerhalb dieser Komponente an den Schnittstellen zu ihren Submodellen angewandt werden. Da es sich bei der Analogsignalverarbeitung um ein elektrisches Netzwerk handelt, sind Terminals als Interface unumgänglich. Zusätzlich sind die Schnittstellen mit Terminals für die Betriebsspannung auszustatten. Die Schnittstelle zu einem OPV muß somit Anschlüsse für

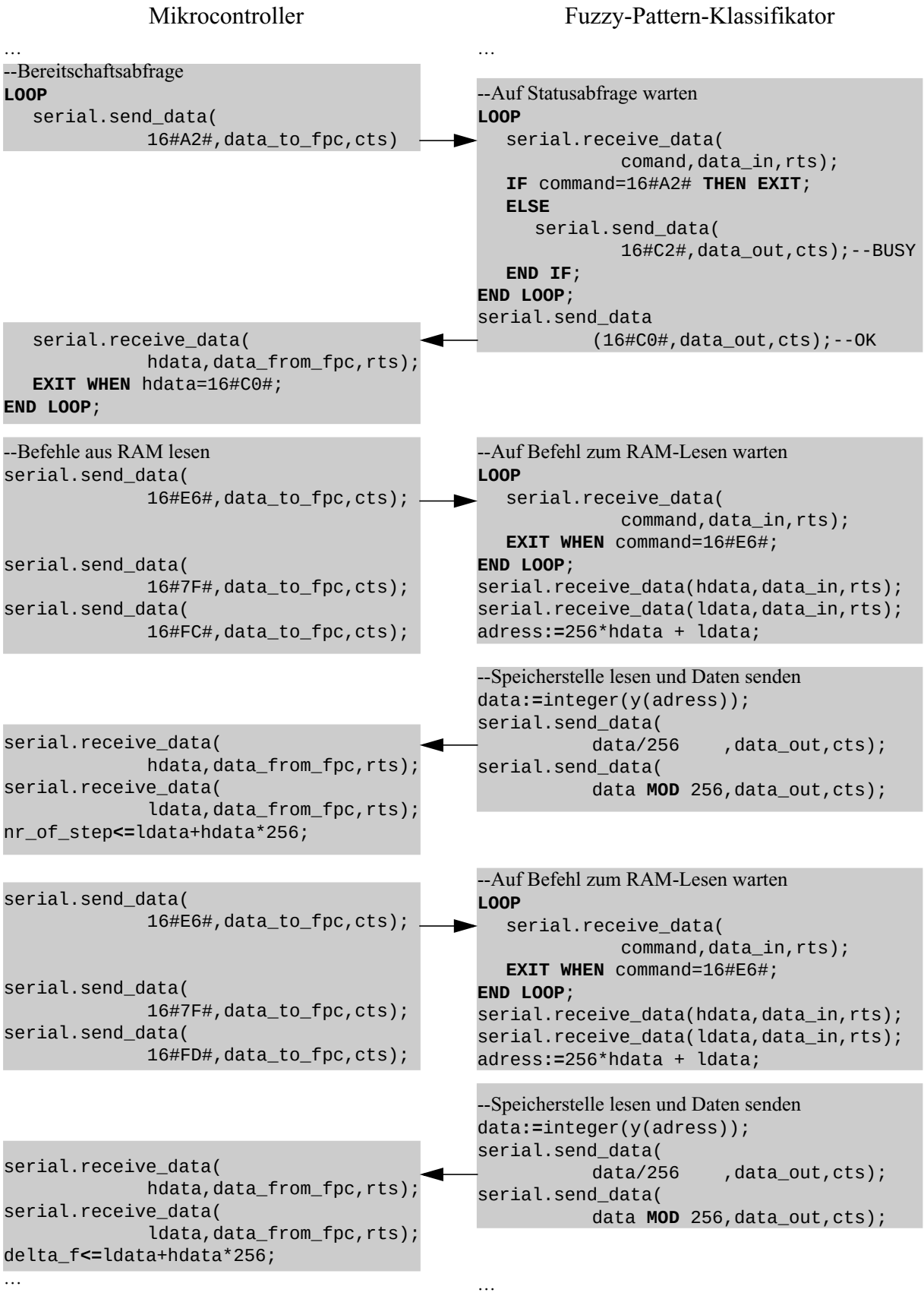
- invertierenden und nichtinvertierenden Eingang,
- Ausgang sowie
- positive und negative Betriebsspannung

aufweisen, auch wenn in der ersten abstrakten Modellierung der Signalverarbeitung nur gesteuerte Quellen zum Einsatz kommen. Die Analogsignalverarbeitung ist selbst ebenfalls mit Anschlüssen für positive und negative Betriebsspannung auszustatten. Will man im Modell thermische Effekte wie Selbsterwärmung oder temperaturabhängige Bauteile simulieren, kann das Modell mit thermischen Knoten ausgestattet werden, was aber in dieser Simulation nicht der Fall ist.

Die Schnittstelle zum A/D-Wandler läßt sich hier als praktisch rückwirkungsfrei betrachten, so daß eine Quantity zum Einsatz kommen kann. Dies gilt insbesondere auch im Hinblick darauf, daß für den Mikrocontroller, der den A/D-Wandler enthält, nur ein abstraktes Modell vorgesehen ist. Darüber hinaus erzeugt der Mikrocontroller digitale Signale für den Lock-In-Verstärker und zur Selektion von Sensorzellen. Diese können als Signale mit den Typen `bit` bzw. `bit_vector` ausgelegt werden.

4.2.3.3 RS232-Schnittstelle zum Fuzzy-Pattern-Klassifikator

Fuzzy-Pattern-Klassifikator und Mikrocontroller kommunizieren über eine RS232-Verbindung. Über diese Verbindung werden sowohl Konfigurationsdaten als auch die Meßwerte ausgetauscht. Ein Ausschnitt dieser Kommunikation ist in Quelltext 4-2 dargestellt.



Quelltext 4-2 Ausschnitt aus der Kommunikation von Mikrocontroller und Fuzzy-Pattern-Klassifikator

Die Kommunikation zwischen dem Mikrocontroller und dem Fuzzy-Pattern-Klassifikator erfolgt dabei nach folgendem Schema:

- Zuerst „fragt“ der Mikrocontroller den Fuzzy-Pattern-Klassifikator, welche Messungen er durchführen soll, indem er bestimmte Speicheradressen im Adreßraum des Fuzzy-Pattern-Klassifikators ausliest.
- Als nächstes führt der Mikrocontroller die angeforderten Messungen aus und schreibt die Ergebnisse auf dafür vorgesehene Adressen im Speicher des Fuzzy-Klassifikators.
- Im dritten Schritt startet der Mikrocontroller die Klassifikation. Ist die Klassifikation beendet, kann der Mikrocontroller neue Meßaufgaben beim Fuzzy-Pattern-Klassifikator abrufen.

Die Kommunikation soll über eine RS232-Schnittstelle mit speziellem *Handshake* erfolgen. Um später die UART des synthetisierbaren RT-Modells mit in die Systemsimulation einbeziehen zu können, müssen bereits die abstrakten Modelle den seriellen Datenstrom der RS232-Schnittstelle zur Kommunikation benutzen und dürfen keinen Austausch von z. B. Integer-Werten vornehmen. Als Konsequenz daraus verwenden das Mikrocontroller-Modell und das abstrakte Modell der Fuzzy-Pattern-Klassifikation die in Abschnitt 3.2.2, S. 48 beschriebene Abstraktionswandler-Prozeduren. Da als Schnittstelle Signale vom Typ `std_logic` dienen sollen, wurden diese Prozeduren so modifiziert, daß sie anstelle der Signale vom Typ `bit` Signale des Typs `std_logic` verwenden. Dank des *open source* Ansatzes stellen solche Anpassungen auch später beim Anwender der Methode kein Problem dar.

4.2.4 Ergebnisse

Die Gesamtsystemsimulation wurde auf einer SUN Blade2 mit 900 MHz CPU und 4 GByte RAM durchgeführt. Das Bild 4-11 enthält das Simulationsergebnis. Die erste Kurve stellt die an den Sensor angelegte Tuning-Spannung dar. Die zweite Kurve zeigt das gefilterte und verstärkte Ausgangssignal der analogen Signalverarbeitung und die dritte Kurve zeigt die vom A/D-Wandler erfaßten Meßdaten. Die unterschiedlich grau schattierten Hintergründe entsprechen der vom Mikrocontroller angesteuerten Sensorzelle.

Die gemessenen Vibrationen werden vom Klassifikator mit einem Sympathiewert von 0,98 der Gutklasse zugeordnet.

Bei der Simulation des Gesamtsystems mit „AdvanceMS“ muß die Kommunikation zwischen Mikrocontroller und Klassifikator derzeit noch aus den in Abschnitt 4.1.4 genannten Gründen abstrakt erfolgen. Für die Bestimmung des Einflusses des Abstraktionsgrades der Kommunikation auf die Simulationsgeschwindigkeit wurde ein Teilsystem bestehend aus Klassifikator und den digitalen Subsystemen des Mikrocontrollers rein digital simuliert.

Zum Vergleich erfolgte die Modellierung und Simulation des gesamten Systems ohne MAM. Dazu konnten im Modell der virtuellen Werkzeugmaschine Teile des Verhaltens, die die Auswirkungen der Rückkopplung des Sensors auf den Drehmeißel beschreiben, entfernt werden. Im Sensormodell mußten, wie in Abschnitt 3.4, S. 66 für den konventionellen Top-Down-Entwurf beschrieben, nicht unerhebliche Anpassungen am Reduced-Order-Modell des Sensors vorgenom-

men werden. Die Verbindung des Sensors mit der Analogelektronik erfolgt weiterhin mit Terminals, da hier Rückwirkungen einen entscheidenden Einfluß auf das Simulationsergebnis haben.

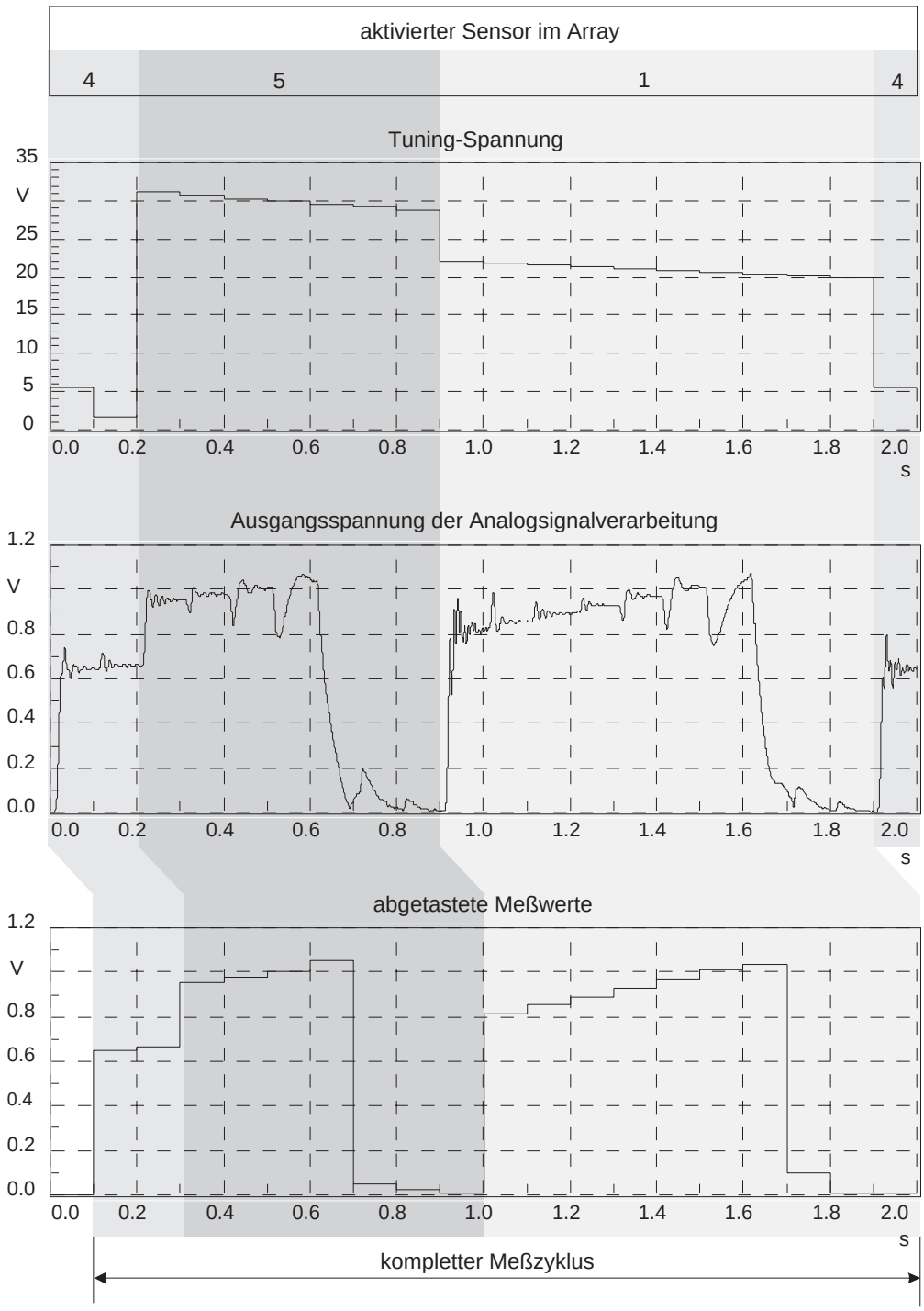


Bild 4-11 Simulationsergebnis des Gesamtsystems

Die sich ergebenden Simulationszeiten sind in Tabelle 4-6 dargestellt. Es ist zu erkennen, daß sich die Simulationszeit mit MAM um ca. 16 % gegenüber der Simulationszeit eines konventionell entworfenen Systems aus abstrakten Modellen erhöht und daß eine Einbeziehung der digitalen Kommunikation in die Gesamtsystemsimulation mit MAM keine weitere Verlängerung der

Simulationszeit zur Folge hat. Im Gegensatz zum konventionellen Entwurf hat man aber die Möglichkeit, im Systemmodell für jede Komponente eine verfeinerte Architekturalternative einzusetzen und so jeden Aspekt des System mit der Genauigkeit der detaillierten Komponenten simulieren zu können.

	heterogenes Gesamtsystem, abstrakte digitale Kommunikation	digitale Kommunikation (Zeiten geschätzt, siehe 4.1.4)
System mit MAM	1 h, 25 min	ca. 1 s
konventionelles System abstrakt	1 h, 11 min	ca. 1 s
konventionelles System detailliert	2 h, 20 min	ca. 1 s

Tabelle 4-6 Simulationszeiten des Gesamtsystems

Die Simulation einer Konfiguration nur aus detaillierten Komponenten erfordert knapp die doppelte Simulationszeit gegenüber der Simulation mit MAM. Weiterhin ist anzumerken, daß sich das detaillierte Modell des Sensors auf der Basis des *Reduced Order Modeling* [4], [46] schneller simulieren läßt als das abstrakte Modell. Wäre dies nicht der Fall, so würde der Simulationszeitunterschied zwischen System mit MAM und dem konventionellen System aus detaillierten Modellen noch deutlicher ausfallen. Eine Verschlechterung der Simulatorkonvergenz oder der Simulationsstabilität konnte nicht beobachtet werden.

4.2.5 Fazit

Die Anwendung der Multi-Architecture-Modellierung beim Entwurf des komplexen heterogenen Systems „Demonstrator Vibrationssensor-Array“ hat gezeigt, daß die Bestimmung der notwendigen Schnittstellen auf niedrigem Abstraktionsniveau bereits in frühen Stadien des Entwurfsablaufes durchführbar ist. Die Anwendung der MAM führt zu einer Vereinfachung und damit zu einer Fehlerminimierung beim Austausch von unterschiedlich abstrakten Komponentenmodellen.

4.3 Übertragbarkeit auf andere Hardwarebeschreibungssprachen

Eine Übertragung der MAM auf andere Hardwarebeschreibungssprachen ist prinzipiell möglich. Die Effektivität des Ansatzes und der Umfang der Anwendbarkeit der Bestandteile der MAM ist aber bei den einzelnen Beschreibungsformen unterschiedlich:

- Für SystemC-AMS werden in Abschnitt 5, S. 89 ff. Möglichkeiten erörtert, unterschiedlich abstrakte Modelle zu verbinden.
- In Spice-basierten Simulatoren kann die MAM nur bedingt eingesetzt werden. In Spice ist es möglich, Modelle unterschiedlich abstrakt zu beschreiben (z. B. Beschreiben einer Diode als Schalter, Kennlinientabelle oder mit Diodengleichung). Da nur analoge konservative Knoten zur Verfügung stehen, ist die Schnittstellenproblematik weit weniger ausgeprägt und reduziert sich in der Regel auf die Bereitstellung von Masse- und Betriebsspannungsknoten.

- Ähnliches gilt für „Matlab/Simulink“, wobei hier nur analoge, nichtkonservative Knoten verwendet werden.
- In Verilog-AMS sollte der Einsatz der MAM ebenfalls möglich sein. Eine detaillierte Untersuchung konnte mangels zur Verfügung stehenden Verilog-AMS Simulators allerdings nicht durchgeführt werden.

Die Anwendung der MAM in einem proprietären Format hängt von verschiedenen Eigenschaften der jeweiligen Implementierung ab. Primär muß das System über die Möglichkeit zur Beschreibung verschieden abstrakter Modelle verfügen. Andernfalls erübrigt sich der Einsatz der MAM. Eine weitere wichtige Eigenschaft des Systems sollte darin bestehen, die Modellgleichungen der einzelnen Komponenten in einem gemeinsamen Gleichungssystem zu berechnen, auch wenn die Komponenten mittels nichtkonservativer Knoten miteinander verbunden sind, wie dies bei VHDL-AMS der Fall ist. Eine getrennte Berechnung wie z. B. bei dem Blocksimulator des „Simplorer“ oder in SystemC-AMS ist erheblich schneller als die Berechnung eines gemeinsamen Gleichungssystems, da die benötigte Zeit zum Lösen eines Gleichungssystems nicht linear mit dessen Größe ansteigt. So erfordern z. B. 100 Gleichungssysteme à 10 Gleichungen erheblich weniger Rechenleistung als ein Gleichungssystem mit 1000 Gleichungen. Bei solchen Simulatoren wäre durch die Anwendung der MAM für analoge Systeme mit einer Simulationszeitverlängerung zu rechnen.

5 Multi-Architecture-Modellierung unter SystemC-AMS

Dieses Kapitel soll die Übertragbarkeit der MAM unter VHDL-AMS auch auf andere Beschreibungssprachen demonstrieren. Dazu kommt exemplarisch die Systembeschreibungssprache SystemC bzw. SystemC-AMS zum Einsatz.

5.1 Kurze Einführung zu SystemC und SystemC-AMS

5.1.1 Modellierung mit SystemC

Der bis heute anhaltende Trend zu immer komplexeren Hardware/Software-Systemen bei immer kürzeren Produktentwicklungszyklen (*time to market*) erfordert neue Konzepte im Entwurf solcher Systeme. Eines dieser neuen Konzepte ist die Systembeschreibungssprache SystemC. Wie bereits in Abschnitt 1.4.2, S. 31 beschrieben, beruht SystemC auf einer Referenz-Implementierung der Firma Synopsys Inc. und wird im Rahmen der OSCI weiterentwickelt. Im Gegensatz zu klassischen Hardwarebeschreibungssprachen wie VHDL oder Verilog stellt SystemC im eigentlichen Sinne keine eigene Sprache dar. SystemC ist eine Klassenbibliothek für C++ und stellt die für die Hardwarespezifikation und -modellierung notwendigen Konstrukte für Parallelität, Reaktivität und Zeitverhalten bereit. Nach [105] besitzt SystemC die folgende geschichtete (*layered*) Sprachархitektur:

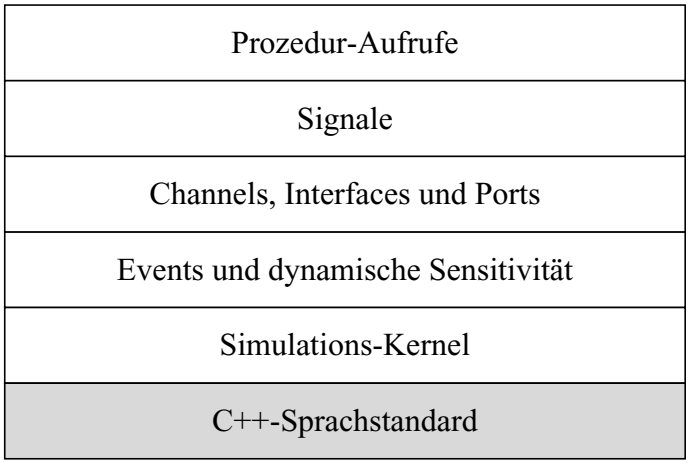


Bild 5-1 Spracharchitektur von SystemC [105]

Aufbauend auf dem C++-Sprachstandard, bildet der Simulationskernel die Grundschicht der Spracharchitektur. Die nächste Ebene steuert die digitalen Ereignisse. Der Datenaustausch zwischen Modellen erfolgt über sog. Kanäle (Channels). Zusammen mit den Interfaces und Ports ermöglicht diese Ebene die sogenannten *Interface Methode Calls* – IMC (Details dazu folgen später). Die obersten Schichten bilden Signale und *Remote Procedure Calls* – RPC. Diese dienen der eigentlichen Beschreibung des Verhaltens.

SystemC erlaubt – wie schon VHDL – die Beschreibung des Systemverhaltens auf unterschiedlichen Abstraktionsebenen. Durch abstraktere Datentypen, z. B. vordefinierte fixed-point Typen, können abstraktere Beschreibungen vorgenommen werden als in VHDL. Die Verwendung von

C++ zur Hardwaremodellierung erleichtert es wesentlich, die Hardwarekomponenten gemeinsam mit in C++ geschriebenen Softwarekomponenten zu simulieren und damit zu verifizieren.

Da SystemC auf C++ beruht, werden zur Modellerstellung und Simulation keine speziellen Werkzeuge benötigt. Wie in Bild 5-2 zu sehen ist, wird ausgehend vom Modell, den C++-Bibliotheken und der SystemC-Bibliothek mit Hilfe eines C++-Compilers ein ausführbares Programm erzeugt.

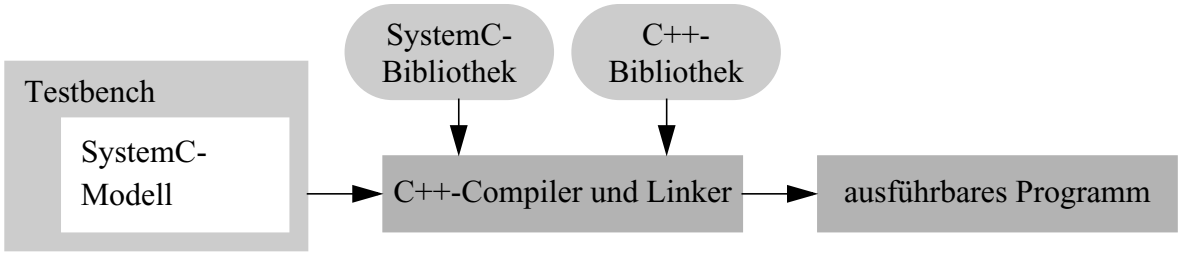


Bild 5-2 Erstellen eines ausführbaren Modells unter SystemC

Das entstandene Programm wird anschließend ausgeführt, die Simulationsresultate lassen sich in Form von Textausgaben oder Signallisten auswerten.

5.1.2 Aufbau eines SystemC-Modells

Die Grundstruktur eines SystemC-Modells ist das Modul (`SC_MODULE`). Dieses nimmt, ähnlich der Entity und der Architecture in VHDL, die Modellschnittstelle sowie das Modellverhalten auf und dient dem Aufbau hierarchischer Modelle durch Anlegen von Instanzen weiterer Module. Ein Modul entspricht einer Klasse in C++. Die Module kommunizieren über Ports mit ihrer Umgebung. Innerhalb der Module kann man lokale Signale und Variablen deklarieren. Die eigentliche Funktionsbeschreibung erfolgt innerhalb von Prozessen, die im jeweiligen Modul instantiiert werden.

Ports definieren die Schnittstelle des Modells und dienen der Kommunikation mit der Umgebung. Man kann sie als Eingang (`sc_in`), Ausgang (`sc_out`) oder als Ein- und Ausgang (`sc_inout`) deklarieren. Dabei wird ihnen ein Datentyp zugewiesen. Die Verbindung von Ports untereinander oder die Kommunikation zwischen Prozessen erfolgt mit Signalen, denen ebenfalls ein Datentyp zugewiesen wird.

Als Datentypen stehen dem Anwender zum einen die in C++ definierten Datentypen wie z. B.

- `bool` (true, false),
- `integer` (Ganzzahlformat -2^{31} bis $+2^{31}-1$ bei 32 Bit Systemen) oder
- `double` (Gleitkommaformat $-1{,}7\ 10^{308}$ bis $+1{,}7\ 10^{308}$)

zur Verfügung. Darüber hinaus definiert SystemC weitere Datentypen wie z. B.

- `sc_bit` (`'0'`, `'1'`),
- `sc_logic` (`'0'`, `'1'`, `'X'`, `'Z'`),
- `sc_int` (1 Bit bis 64 Bit Integer, Zweierkomplementdarstellung),
- `sc_uint` (1 Bit bis 64 Bit Integer, vorzeichenlos),

- `sc_fixed` (nutzerdefinierbares Festkommaformat in Zweierkomplementdarstellung),
- `sc_ufixed` (vorzeichenloses nutzerdefinierbares Festkommaformat),
- `sc_bv` (Vektor aus `sc_bit`, ähnlich dem `bit_vector` aus VHDL),
- `sc_lv` (Vektor aus `sc_logic`).

Zusätzlich hat der Nutzer die Möglichkeit, mit den sprachlichen Konstrukten von C++ weitere Datentypen zu definieren.

Das Verhalten der Komponente wird in Form von Prozessen beschieben. Der SystemC-Kernel arbeitet die Prozesse wie in VHDL quasiparallel ab. In SystemC stehen drei Arten von Prozessen zur Verfügung:

- Methods (`SC_METHOD`)
- Threads (`SC_THREAD`)
- getaktete Threads (`SC_CTHREAD`)

Methods und Threads lassen sich durch Ereignisse auf Signalen einer Sensitivity-Liste starten. Methods werden nach ihrer Aktivierung bis zum Ende abgearbeitet, während sich Threads stoppen lassen und ihre Abarbeitung nach einem weiteren *event* fortsetzen. Getaktete Threads sind eine Sonderform der Threads und dienen der Modellierung getakteter, synchroner Systeme.

Für weitere Details der Sprache SystemC sei auf [105], [106] und [107] verwiesen.

5.1.3 Konzept von SystemC-AMS

Mit SystemC steht eine leistungsfähige Beschreibungsform für digitale Hardware/Software-Systeme zur Verfügung. Technische Systeme bestehen jedoch oft nicht nur aus digitalen Komponenten, sondern häufig auch aus analogen Schaltungen oder nichtelektrischen Komponenten. Um solche heterogenen Systeme auf der Basis von SystemC beschreiben zu können, ist eine Erweiterung um semantische Elemente zur Beschreibung und Simulation analogen Verhaltens notwendig. Zu diesem Zweck hat sich im Jahr 2002 die internationale SystemC-AMS Study Group gegründet, die aus Repräsentanten verschiedener Forschungseinrichtungen und Industrieunternehmen besteht und der der Autor ebenfalls angehört.

SystemC-AMS ist wie bereits SystemC eine Klassenbibliothek für C++ und kann als Obermenge von SystemC angesehen werden, was eine einheitliche Beschreibung heterogener Systeme ermöglicht.

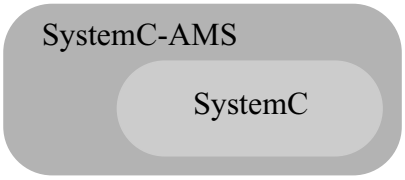


Bild 5-3 SystemC-AMS als Obermenge von SystemC

Die sich bietenden Möglichkeiten einer solchen Beschreibungssprache erkannten mehrere Wissenschaftler, so daß eine gleichzeitige Entwicklung mehrerer Implementierungen von analogen

Erweiterungen für SystemC erfolgte. Die Implementierungen unterscheiden sich dabei in Umfang und Art der Systembeschreibungsmöglichkeiten, je nach zu Grunde liegenden Anwendungsfällen. So ist die Implementierung des Fraunhofer-Instituts für Integrierte Schaltungen (FhG IIS/EAS) [108] eher für Kommunikationssysteme ausgelegt, während der Schwerpunkt der Implementierung der Universität Frankfurt a. M. [30] Anforderungen des Automotive-Sektors abdeckt. Auf dem „Forum on Specification & Design Languages“ wurde im September 2004 eine gemeinsame Implementierung verabschiedet, die zur Zeit aber noch nicht öffentlich zur Verfügung steht. Die im folgenden beschriebenen SystemC-AMS-Modelle beziehen sich daher auf die Klassenbibliothek MixSigC für SystemC-AMS in der Version 0.12, die vom Fraunhofer-Institut für Integrierte Schaltungen bereitgestellt wurde. Für spätere Versionen ist mit einer deutlichen Steigerung des Funktionsumfanges im Vergleich zu der im Rahmen dieser Arbeit verwendeten Version zu rechnen.

Die Erweiterungen für SystemC-AMS können als Schichtenmodell [109] auf der Basis des SystemC-Kernels angesehen werden (Bild 5-4). Die Synchronisationsschicht dient als Schnittstelle zwischen digitalen und analogen Berechnungsmodellen sowie zur Synchronisation der einzelnen analogen Solver untereinander. Die Solver-Schicht stellt verschiedene algorithmische Lösungsansätze (MOC) zur Berechnung analogen Verhaltens zur Verfügung. Es können dabei verschiedene Solver z. B. zur Berechnung von Signalflüssen oder zur Berechnung von linearen, nichtlinearen oder mechanischen Netzwerken eingebunden werden. Dabei ist die Spracharchitektur keinesfalls als abgeschlossen zu betrachten. Durch den geschichteten Aufbau der Spracharchitektur hat der Nutzer jederzeit die Möglichkeit eigene, nutzerspezifische Solver zu implementieren. Ebenso findet eine kontinuierliche Weiterentwicklung der Implementierungen der FhG IIS/EAS sowie der Universität Frankfurt statt.

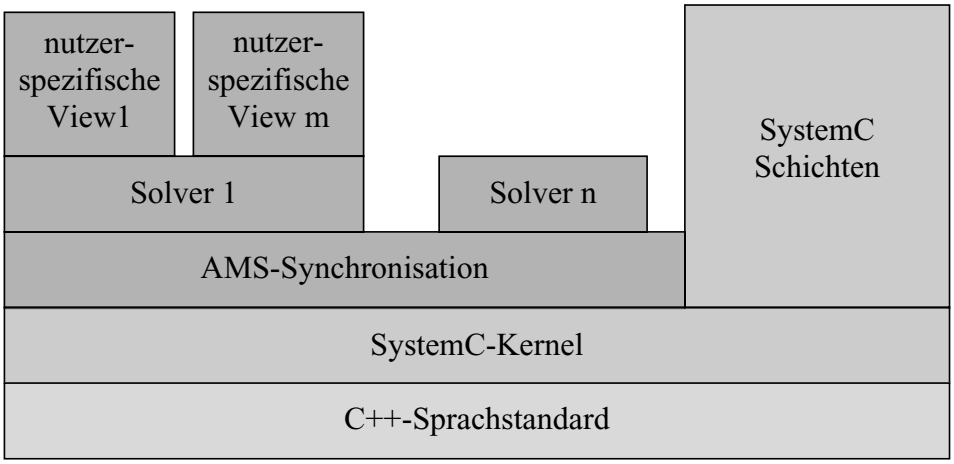


Bild 5-4 Schichtenmodell von SystemC-AMS nach Einwich [109]

Die oberste Schicht, der nutzerspezifische View, repräsentiert die unterschiedlichen Beschreibungsmöglichkeiten eines zeitkontinuierlichen Verhaltens auf der Basis von SystemC-AMS durch den Nutzer sowie die Ausgabe der Simulationsergebnisse. Mögliche Beschreibungsformen sind z. B. die Modellierung eines Sachverhaltes mittels Differentialgleichung oder als Übertragungsfunktion.

5.1.4 Sprachliche Erweiterungen von SystemC zu SystemC-AMS

Bei der vorliegenden Implementierung für SystemC-AMS wurde aufbauend auf dem Datenfluß-Berechnungsmodell von SystemC 2.0 ein statisches Datenfluß-Berechnungsmodell (SDF) eingeführt. Damit lassen sich bereits eine Vielzahl analoger, nichtkonservativer Systeme abstrakt beschreiben, sofern keine Rückwirkungen einer signalempfangenden Komponente auf die signalaussendende Komponente bestehen. Bei der vorliegenden Version sind allerdings keine unverzögerten Rückkopplungen erlaubt, was die Berechnung des Systems erheblich vereinfacht und damit beschleunigt, aber auch den Einsatzbereich einschränkt. Zur Berechnung konservativer Systeme in Form von linearen elektrischen Netzwerken ist ein weiterer *Solver* zur Simulation im Zeit- und Frequenzbereich (Transient- und AC-Simulation) implementiert. Da diese Version von SystemC-AMS primär für die Modellierung analoger Kommunikationselemente in Verbindung mit digitalen Hardware-/Softwaresystemen entwickelt wurde, ist bei der Beschreibung nichtelektrischer Strukturen mit Einschränkungen zu rechnen. So ist z. B. die Simulation nichtlinearer Netzwerke noch nicht möglich. Nichtelektrisches Verhalten kann durch ableiten der Bibliotheks-Elemente für elektrische Systeme nach der jeweiligen physikalischen Domäne direkt beschrieben werden, man ist jedoch auf die Analogien zu den elektrischen Elemente beschränkt. Die offene Struktur der SystemC-AMS Spracharchitektur, insbesondere auf Solver-Ebene, erlaubt es dem Nutzer jedoch, jederzeit eigene auf ein spezielles Problem angepaßte Solver zu integrieren und damit die Sprache, ausgehend vom aktuellen Entwicklungsstand, für eigene spezielle Anwendungsfälle zu erweitern. Der Nutzer erhält somit ein auf das jeweilige Problem angepaßtes und dennoch universelles Simulationswerkzeug, welches die Simulationsaufgabe mit hoher Performance lösen kann. Da der Quellcode von SystemC-AMS zu Beginn der Arbeiten nicht zur Verfügung stand, konnte diese Möglichkeit jedoch nicht weiter untersucht werden.

Die Struktur eines SystemC-AMS-Modells ist dabei mit der eines SystemC-Modells vergleichbar. Neben dem Modul `SC_MODULE` für digitales Verhalten werden mit SystemC-AMS zwei weitere Module eingeführt. Dies sind das `SDF_MODULE` für nichtkonservative Systeme und das `ELEC_MODULE` für konservative Systeme in Form elektrischer Schaltungen.

Für die Modellierung elektrischer Schaltungen im `ELEC_MODULE` stehen die folgenden Modelle zur Verfügung:

- Widerstand, Induktivität, Kapazität
- SDF-gesteuerter Widerstand
- spannungs- bzw. stromgesteuerte Spannungs- und Stromquellen
- SDF-gesteuerte Strom- und Spannungsquelle

Ein SDF-gesteuertes Bauelement kann man durch ein Signal des statischen Datenflusses steuern.

Das `ELEC_MODULE` besitzt Anschlüsse in Form konservativer Knoten mit Strom und Spannung. Diese werden mittels des SystemC-AMS-Konstrukts `elec_port` deklariert. Das `SDF_MODULE` besitzt nichtkonservative Schnittstellen, die sich mittels `sdf_inport` als Eingang bzw. `sdf_outport` als Ausgang konfigurieren lassen. Darüber hinaus sind Mischformen dieser Module möglich.

Verbunden werden die Module mit entsprechenden Signalen. Dabei dient

- das `SC_SIGNAL` der Verbindung digitaler Ports,
- das `SDF_SIGNAL` der Verbindung nichtkonservativer analoger Ports und
- der `ELEC_WIRE` der Verbindung konservativer Knoten.

Eine Sonderform für konservative Knoten stellt das Signal `ELEC2SDF` dar, mit dem einerseits konservative Ports verbunden werden können und mit dem man andererseits auch den aktuellen Spannungswert des Knotens an einen nichtkonservativen SDF-Knoten übergeben kann. Details zur Modellierung mit SystemC-AMS sind in [108] zu finden.

5.2 Direkte Anwendung der MAM unter SystemC-AMS

Auch bei der Modellierung unter SystemC-AMS kann das Hinterlegen verschieden abstrakter Architekturen für ein Modell dazu beitragen, die Genauigkeit und Simulationsgeschwindigkeit der Systemsimulation für den jeweils zu verifizierenden Sachverhalt zu optimieren. Wie schon bei der Modellierung mit VHDL-AMS tritt auch bei der Modellierung mit SystemC-AMS das Problem auf, daß unterschiedlich abstrakte Modelle unterschiedlich abstrakte Schnittstellen verwenden. Daher sollen auch unter SystemC die unter Abschnitt 3.2, S. 46 ff. dargelegten Ansätze zur Anwendung kommen. Dabei ist allerdings zu beachten, daß SystemC-AMS zum einen abstraktere Beschreibungen eines Verhaltens zuläßt und zum anderen die Simulationsgeschwindigkeit die von VHDL-AMS-Modellen z. T. deutlich übersteigt, wie in [110] und [20] gezeigt werden konnte.

5.2.1 Digitale Schnittstellen

Digitale Ports werden in SystemC mit `sc_in`, `sc_out` bzw. `sc_inout` plus Angabe eines Datentyps deklariert. Wie schon in VHDL, so bestimmt der Abstraktionsgrad des Modells den Datentyp der Schnittstelle. Als am wenigsten abstrakter Datentyp stellt SystemC den Typ `sc_logic` bzw. einen Vektor aus solchen Objekten (`sc_lv`) bereit. Im SystemC Users Guide [107] wird die Verwendung dieses Typs aus Simulationszeitgründen jedoch nur für Busse empfohlen, wogegen in VHDL die Verwendung der vergleichbaren neunwertigen Logik üblich ist. Für die MAM ergibt sich daraus, daß in SystemC der Datentyp `sc_logic` bzw. `sc_lv` ebenfalls nur dann als Datentyp für eine einheitliche Schnittstelle zu verwenden ist, wenn ein Bus modelliert werden soll. Für andere digitale Verbindungen kann man z. B. die Datentypen `sc_bit`, `sc_bv` oder `sc_int` usw. verwenden. Da Signale dieser Typen ohne den in Abschnitt 3.2.1 beschriebenen Verlust an Information in andere Datentypen konvertiert werden können, spielt die Wahl des Datentyps für einfache digitale Schnittstellen eine untergeordnete Rolle. Man muß lediglich zu Beginn des Entwurfsprozesses eine Entscheidung treffen. Für die Umwandlung der Datentypen stellt SystemC teilweise Funktionen oder cast-Operationen zur Verfügung. Ist dies nicht der Fall, müssen wie in VHDL passenden Konvertierungsfunktionen geschrieben werden. Quelltext 5-1 zeigt eine einfache Konvertierung eines 8 Bit breiten SystemC-Datentyps `sc_uint` (unsigned integer) in einen SystemC-Bitvektor `sc_bv`.

```
void uint2bv (sc_uint<8> *p_uint, sc_bv<8> *p_bv)
{
   for(int i=0; i<8; i++)
   {
      if ((*p_uint)[i] == 0)
         (*p_bv)[i]=(sc_bit)'0';
      else
         (*p_bv)[i]=(sc_bit)'1';
   }
   return;
}
```

Quelltext 5-1 Einfache Konvertierung von SystemC-Datentypen

5.2.2 Protokolle digitaler Schnittstellen

Die Beschreibung von Schnittstellen auf hohem Abstraktionsniveau gewinnt bei der Modellierung mit SystemC gegenüber VHDL an Bedeutung. SystemC bietet daher spezielle Mechanismen zur Beschreibung von Schnittstellen, worauf in Abschnitt 5.3 etwas näher eingegangen werden soll. Im folgenden wird die direkte Anwendung des in VHDL erprobten Ansatzes in SystemC demonstriert. Als Anschauungsobjekt soll auch hier die abstrakte RS232-UART dienen.

```
SC_MODULE(sender)
{
   …

   void send()
   {
      //Initialisieren der Datenübertragung
      RS232_init(&s_tmp,NULL);
      wait(10, SC_US);

      //Datenwort „0“ senden
      a=0;
      RS232_senddata(a, &s_tmp, NULL);
      wait();

      //Datenwort „2“ senden
      a=2;
      RS232_senddata(a, &s_tmp, NULL);
      wait();
      a=3;

      //Datenwort „3“ senden
      RS232_senddata(a, &s_tmp, NULL);
      wait();
   };

   //Konstruktor für Thread-Prozeß send, der durch clk gestartet wird
   SC_CTOR(sender)
   {
      SC_THREAD(send);
      sensitive_pos << clk;
      …
   }
};
```

Quelltext 5-2 Sequentieller Aufruf der RS232-Sendefunktion

Die Funktionen für das Senden und Empfangen der Daten sowie zur Initialisierung der Schnittstelle sind dabei, abgesehen von den SystemC-Signalen `sc_signal`, dem Datentyp `sc_bit` und den `wait`-Statements, in konventionellem ANSI-C++ beschrieben. Bei der Anwendung dieser Funktion ist zu beachten, daß eine solche C++-Funktion im Rahmen von SystemC nur sequentiell innerhalb eines Prozesses aufgerufen werden kann. Auf Grund der Anwendung des `wait`-Statements muß man dabei einen `SC_THREAD` verwenden. Den konkurrenten Aufruf von Prozeduren wie in VHDL gibt es in SystemC nicht. Sollen Sender oder Empfänger eine eigenständige Komponente darstellen, so müssen sie manuell in einen entsprechenden `SC_THREAD` eingefügt werden. Beispiele dafür zeigen Quelltext 5-2 und Quelltext 5-3. Der vollständige Quellcode zur abstrakten UART ist im Anhang C abgebildet.

```
SC_MODULE(receiver)
{
   sc_in<sc_bit> s;

   sc_signal<sc_bit> s_tmp;
   sc_signal<sc_uint<8> > b;

   //Verbinden des Eingangsports mit einem SC_SIGNAL aufgrund Restriktionen der Sprache SystemC
   void assign()
   {  s_tmp=s;}

   //Empfänger
   void rcv()
   {
      while(true)
      {
         RS232_receivedata(&b, &s_tmp, NULL);
      }
   };

   //Konstruktor für Thread-Prozeß rcv, der durch Ereignisse auf s_tmp gestartet wird und für
   //Methoden-Prozeß assign, der durch Ereignisse auf s gestartet wird
   SC_CTOR(receiver)
   {
      SC_THREAD(rcv);
      sensitive << s_tmp;

      SC_METHOD(assign);
      sensitive << s;
   }
};
```

Quelltext 5-3 Pseudo-konkurrenter Aufruf des RS232-Receivers

Da die manuelle Erstellung solcher Konvertierungsalgorithmen zeitaufwendig und fehleranfällig sein kann, sei eine Kombination mit der automatischen Interfacegenerierung mittels der SVE-Protokollbeschreibung [80] empfohlen. In Abschnitt 5.3.1 wird auf diese Methode noch etwas näher eingegangen.

5.2.3 Analoge elektrische und nichtelektrische Schnittstellen

5.2.3.1 Schnittstellenobjekte

Wie bereits in Abschnitt 3.2.3, S. 52 ff. dargelegt, sind im Rahmen der MAM für analoge Modelle, die bis auf Netzlistenniveau verfeinert werden sollen, konservative Knoten als Schnittstellen vorzusehen. Diese entsprechen in SystemC-AMS dem Port-Typ `elec_wire`. Die auf hoher Abstraktionsebene üblichen nichtkonservativen Knoten entsprechen in SystemC-AMS einem `sdf_inport` bzw. `sdf_outport`.

Während in VHDL-AMS syntaktisch nicht zwischen Modellen mit konservativen und nichtkonservativen Knoten unterschieden wird, ist dies in der vorliegenden SystemC-AMS-Version notwendig. In Tabelle 5-1 sind die Hauptunterscheidungsmerkmale aufgeführt.

Modell-Typ	konservative Knoten als Ports	nichtkonservative Knoten als Ports	Verhaltensbeschreibung
`ELEC_MODULE`	ja	nein	nein
`SDF_MODULE`	nein	ja	ja
`ELSDF_MODULE`	ja	ja	nein

Tabelle 5-1 Hauptunterscheidungsmerkmale der Modell-Typen in SystemC-AMS

Diese Eigenschaft von SystemC-AMS hat zur Folge, daß sich ohne Anwendung der MAM der Typ eines Ports praktisch nicht mehr nachträglich verfeinern läßt. Für die Modellierung mit MAM bedeutet dies, daß man das abstrakte Verhalten weiterhin in einem `SDF_MODULE` beschreiben muß, welches dann in einem `ELEC_MODULE` instantiiert wird.

Für die Verbindung nichtkonservativer Knoten mit konservativen Netzwerken entstanden daher für die jeweilige Anzahl an Ein- und Ausgängen angepaßte Koppelmodelle. Diese Modelle sind als C++-Template ausgeführt, so daß das zu instantiierende Modell mit nichtkonservativen Anschlüssen nur an das Interface-Modell übergeben werden muß. Quelltext 5-4 zeigt das Interface-Template für Modelle mit einem Ein- und einem Ausgang.

Für die Umwandlung eines nichtkonservativen Knotens in einen konservativen Knoten kommt im Interface-Modell eine SDF-gesteuerte Spannungsquelle zum Einsatz. Die Umwandlung eines konservativen Knotens in einen nichtkonservativen Knoten geschieht mittels des `elecv2sdf`-Objekts aus SystemC-AMS. Dieses Objekt ist eine Sonderform eines konservativen Knotens. Mit ihm können zum einen konservative Schnittstellen verbunden werden, und zum anderen kann ein nichtkonservativer Port lesend darauf zugreifen. Das Modell der spannungsgesteuerten Spannungsquelle (VCVS in Quelltext 5-4) ist notwendig, weil in der vorliegenden Version von SystemC-AMS Schnittstellen nicht die Eigenschaften des `elecv2sdf`-Objekts besitzen können, sondern ein solches Objekt nur als interner konservativer Knoten hierarchisch untergeordnete Modelle verbinden kann. Besitzt das abstrakte Modell nur Eingänge, so besteht auch die Möglichkeit, im hierarchisch übergeordneten Modell statt der konservativen Knoten ein solches `elecv2sdf`-Objekt zu verwen-

den. In diesem Fall können abstrakte Modelle weiter als `SDF_MODULE` beschrieben und ihre nichtkonservativen Knoten direkt mit den konservativen Knoten des Systemmodells verbunden werden. Allerdings muß man in der vorliegenden Version von SystemC-AMS ein `elecv2sdf`-Objekt stets mit einem SDF-Eingang verbinden, was das Anlegen einer Instanz eines Dummy-SDF-Objekts für jeden als `elecv2sdf` beschriebenen konservativen Knoten erfordert.

```
//MAM-Interface-Template für Modelle mit 1 Eingang und 1 Ausgang

template <class SC_modell>
ELEC_MODULE(MAM_IF_0101)
{
   elec_port ti1,to1;            //konservative Schnittstellen-Knoten

   sdf_signal<double> co01;      //interne Verbindungsleitungen
   elecv2sdf ci01;
   elec_gnd gnd;

   SC_modell *model;             //zu instantiierendes Modell
   VCVS *interface_i01;
   Vsdf *interface_o01;

   //Konstruktor
   ELEC_HTOR(MAM_IF_0101)
   {
      //konservativer Knoten -> nichtkonservativer Knoten
      interface_i01=new VCVS(1);
      interface_i01->np(ci01);
      interface_i01->nn(gnd);
      interface_i01->ncp(ti1);
      interface_i01->ncn(gnd);

      //eigentliches Modell
      model=new SC_modell("regel");
      model->inp1(ci01);
      model->out1(co01);
      model->out1.T(Sample_Time,SDF_SEC); //Sample_Time muß globaler Parameter sein

      //nichtkonservativer Knoten -> konservativer Knoten
      interface_o01=new Vsdf;
      interface_o01->a(to1);
      interface_o01->b(gnd);
      interface_o01->ctl(co01);
   }

   //Destruktor
   MAM_IF_0101::~MAM_IF_0101()
   {
      delete model;
      delete interface_i01;
      delete interface_o01;
   }
};
```

Quelltext 5-4 MAM-Interface-Template für Modelle mit einem Ein- und einem Ausgang

Das Anlegen einer Instanz eines SDF-Modelles über das MAM-Interface-Template zeigt Quelltext 5-5. Dabei handelt es sich um einen Auszug aus dem Modell eines Regelkreises, dessen Aufbau in Bild 5-5 dargestellt ist.

```
#include "regel.h"
…
int sc_main(int argc, char* argv[])
{
   elec_wire …,error,regel;
   …
   //Anlegen einer Instanz des Modells „regel“ als Parameter des Interface-Templates MAM_IF_0101
   MAM_IF_0101<regel> reg1;
   reg1.ti1(error);
   reg1.to1(regel);
   …

   return mixsigc_finish();
}
```

Quelltext 5-5 Anlegen einer Instanz eines SDF-Modells mit MAM-Interface-Template

Interface-Templates entstanden für Modelle mit:

- 0 Eingänge, 1 Ausgang
- 1 Eingang, 0 Ausgänge
- 1 Eingang, 1 Ausgang
- 2 Eingänge, 1 Ausgang

Andere Konfigurationen kann man leicht aus diesen entwickeln.

5.2.3.2 Mögliche Probleme mit unterschiedlichen MOC

In VHDL-AMS werden Verhaltensgleichungen stets in **ein** Gleichungssystem eingetragen, unabhängig davon, ob die unbekannten Größen Branch-Quantities eines Netzwerkes, freie oder Port-Quantities sind. Die vorliegende SystemC-AMS Version dagegen berechnet das Verhalten in SDF-Modulen per Zuweisungen entsprechend des Signalflusses. Unverzögerte Rückkopplungen (algebraische Schleifen) sind in der vorliegenden Version verboten. Das Verhalten in ELEC-Modulen berechnet der Simulationskernel durch Lösen linearer Gleichungssysteme. Da die Zuweisungen in SDF-Modulen wesentlich schneller ausgeführt werden können als das Lösen eines Gleichungssystems, läßt dies bei Anwendung der MAM eine deutliche Verlangsamung der Simulation erwarten.

Ebenfalls beachtet werden muß der Umstand, daß der Austausch des zu Grunde liegenden Berechnungsmodells die Simulationsergebnisse ändern kann, wie folgendes Beispiel zeigt:

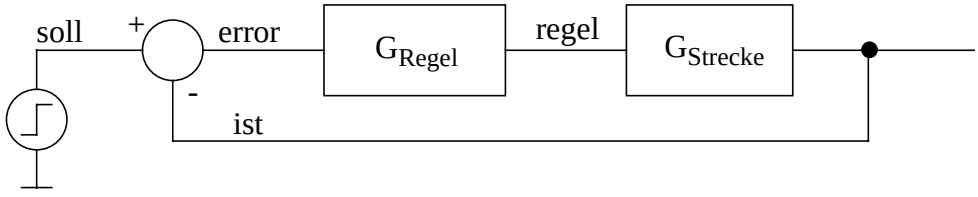


Bild 5-5 Testbeispiel Regler

Es handelt sich dabei um den bereits in Abschnitt 4.1.1 erwähnten PID-Regler. Für die Test-Simulationen unter SystemC-AMS wurde dieser jedoch zunächst auf einen P-Regler mit einer Verstärkung von G_{Regler} reduziert. Für die zu regelnde Strecke wurde eine vereinfachte Übertragungsfunktion ohne integrierendes oder differenzierendes Verhalten von $H(s)=G_{\text{Strecke}}$ angenommen. Löst man dieses System als Gleichungssystem ergibt sich:

$$error = soll - ist \quad \text{(GL 5-1)}$$

$$regel = error \cdot G_{Regler} \quad \text{(GL 5-2)}$$

$$ist = regel \cdot G_{Strecke} \quad \text{(GL 5-3)}$$

Durch entsprechendes Einsetzen erhält man:

$$ist = \frac{G_{Regler} \cdot G_{Strecke}}{1 + G_{Regler} \cdot G_{Strecke}} \cdot soll \quad \text{(GL 5-4)}$$

Setzt man z. B. $G_{\text{Regler}}=20$ und $G_{\text{Strecke}}=1$ ergibt dies für jeden Wert bzw. Werteverlauf von *soll* eine stabile Lösung.

Berechnet man das gleiche System modulweise mit um einen Zeitschritt verzögerter Rückführung, so erhält man den in Tabelle 5-2 angegebenen Werteverlauf. Die Regelung ist dieser Simulation nach instabil und beginnt als Reaktion auf einen Sprung am Eingang mit exponentiell steigender Amplitude zu schwingen.

Zeitschritt	soll	error	regel	ist
0	0	0	0	0
1	1	1	20	20
2	1	-19	-380	-380
3	1	381	7620	7620
4	1	-7619	-152380	-152380

Tabelle 5-2 Schrittweise Simulation des Reglers

Ein solches Verhalten liegt in der Natur von Signalflußalgorithmen, ist allgemein bekannt und darf nicht als Nachteil der vorliegenden SystemC-AMS Version betrachtet werden, da diese Version für andere Modellierungsaufgaben entwickelt wurde. Das in diesem Abschnitt betrachtete Beispiel soll lediglich die beim Wechsel des MOC möglicherweise auftretenden Probleme verdeutlichen.

5.2.3.3 Simulationsergebnisse

SystemC-AMS erzeugt kein einheitliches großes Gleichungssystem, sondern versucht Teilgleichungssysteme zu finden, die unabhängig voneinander sind. Dies ermöglicht eine deutlich schnellere Simulation, hat aber auch zur Folge, daß die Teilgleichungssysteme nacheinander berechnet werden. Dadurch erhält man ohne und mit Anwendung der MAM identische Ergebnisse, wie Tabelle 5-3 zeigt.

Simulationszeit [s]	Wert für „error“ ohne MAM	Wert für „error“ mit MAM
0.00099	0	0
0.001	1	1
0.00101	-19	-19
0.00102	381	381
0.00103	-7619	-7619

Tabelle 5-3 Simulationswerte für das Signal „error“ mit und ohne Anwendung der MAM

Dabei ergeben sich auf einer Sun UltraSparc-III mit 900 MHz folgende Simulationszeiten (CPU-Zeit):

Simulations-Endzeit	Rechenzeit ohne Anwendung der MAM	Rechenzeit mit Anwendung der MAM
10 ms	30 ms	40 ms
100 ms	210 ms	280 ms

Tabelle 5-4 Simulationszeiten für Regler mit und ohne MAM

In diesem Beispiel verlängert sich die Rechenzeit um ca. 30 %, wobei aus den zuvor genannten Gründen ein größerer Anstieg erwartet wurde. Das System ist zwar instabil, erreicht aber mit den vorgegebenen Simulationsparametern noch keine Begrenzung der Zahlendarstellung, so daß die Simulationszeiten als repräsentativ angesehen werden.

Da kein Einfluß der Reihenfolge des Anlegens der Instanzen der Komponentenmodelle beobachtet werden konnte, ist anzunehmen, daß der hier vorgestellte Ansatz auch in komplexeren Systemen keinen Einfluß auf das Simulationsergebnis hervorruft. Wenn allerdings in einer zukünftigen Implementierung von SystemC-AMS algebraische Schleifen zulässig sind, so ist vor der Anwendung der MAM zu überprüfen, ob sich die Simulationsergebnisse durch die Schnittstellengestaltung der MAM ändern.

5.2.4 Digital/analoge Schnittstellen

Für die Verbindung von digitalen Signalen im Komponentenmodell mit konservativen Knoten im Systemmodell wurden im Rahmen der MAM für VHDL-AMS zwei spezielle A/D- und D/A-Wandler (siehe Abschnitt 3.2.4, S. 54 ff.) entwickelt. Dementsprechend entstanden vergleichbare Wandler für SystemC-AMS sowie eine vereinfachte Form, die lediglich analoge SDF-Knoten auf digitale Signale und umgekehrt abbildet. Quelltext 5-6 zeigt den D/A-Wandler zum Anschluß eines SystemC-Signals an einen SDF-Knoten. Es ist zu erkennen, daß die Wandlung sehr einfach vorgenommen werden kann, wobei sich bei dieser einfachen Form der digitale Zustand `'Z'` nicht codieren und wiedererkennen läßt. Wie in VHDL ist der D/A-Wandler am Ausgang bzw. der

A/D-Wandler am Eingang der abstrakten Modelle zu instantiieren, wodurch der Austausch digitaler Modelle gegen analoge SDF-Modelle ermöglicht wird.

```
SDF_MODULE(MAM_outdrv_SDF)
{
   sc2sdf_inport<sc_logic> digital_in;      //digitaler Eingang
   sdf_outport<double> sdf_out;             //analoger Ausgang

   //Konstruktor für SDF_MODULE
   SDF_CTOR(MAM_outdrv_SDF);

   //Initialisierungs-Methode für SDF_MODULE
   void init()
   {
      sdf_out=0.0;
   }

   //eigentliche Verhaltensbeschreibung
   void sig_proc()
   {
      if (digital_in.read()=='1') sdf_out=U1;
      if (digital_in.read()=='0') sdf_out=U0;
      if (digital_in.read()=='X') sdf_out=Ux;
      if (digital_in.read()=='Z') sdf_out=U0;
   }
};
```

Quelltext 5-6 D/A-Wandler zum Anschluß eines SystemC-Signals an einen SDF-Knoten

Eine Besonderheit von SystemC-AMS sind die speziellen SDF-Ports `sc2sdf_inport` und `sdf2sc_outport`. Diese stellen die eigentliche Schnittstelle zwischen analogem und digitalem Simulator her, indem ausgehend von Zeitschritten des analogen Simulators digitale *events* erzeugt werden. Daraus ergibt sich, daß man in SystemC an dieser Stelle auch die abstrakteren digitalen Schnittstellen im Systemmodell verwenden kann. Abstrakte algorithmische Modelle werden somit konventionell mit digitalen Ports an die digitalen Signale des hierarchisch übergeordneten Modells angekoppelt. Zu analogen Signalflußmodellen verfeinerte Modelle lassen sich über `sdf2sc_outport` bzw. `sc2sdf_inport` Schnittstellen ebenfalls mit diesen Signalen verbinden.

Soll die Verfeinerung wie in Abschnitt 3.2.5, S. 60 darin bestehen, ein abstraktes analoges Modell zu einer digitalen Implementierung zu verfeinern, so lassen sich dafür ebenfalls die `sdf2sc_outport` und `sc2sdf_inport` Schnittstellen verwenden. Da diese Schnittstellen die Synchronisation zwischen Analog- und Digitalsimulator implizit enthalten, kann die in Abschnitt 3.2.5 beschriebene Enable-Leitung bei einem einfachen Datenaustausch entfallen. Soll ein bestimmtes *Handshake* zwischen den digitalen Komponenten realisiert werden, so sind die dafür notwendigen Steuerleitungen auf hoher Abstraktionsebene entsprechend der Enable-Leitung mit zu berücksichtigen. Zur Anpassung der Datentypen kommen vergleichbare Funktionen zum Einsatz, wie sie für VHDL-AMS beschrieben wurden. Mit der Übertragung der Daten über digitale Signale entfällt jedoch – wie schon unter VHDL-AMS – die Möglichkeit einer AC-Simulation der analogen Modelle.

Die für die Abbildung digitaler Signale auf konservative Knoten entstandenen speziellen A/D- und D/A-Konverter sind vergleichbar mit denen in VHDL und in der Lage, die *resolution function* des SystemC Datentyps `sc_logic` oder technologiespezifische Eigenschaften nachzubilden. Die Funktionalität des Ansatzes konnte – analog zu Bild 4-6 und Bild 4-7 – verifiziert werden (Bild 5-6). Der derzeitige Stand der Sprachsyntax erfordert die Modellierung der Wandler in mehreren Teilmodellen, was aber keinen Einfluß auf die Funktionalität hat. Der Quellcode ist im Anhang D abgebildet.

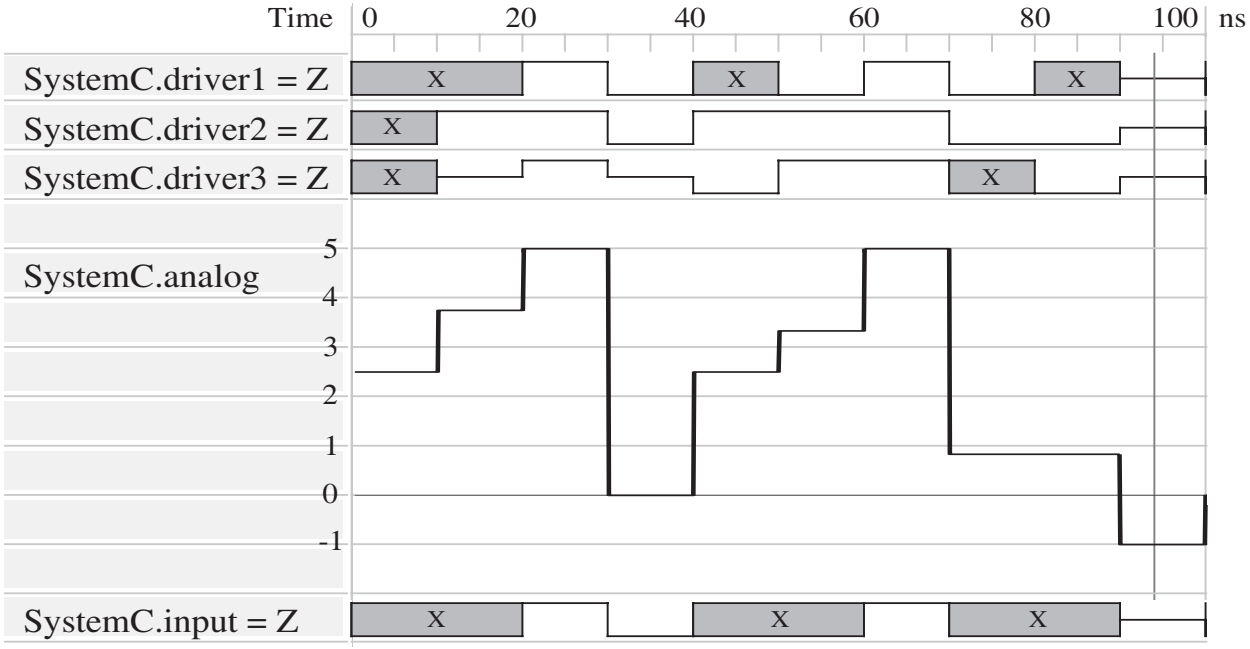


Bild 5-6 Übertragung digitaler Signale über konservative Knoten

Da der SystemC-Datentyp `sc_logic` nur 4 Zustände (`'1'`, `'0'`, `'X'` und `'Z'`) besitzt, ist die Wiedererkennung dieser aus dem analogen Signal gegenüber der neunwertigen Logik aus VHDL wesentlich einfacher und damit vollständig möglich.

5.2.5 Fazit

Die in den vorangegangenen Abschnitten beschriebenen Ansätze zur Schaffung vereinheitlichter Schnittstellen für Modelle auf unterschiedlichen Abstraktionsebenen arbeiten auf der obersten Schicht (*View)* der SystemC-AMS Struktur (vgl. Bild 5-4). Die Modell-Komponenten zur Umsetzung der MAM in SystemC-AMS werden wie in VHDL-AMS als Modell beschrieben. Dies hat den Vorteil, daß der Nutzer – ohne tiefergreifende Kenntnisse der Implementation von SystemC-AMS – die zur MAM gehörenden Komponenten verstehen und individuellen Erfordernissen anpassen kann. Bei der Umsetzung der MAM ist man dabei auf die von C++ und SystemC bzw. SystemC-AMS zur Verfügung gestellten syntaktischen Konstrukte angewiesen, wobei der Übergang vom Modellieren zum Programmieren fließend ist. Die prinzipielle Umsetzbarkeit der MAM mit diesen Konstrukten konnte an verschiedenen Beispielen gezeigt werden. Im Vergleich zu VHDL-AMS ergeben sich dabei einige Unterschiede, die die prinzipielle Umsetzbarkeit der MAM aber nicht einschränken.

Zusätzliche Möglichkeiten zur Umsetzung der Funktionalität der MAM bieten auch tiefere Schichten des SystemC-AMS Schichtenmodells (vgl. Bild 5-4). Dabei wäre es z. B. denkbar, analog zum `elecv2sdf` Objekt ein `sdf2elec` Objekt zu schaffen, das SDF-Ausgänge mit konservativen Knoten verbinden kann oder die Überarbeitung des `elecv2sdf` Objekts, so daß kein SDF-Port mit diesem Verbunden sein muß. Derartige Eingriffe in tiefere Schichten erfordern aber sehr umfangreiche Kenntnisse über die Implementierung der SystemC-AMS-Klassenbibliothek. Solche Implementierungen bergen dabei jedoch die Gefahr einer zur Originalversion der FhG IIS/EAS inkompatiblen Weiterentwicklung, so daß dieser Weg hier nicht verfolgt werden soll.

Die Anwendung von C-Präprozessoranweisungen, wie sie bereits in Abschnitt 1.6, S. 34 ff. erläutert wurden, bietet sich syntaktisch unter SystemC an. Die Nachteile dieser Methode wie geringe Übersichtlichkeit der Modell und höherer Modellierungsaufwand bleiben jedoch bestehen.

Die Anwendung objektorientierter Mechanismen zur Sicherstellung der Austauschbarkeit von Modellen über Hierarchieebenen hinweg wird im nächsten Abschnitt untersucht.

5.3 Verwendung objektorientierter Mechanismen

Bereits seit ca. 15 Jahren gibt es Ansätze zur Erweiterung von VHDL um objektorientierte Mechanismen („objective VHDL“, „OO-VHDL“, „VHDL++“) wie z. B. in [111], [112], [113], [114] [115] oder [116]. Neben diesen Erweiterungen von VHDL existiert eine Vielzahl weiterer objektorientierter Sprachkonzepte zur Spezifikation und Beschreibung von Hardware- bzw. Hardware/ Software-Systemen. Beispiele dafür sind die Sprachen „SDL“, „ESTELLE“, „LOTOS“, „CSP“ u. v. a. [117]. Doch erst mit SystemC konnte sich eine standardisierte objektorientierte Beschreibungssprache im Hardware- bzw. Systementwurf in größerem Umfang etablieren.

5.3.1 Digitale Schnittstellen und deren Protokolle

Ein wesentliches Merkmal objektorientierter Sprachen ist die Möglichkeit, Eigenschaften einer Klasse von Objekten auf abgeleitete Klassen von Objekten zu vererben. Genau diese Eigenschaft soll im folgenden ausgenutzt werden. Dazu erfolgt der Entwurf eines Models im Rahmen eines Top-Down-Entwurfes zunächst mit abstrakter Schnittstelle. Mit der Weiterentwicklung der Komponenten und des Systems werden von den abstrakten Modellen verfeinerte Modelle abgeleitet. Diese Modelle erhalten dadurch eine verfeinerte Schnittstelle und erben die abstrakte Schnittstelle und das abstrakte Verhalten.

Bei der Umsetzung dieses Ansatzes in SystemC ergibt sich aber zunächst das Problem, daß elementare Konstrukte von SystemC zur Unterstützung der Lesbarkeit des Modellcodes als sogenannte Makros implementiert sind. Bei diesen Makros wird ein bestimmter Umfang von C-Code hinter dem Makrobezeichner „versteckt“. Beim Kompilieren ersetzt dann ein Präprozessor das Makro gegen den C- bzw. C++-Code. Makros kann man aber nicht ableiten. Dadurch erschwert sich das Vorhaben, ein Modell aus einem anderen abzuleiten. SystemC bietet ab der Version 2.0 ein *workaround* für dieses Problem an, so daß diese Vorgehensweise getestet werden konnte. Allerdings hat sich dabei herausgestellt, daß man beim abgeleiteten Modell sowohl die verfeinerten, als auch die geerbten abstrakten Anschlüsse mit den entsprechenden benachbarten Modellen

verbinden muß. Damit entspricht diese Vorgehensweise dem bereits in Abschnitt 3.2, S. 46 ff. verworfenen Ansatz, sowohl abstrakte als auch detaillierte Schnittstelle zu hinterlegen. Somit erzeugt diese Vorgehensweise keine neuen Ansatzpunkte zur Unterstützung der MAM.

Die Kommunikation zwischen digitalen Komponenten auch auf unterschiedlichen Abstraktionsebenen, stellt jedoch eine elementare Grundfunktion im Entwurf komplexer digitaler Systeme dar, so daß SystemC für solche Belange spezielle Klassen, die Interfaces, Ports und Channels, zur Verfügung stellt [105]. Mit diesen Klassen werden spezielle Zugriffsmethoden, sog. *Interface Method Calls* (IMC), eingeführt. Das Interface definiert diese Zugriffsmethoden auf den Datenübertragungskanal, ohne diese zu implementieren. Der Channel wird vom Interface abgeleitet und stellt den Datenübertragungskanal dar. In diesem Channel erfolgt dann die Implementierung der IMC. Die Ports erlauben den Modulen und damit den darin enthaltenen Prozessen den Zugriff auf das Interface des Channels. Die IMC werden zwar im Channel implementiert, aber im Kontext des die IMC rufenden Prozesses ausgeführt.

Diese Vorgehensweise ermöglicht eine Mixed-Level-Simulation unterschiedlich abstrakter Modelle. In [105] wird dazu vorgeschlagen, basierend auf dem klassischen Top-Down-Entwurf zunächst die Modelle und Channels abstrakt zu beschreiben. Anschließend verfeinert man die Modelle und Channels. Noch nicht verfeinerte Modelle werden über Konverter an den verfeinerten Channel angeschlossen. Die prinzipiellen Nachteile eines solchen Ansatzes wurden in Abschnitt 3.4, S. 66 ff. bereits diskutiert. Mit der im Rahmen dieser Arbeit entstandenen Methodik, die abstrakte Schnittstelle bereits auf hoher Abstraktionsebene zu verwenden, läßt sich diese Vorgehensweise noch optimieren, was im folgenden anhand der bereits in Abschnitt 5.2.2 angewanden seriellen RS232-Datenübertragung demonstriert werden soll.

Im Gegensatz zu der in [105] beschriebenen Vorgehensweise erfolgt mit MAM die Beschreibung des Übertragungskanals zu Beginn der Modellierung so, daß der Channel IMC sowohl für abstrakte als auch für verfeinerte Zugriffe auf den Channel erhält. Die Definition des Interface und des Channel zeigt Quelltext 5-7 und Quelltext 5-8:

```
//abstraktes Interface
class RS232_abstract_if: virtual public sc_interface
{
   public:
      virtual void channel_init()=0;
      virtual void read_data(sc_signal<sc_uint<8> > *data)=0;
      virtual void write_data(sc_uint<8> *data)=0;
      virtual void write_data(int data)=0;
};

//verfeinertes Interface
class RS232_refined_if: public RS232_abstract_if
{
   public:
      virtual void read_bit(sc_bit *data)=0;
      virtual void write_bit(sc_bit *data)=0;
};
```

Quelltext 5-7 Interface-Definition für serielle RS232-Kommunikation

Der Channel wird dabei vom bereits verfeinerten Interface abgeleitet, so daß er sowohl abstrakte als auch verfeinerte IMC zur Verfügung stellen kann. Abstrakte Komponenten können nun über die IMC `read_data` und `write_data` Integer-Werte vom Channel lesen bzw. auf ihn schreiben. Verfeinerte Komponenten können den seriellen Datenstrom mittels der IMC `read_bit` und `write_bit` mit dem Channel austauschen. Für die Wandlung der abstrakten Daten in einen seriellen Datenstrom kommen dieselben Funktionen wie in Abschnitt 5.2.2 zum Einsatz, allerdings mit dem Unterschied, daß sie nun Teil des Channels sind und nicht mehr Teil des Modells, wodurch ein abstraktes Modell genau so beschrieben werden kann, als sollte es nur mit anderen abstrakten Modellen kommunizieren.

```
//Channel-Definition
#include "serial_transmit.h"

class RS232_connect: public sc_prim_channel, public RS232_refined_if
{
public:
   //Channeldaten
   sc_signal<sc_bit> channel_data;

   //Bearbeitung der events im Kanal
   virtual const sc_event& default_event() const
   {  return channel_data.default_event();}

   //IMC
   virtual void channel_init()
   {  RS232_init(&channel_data,NULL);}

   virtual void read_data(sc_signal<sc_uint<8> > *data)
   {  RS232_receivedata(data, &channel_data, NULL);}

   virtual void write_data(sc_uint<8> *data)
   {  RS232_senddata(*data, &channel_data, NULL);}

   virtual void write_data(int data)
   {  RS232_senddata((sc_uint<8>)data, &channel_data, NULL);}

   virtual void read_bit(sc_bit *data)
   {  *data=channel_data;}

   virtual void write_bit(sc_bit *data)
   {  channel_data=*data;}

   //Constructor
   RS232_connect::RS232_connect()
   {    channel_data=sc_bit('1');}
};
```

Quelltext 5-8 Kanaldefinition für RS232-Übertragung

Die so erstellten Channels lassen sich in Bibliotheken ablegen. In einem späteren Design, das die gleiche Schnittstelle verwendet, können die Channels ohne nennenswerten Mehraufwand wiederverwendet werden. Bezüglich der Komplexität der Protokolle gelten die schon bei VHDL-AMS aufgezeigten Grenzen.

Für komplexere (aber auch für einfache) Protokolle besteht die alternative Möglichkeit, den Entwurf mit MAM mit der in [80] vorgestellten Erweiterung von SystemC – der SVE-Protokollbeschreibung – zu kombinieren und solche Channels automatisch zu generieren.

SVE, auch bekannt als SystemC^{SV}, ist eine an der Professur für Schaltungs- und Systementwurf der TU Chemnitz entwickelte SystemC-Erweiterung, die unter anderem eine spezielle Modellierungsmöglichkeiten für Kommunikationsschnittstellen bietet [80]. Ausgehend von einer deklarativen Beschreibung der Protokollspezifikation erfolgt die Generierung kompositorischer und dekompositorischer Verhaltensbeschreibungen, die Sender und Empfänger des Protokolls repräsentieren. Diese generierten Verhaltensbeschreibungen entsprechen im Grunde den in die Channels verlagerten Protokoll-Abstraktionswandlern dieser Arbeit. Darüber hinaus enthält die Methode einen Simulationskernel zur Simulation solcher Protokollspezifikationen sowie die Möglichkeit, aus der Protokollspezifikationen synthetisierbaren SystemC-RTL-Code zu generieren. Dieser RTL-Code kann mittels geeigneter Werkzeuge nach VHDL übersetzt werden, so daß eine Kombination von SVE auch mit der MAM unter VHDL-AMS vorstellbar ist.

Für eine ausführlichere Betrachtung der SVE-Protokollbeschreibung und für Anwendungsbeispiele sei auf [80], [118], [119] und [120] verwiesen. Eine Anwendung auf ein Beispiel in dieser Arbeit würde deren Rahmen allerdings überschreiten.

5.3.2 Analoge Schnittstellen

Bei der Ableitung konservativer Netzwerkmodelle aus abstrakten Signalflußmodellen treten dieselben Probleme auf, wie sie schon für die Ableitung digitaler Modelle beschrieben wurden. Zusätzlich ist zu beachten, daß in der vorliegenden Implementierung von SystemC-AMS Signalfluß- und Netzwerkmodelle unterschiedliche Modultypen mit unterschiedlichen Möglichkeiten zur Beschreibung des Verhaltens benötigen. Daher soll dieser Ansatz nicht weiter verfolgt werden.

Alternativ wurden Möglichkeiten untersucht, Channels und deren IMC ähnlich denen der digitale Kommunikation bei der Verfeinerung analoger Modelle anzuwenden. Die dafür notwendigen Channels könnten die in Bild 5-7 dargestellte prinzipielle Struktur besitzen.

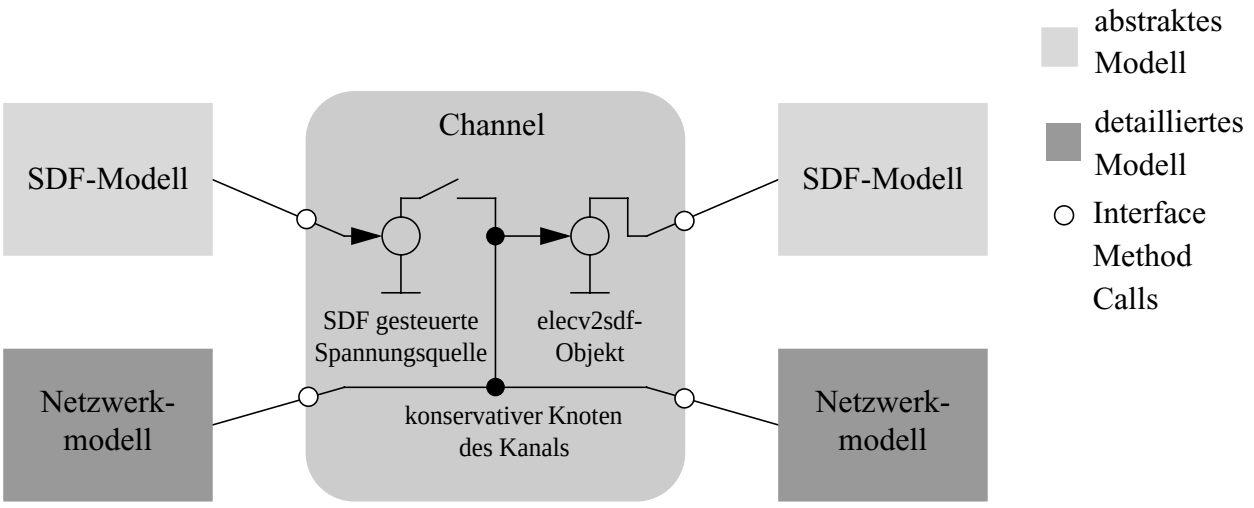


Bild 5-7 Theoretische Kanalstruktur für analoge Channels

Die Interface-Definition würde in SystemC-AMS dann wie folgt aussehen:

```
class analog_if: virtual public sc_interface
{
public:
   //Verbinden eines SDF-Signals mit dem SDF-Eingang des Channels
   virtual void write_sdf(sdf_signal<double> *data)=0;

   //Direktes Schreiben von Daten in den SDF-Eingang des Channels
   virtual void write_sdf(double data)=0;

   //Verbinden eines SDF-Signals mit dem SDF-Eingang des Channels
   virtual void read_sdf(sdf_signal<double> *data)=0;

   //Direktes Lesen von Daten aus dem SDF-Ausgang des Channels
   virtual void read_sdf(double *data)=0;

   //Verbinden eines konservativen Knotens mit dem konservativen Knoten des Channels
   virtual void connect_node(elec_wire t1)=0;
};
```

Quelltext 5-9 Pseudocode für analoges Channel-Interface

Es werden dabei Methoden für einen lesenden und schreibenden Zugriff auf die SDF-Signale im Channel bereitgestellt. Ebenso erfolgt die Bereitstellung einer Methode zur ungerichteten Verbindung von konservativen Knoten mit dem Channel-internen konservativen Knoten. Eine mögliche Implementierung des Channels ist in Quelltext 5-10 beschrieben.

Der Kanal enthält entsprechend Bild 5-7 ein internes SDF-Signal `data_in`, auf den die angeschlossenen abstrakten Modelle mittels der `write_sdf` Funktionen Daten schreiben können. Der interne konservative Knoten ist ein `elecv2sdf` Objekt, über den die `read_sdf` Funktionen den Spannungswert an abstrakte Modelle übermitteln können. Konservative Knoten verfeinerter Modelle können über die `connect` Methode mit dem internen konservativen Knoten verbunden werden.

Zur Umsetzung dieser analogen Channels wird eine SystemC-AMS-Version benötigt, die die folgenden Statements erlaubt:

- direktes Lesen oder Schreiben auf ein `sdf_signal`,
- explizite Anweisung zur Verbindung konservativer Knoten,
- Modell eines Schalters (evtl. durch SDF-gesteuerten Widerstand ersetzbar),
- Möglichkeit zum Aufruf der IMC im Konstruktor eines hierarchischen Modells.

Die Implementierung dieser, in der vorliegenden Version von SystemC-AMS nicht enthaltenen Funktionalität erfordert umfangreiche Arbeiten auf der Solver- und möglicherweise auch auf der AMS-Synchronisationsschicht (vgl. Bild 5-4). Da der Quellcode der verwendeten SystemC-AMS Implementierung nicht zur Verfügung steht, können leider keine Arbeiten diesbezüglich durchgeführt werden. Statt dessen seien für die Vereinheitlichung der analogen Schnittstelle die Template-Interfacemodelle aus Abschnitt 5.2.3 empfohlen.

```
class analog_channel: public elec_elements, public sdf_module, public analog_if
{
public:
   elecv2sdf channel_node;        //konservativer Knoten des Kanals
   elec_wire sdf_node;            //konservativer Zwischenknoten
   elec_gnd gnd;

   sdf_signal<double> data_in;
   sdf_signal<bool> switch_ctrl;

   Vsdf *mod1;
   Switch *mod2;

   ELSDF_CTOR(analog_channel)
   {
      switch_ctrl=true;

      //SDF-gesteuerte Spannungsquelle zur Verbindung des SDF-Eingangs mit dem Zwischenknoten
      mod1=new Vsdf;
      mod1->a(sdf_node);
      mod1->b(gnd);
      mod1->ctl(data_in);

      //Schalter zum Verbinden des Zwischenknotens mit dem Knoten des Kanals
      mod2=new Switch;
      mod2->a(sdf_node);
      mod2->b(channel_node);
      mod2->ctl(switch_ctrl);
   }

   //Zugriffsfunktionen (IMC)
   virtual void write_sdf(sdf_signal<double> *data)
   {
      if (switch_ctrl)
         data_in=*data;
      else
         cout<<“no parallel use of write_sdf and connect_node possible“
   }

   virtual void write_sdf(double data)
   {
      if (switch_ctrl)
         data_in.write(data);
      else
         cout<<“no parallel use of write_sdf and connect_node possible“
   }

   virtual void read_sdf(sdf_signal<double> *data)
   {  *data=channel_node;}

   virtual void read_sdf(double *data)
   {  *data=channel_node.read();}

   virtual void connect_node(elec_wire t1)
   {
      channel_node.connect(t1);
      switch_ctrl=false;
   }
};
```

Quelltext 5-10 Pseudocode für analogen Channel

5.3.3 Analog/Digitale Schnittstellen

Im folgenden soll untersucht werden, welche Funktionalität ein Channel zur Verbindung abstrakter analoger Modelle mit digitalen RT-Modellen aufweisen muß. Die Hauptnachteile der bisher für diesen Fall diskutierten Varianten für VHDL-AMS und SystemC-AMS bestehen darin, daß entweder die analogen abstrakten Modelle nicht einer AC-Simulation unterzogen werden können, bzw. bei der Kommunikation zwischen digitalen Modellen Signal-Verzögerungen entstehen. Als Ausweg daraus sind sowohl analoge als auch digitale Verbindungen bereitzustellen. Für VHDL-AMS wurde ein solcher Ansatz verworfen, da er den Modellierungsaufwand deutlich erhöht. SystemC bietet mit den IMCs und Channels die Möglichkeit, die zur Überführung der Daten über die Abstraktionsebenen hinweg notwendigen Verhaltensbeschreibungen in den Channels zu „verstekken“. Dadurch entfällt jeglicher Modellierungs-Mehraufwand in den Modellen. Eine mögliche Beschreibung eines solchen Channels ist in Bild 5-8 dargestellt.

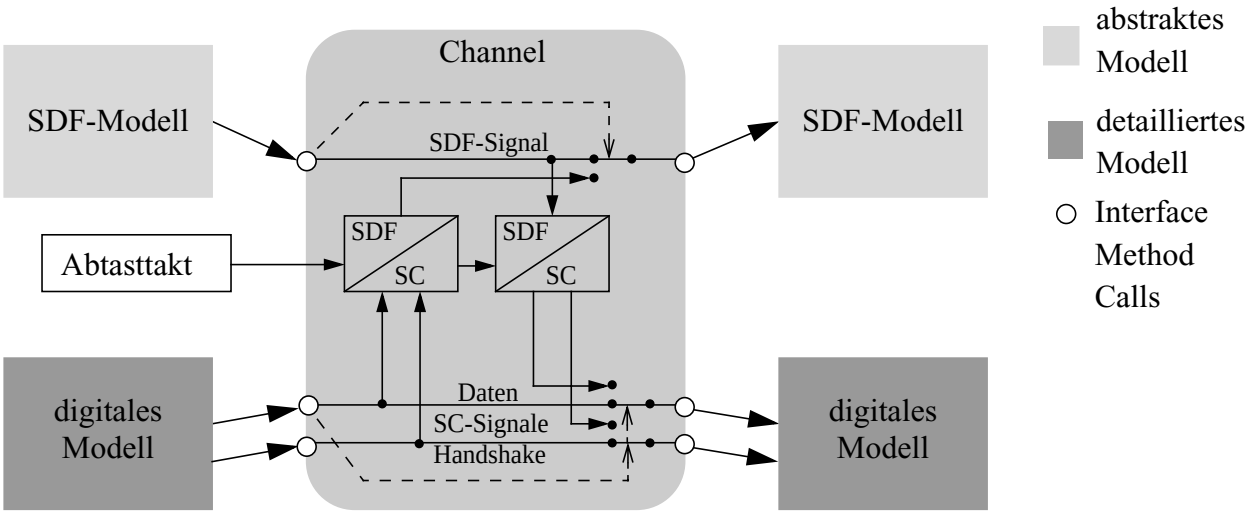


Bild 5-8 Theoretische Kanalstruktur für analog/digitale Channels

Der Channel enthält sowohl ein analoges SDF-Signal als auch digitale SC-Signale für Daten- und *Handshake*-Leitungen. Damit können zum einen analoge Modelle über die entsprechenden IMC mit dem Kanal verbunden werden, ohne die Möglichkeit der AC-Simulation zu verlieren. Zum anderen lassen sich digitale Komponenten genau so miteinander verbinden, wie dies ohne MAM der Fall wäre. Benutzt man den IMC für den analogen Eingang des Channels, so werden mit Hilfe der Konverter auch die digitalen Ausgänge des Channels angesteuert. Erfolgt im Gegensatz dazu die Ansteuerung der Channels über den digitalen IMC, so generiert ein Konverter die Daten für den analogen Ausgang. Der Aufbau der Konverter entspricht dem Aufbau der Konverter, die für VHDL-AMS implementiert wurden. Sie übersetzen z. B. ein SDF-Signal vom Typ `double` in ein SC-Signal vom Typ `sc_int` und generieren notwendige *Handshake*-Signale oder umgekehrt.

Das Ziel der hier beschriebenen Vorgehensweise ist mit dem Ziel der in Abschnitt 5.4 beschriebenen Methode vergleichbar. Mit der Vereinheitlichung der SystemC-AMS Sprachsyntax und -semantik ist zu erwarten, daß auch Elemente der in Abschnitt 5.4 beschriebenen SystemC-AMS-Implementierung in die gemeinsame SystemC-AMS Version einfließen. Darüber hinaus gelten

auch hier die in Abschnitt 5.3.2 genannten Anforderungen bezüglich erweiterter Statements der SystemC-AMS-Klassenbibliothek. Daher soll hier von einer konkreten Implementierung der analog/digitale Channels abgesehen werden.

5.3.4 Fazit

Die direkte Ableitung verfeinerter Modelle aus abstrakten Modellen zur Vereinheitlichung der Schnittstelle hat sich nicht bewährt. Sehr gut anwendbar dagegen ist die in SystemC vorgesehene Nutzung von Channels mit *Interface Method Calls* bei digitalen Modellen. Implementiert man dabei den Channel so, daß er sowohl abstrakte als auch verfeinerte IMC unterstützt, so können im Verlauf der Modellentwicklung sowohl abstrakte als auch verfeinerte Modelle ohne Adapter an den Channel angeschlossen werden. Die Verwendung von Channels bei analogen Modellen zur Verbindung von Modellen auf unterschiedlichen Abstraktionsebenen, wie sie im Rahmen dieser Arbeit vorgeschlagen wird, unterstützt die vorliegenden Version von SystemC-AMS leider noch nicht. Die Implementierung derartiger Channels ist aber bereits Gegenstand der Forschung an der Universität Frankfurt (s. auch Abschnitt 5.4), so daß in absehbarer Zeit mit einer Unterstützung analoger Channels durch SystemC-AMS zu rechnen ist.

5.4 Mixed-Level-Simulation mit „ASC-Bibliothek“

Parallel zu der im Rahmen dieser Arbeit verwendeten SystemC-AMS-Bibliothek „MixSig-C“ der FhG IIS/EAS Dresden entstand an der Universität Frankfurt eine Implementierung unter der Bezeichnung ASC-Bibliothek (Analog SystemC). Im Jahr 2002 wurde in [30] basierend auf dieser SystemC-AMS-Implementierung eine Methode zur Mixed-Level-Simulation vorgestellt, auf die im folgenden kurz eingegangen werden soll.

Die in [30] vorgestellte Methodik orientiert sich an Systemen, die primär im Automotive-Sektor eingesetzt werden, wobei der Schwerpunkt auf kontrollflußorientierten Automaten und datenflußorientierten Steuer- und Regelsystemen liegt. Elektrische Netzwerke mit konservativen Knoten werden nicht berücksichtigt. Die unterstützten Entwurfsschritte umfassen die Verfeinerung von mathematischen Modellen auf analoge oder digitale Implementierungen. Die Schaffung der Möglichkeit zur Mixed-Level-Simulation wird dabei genau so betont wie im Rahmen dieser Arbeit.

Die grundlegende Idee der ASC-Design-Bibliothek besteht im Gegensatz zur MAM darin, die abstrakten Schnittstelle auf allen Abstraktionsebenen zu verwenden. Dabei enthält diese Schnittstelle neben den eigentlichen Anschlüssen zur Datenübertragung weitere Anschlüsse zur Schrittweiten- und Ausführungssteuerung. Als Übertragungskanal wird ein SystemC-Channel mit der Bezeichnung `asc_signal` eingeführt, der mit dem `SDF_SIGNAL` der Implementierung der FhG IIS/EAS Dresden [108] vergleichbar ist, jedoch erweiterte Zugriffsmöglichkeiten (IMC) insbesondere auf die *events* des Kanals bietet.

Die im Laufe des Entwurfsprozesses verfeinerten Schnittstellen können durch Überladen der Methoden des `asc_signal` durch entsprechende Konvertierungsmethoden modelliert werden. Für Signale, die während des Entwurfsprozeß neu hinzukommen, werden Adapter eingeführt. Diese benutzen zum einen das vorhandenen Interface, zum anderen stellen sie die zusätzlich not-

wendigen Anschlüsse bereit. Die sich daraus ergebende Struktur der Schnittstellen bei der Verfeinerung ist in Bild 5-9 dargestellt.

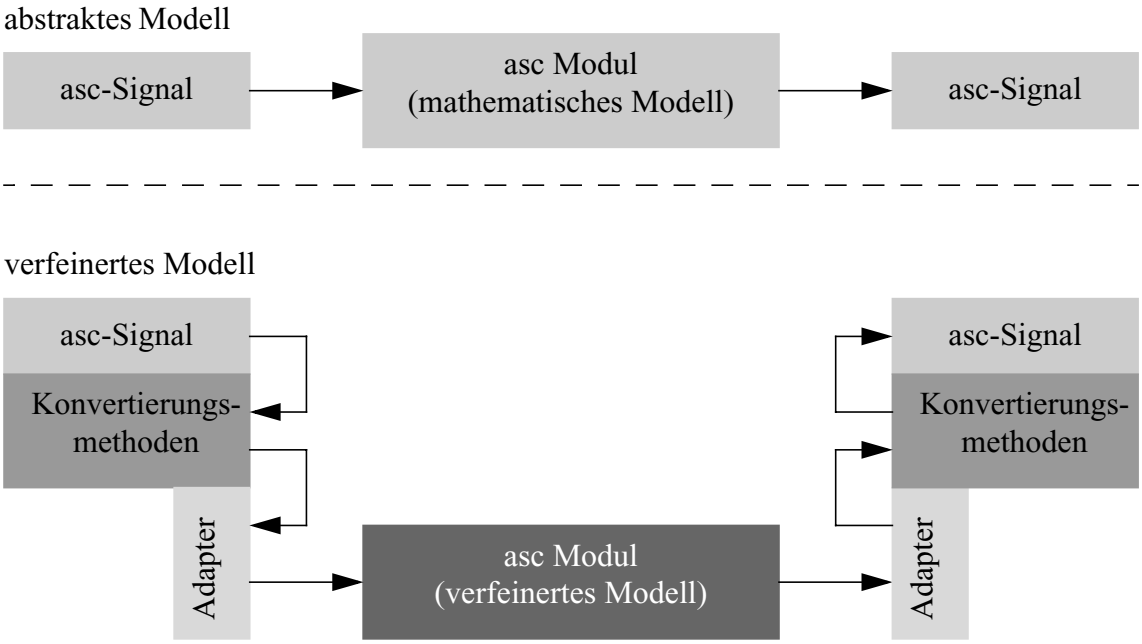


Bild 5-9 Verfeinerung der Schnittstelle nach Grimm [30]

Die in [30] vorgestellte Methode stellt einen leistungsfähigen Ansatz zur Mixed-Level-Simulation analog/digital gemischter Systeme dar, der dank der Eigenschaften der objektorientierten Sprache SystemC einige mit der MAM einhergehende Probleme, wie die Notwendigkeit der Bekanntheit der detaillierten Schnittstelle auf hohem Abstraktionsniveau, umgeht. Allerdings läßt sich der Ansatz dadurch nur für objektorientierte Beschreibungsformen, wenn auch nicht ausschließlich auf SystemC beschränkt, anwenden, während die MAM auf verschiedene Hardwarebeschreibungssprachen übertragbar ist. Die beschriebene Vorgehensweise setzt auf der Solver- und Synchronisationsschicht von SystemC-AMS (vgl. Bild 5-4) an, während die MAM ausschließlich auf der dem Nutzer einfacher zugänglichen nutzerspezifischen Schicht beruht. Darüber hinaus berücksichtigt die MAM auch konservative Systeme mit konservativen Knoten, während diese Gruppe von Modellen bei der asc-Bibliothek ausgeklammert wird.

Mit der Zusammenführung der SystemC-AMS Implementierung der Universität Frankfurt mit der im Rahmen dieser Arbeit verwendeten Implementierung der FhG IIS/EAS sind neue interessante Ansatzpunkte für eine Weiterentwicklung der MAM unter SystemC-AMS zu erwarten.

6 Kostenmodellierung auf Basis der Multi-Architecture-Modellierung

6.1 Motivation

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, da diese Parameter neben den funktionalen Daten wichtige Kenngrößen eines Systems darstellen. Dabei sind unter dem Begriff Kosten nicht nur fiskalische Kosten zu verstehen. Vielmehr wird der Begriff Kosten hier im Sinne der Terminologie der mathematischen Optimierung verwendet und repräsentiert das Ergebnis einer zu optimierenden Zielfunktion. Mit dieser Zielfunktion sollen Größen wie z B. Latenz, Datenrate, Leistungsaufnahme, Chipfläche, Fehlertoleranz, Testbarkeit, Speicherverbrauch, Herstellungskosten u. a., gewichtet bewertet werden [121]. Erste Ansätze dazu sind z. B. in [122] dokumentiert.

Bisher werden vorrangig Tools verwendet, die die Kostenparameter parallel zum Modell erfassen und vom Modell getrennt auswerten und optimieren. Daraus ergeben sich jedoch zwei Nachteile:

- für die Optimierung des Verhaltens und für die Optimierung der Kosten werden zwei Werkzeuge mit ähnlicher Grundfunktion benötigt,
- bei Änderungen am Modell können die zugehörigen Kostenfaktoren u. U. vergessen werden und damit die Konsistenz der Daten verloren gehen.

Das Ziel der Arbeiten zur Kostenmodellierung war daher, einen Ansatz zu finden, der diese Kostenfaktoren auf funktionelle Bestandteile des Verhaltens des Modells abbildet und gleichzeitig mit der Simulation auswertet.

Bei der Kostenoptimierung auf hohem Abstraktionsniveau ergibt sich aber das Problem, daß die Bestimmung von Kostenwerten in diesem frühen Entwurfsstadium, wie z. B. in [123] und [124] für die Leistungsaufnahme gezeigt, sehr aufwendig, wenn nicht gar unmöglich sein kann. In diesen Fällen bietet der Ansatz der MAM eine zumindest teilweise Lösung dieses Problems, wie in Abschnitt 6.5 gezeigt wird.

6.2 Grundlagen der Kostenmodellierung

Die Modellierung der Kosten als Bestandteil des Verhaltens soll, wie die MAM, in der Hardwarebeschreibungssprache VHDL bzw. VHDL-AMS realisiert werden, da diese Sprache die Möglichkeit zur Modellierung und Simulation eines breiten Spektrums an heterogenen Systemen bietet.

Als Randbedingungen für die Entwicklung des methodischen Ansatzes sollen gelten:

- leichte Anwendbarkeit ohne große Einarbeitungszeiten,
- minimaler Modellierungs- und Berechnungsmehraufwand,
- Sicherstellung der Datenkonsistenz zwischen Modell und Kostenfaktoren.

Dieser Ansatz zur Kostenmodellierung für heterogenen Systeme unter VHDL-AMS wurde in [2] und [3] diskutiert.

6.2.1 Datenstrukturen

Für die Integration der Kostenfaktoren in das Modell müssen zuerst geeignete Datenstrukturen geschaffen werden, die die Informationen aufnehmen können. Der Datentyp `criterion` ist eine Aufzählung von Kostenfaktoren, die zu optimieren sind. Den Inhalt dieses Typs kann man dabei auf die im jeweiligen Entwurf erforderlichen Parameter anpassen. Der Typ `criterion_val` repräsentiert ein 2-Tupel aus einem Kostenwert und einer Kostengrenze für den jeweiligen Kostenfaktor. Die Kostengrenze stellt dabei einen Referenzwert dar, den der Kostenwert nicht überschreiten soll. Wenn ein bestimmter Kostenfaktor einer Komponente bei der Kostenoptimierung nicht berücksichtigt werden soll, kann man für die Kostengrenze den Wert `-1` einsetzen. Die Funktionen zur Kostenberechnung ignorieren dann das entsprechende 2-Tupel. Dies gilt ebenso, wenn für den Kostenwert `-1` angegeben wird, weil man z. B. die Größe des Kostenfaktors im aktuellen Entwurfsschritt noch nicht ermitteln kann.

```
--Kostenfaktoren
TYPE criterion IS (  develop_cost, product_cost, space, power, testability,
                     error_safety);

TYPE criterion_val IS RECORD
                        val:real;  --Kostenwert
                        lim:real;  --Kostengrenze
             END RECORD;

--Kostenvektor
TYPE criterion_vect IS ARRAY (criterion) OF criterion_val;
```

Quelltext 6-1 Datenstrukturen für Kostenfaktoren

Der Typ `criterion_vect` ist ein Vektor, der alle Kostenwerte und Kostengrenzen einer Komponente zusammenfaßt. Dieser Datentyp kann in VHDL auf Konstanten (`CONSTANT`) oder Signale (`SIGNAL`) angewendet werden. Die Benutzung von Konstanten hat den Vorteil, daß die Berechnungen zur Kostenmodellierung nur zu Beginn der Simulation einmal ausgeführt und die Simulationsgeschwindigkeit dadurch kaum beeinträchtigt wird. Die Kostenwerte können so auch per `GENERIC` parametrisiert werden, sie sind zur Simulationslaufzeit aber nicht mehr änderbar. Ist dies erforderlich, so läßt sich der Datentyp `criterion_vect` auf ein Signal anwenden, dessen Wert sich simulationsabhängig ändert. Ergibt sich der Kostenwert aus einer analogen Größe (`QUANTITY`) als Ergebnis der Berechnung eines *simultaneous statements* (Differentialgleichung), so ist dafür Sorge zu tragen, daß die Quantity bei der Zuweisung auf das Signal auch ein *event* auslöst (z. B. mittels des Attributs `'Above`). Durch die Verwendung eines Signals kann sich allerdings, je nachdem wie oft sich dessen Wert ändert, die Simulationsgeschwindigkeit verringern.

Für die getrennte Definition der Vektoren für Kostenwert und -grenze werden zusätzlich die Datentypen `criterion_limit_vect` und `criterion_val_vect` definiert.

6.2.2 Definition der Kostenparameter und Kostengrenzen im Modell

Neben der im vorigen Abschnitt beschriebenen Verwendung von Konstanten oder Signalen für die Kostenvektoren besteht weiterhin die Möglichkeit, die Werte für Kostenwert und -grenze gemeinsam oder in getrennten Definitionen zu setzen. Die gemeinsame Definition bietet sich an, wenn für ein Modell nur eine Architekturvariante zur Verfügung steht. Sind dagegen für eine Komponente mehrere Architekturvarianten verfügbar, so kann in der `ENTITY` die Definition der Grenzen erfolgen, die dann für alle Architekturalternativen gültig sind. In der jeweiligen `ARCHITECTURE` werden dann die für diese Variante gültigen Kostenwerte gesetzt.

6.2.3 Verbinden der Kostenvektoren über Hierarchieebenen

Im nächsten Schritt erfolgt die Zusammenfassung der Kostenvektoren der einzelnen Komponenten im übergeordneten Strukturmodell zu einem neuen Kostenvektor, der die Kosten z. B. einer Baugruppe repräsentiert. Dabei muß die Anzahl der Komponenten *n* dieser Struktur variabel sein und darf nicht durch die Funktion, die die Kostenvektoren zusammenfassen soll, eingeschränkt werden.

$$K_{j_{val,\,limit}} = \sum_{i=1}^{n} K_{i,\,j_{val,\,limit}} \qquad \forall j(criterion(j)) \qquad \text{(GL 6-1)}$$

Normale VHDL-Funktionen haben aber eine feste Anzahl von Parametern und sind somit nicht zur Zusammenfassung einer variablen Anzahl von Kostenvektoren geeignet. Als Lösung bieten sich hier die VHDL *resolution functions* an. Diese Funktionen dienen zur Auflösung von Konflikten, die entstehen, wenn mehrere Treiber auf ein *resolved signal* schreiben.

Die Kostenvektoren der einzelnen Komponenten werden über Signal-Ports mit dem Richtungsattribut `out` in die nächsthöhere Hierarchieebene übertragen und dort mit einem *resolved signal* vom Typ `criterion_resolve_vect` verbunden. Die für die Zusammenfassung der einzelnen Kostenvektoren notwendige spezielle *resolution function* addiert im einfachsten Fall die Kostenwerte und Kostengrenzen für alle Kostenfaktoren. Ist die Kostengrenze eines Kostenfaktors in einer Komponente auf `–1` gesetzt, so wird diese Grenze und der dazugehörige Kostenwert ignoriert. Betragen die Kostengrenzen für einen Kostenfaktor in allen Komponenten `-1`, so erhält der entsprechende Faktor im resultierenden Kostenvektor ebenfalls die Grenze `-1`. Die *resolution function* ermittelt ebenfalls, ob ein Kostenwert einer Komponente seine Kostengrenze übersteigt und gibt ggfs. eine Meldung aus. Dabei kann der Anwender festlegen, ob die Meldung ein Hinweis, eine Warnung oder eine Fehlermeldung sein soll.

Darüber hinaus wurden weitere Funktionen geschaffen, mit deren Hilfe sich der von der *resolution function* erzeugte Kostenvektor bearbeitet läßt. Mit diesen Funktionen ist es möglich, die Werte von Kostenfaktoren zu ändern bzw. die Kostengrenzen neu festzulegen.

Zusätzlich wird in [2] eine Erweiterung der Kostenmodellierung für den Fall beschrieben, daß die Anwendung dieser Methode auf bestehende Modelle erfolgen soll, bei denen eine Erweiterung der Schnittstelle um Kostenparameter nicht gewünscht ist. In diesem Fall werden die Kostenvektoren nicht über Ports übertragen, sondern in einem globalen Signal berechnet.

6.2.4 Bestimmung eines skalaren Gütemaßes

Die Optimierung der Kostenfaktoren erfordert im allgemeinen, aus den Kostenvektoren ein skalares Gütemaß *S* zu bestimmen [121].

$$S = \sum_{\forall j(criterion(j))} \frac{K_{j_{val}}}{K_{j_{lim}}} \cdot W_j \qquad \text{(GL 6-2)}$$

Dazu existiert im Rahmen der Methode der Kostenmodellierung ebenfalls eine Funktion, die auf der Basis der Kostenwerte K_{val}, der Kostengrenzen K_{lim} und eines Wichtungsvektors W einen skalaren Wert ermittelt, der ein Gütemaß für die Einhaltung der Kostengrenzen darstellt (Zielfunktion). Beträgt der Kostenwert oder die Kostengrenze eines Kostenfaktors `-1`, so ignoriert die Funktion diesen Faktor bei der Berechnung. Sind sowohl Kostenwert als auch -grenze `0`, so wird als Quotient `1` angenommen.

6.3 Anwendungsbeispiel

Die Anwendung dieser Methode war ursprünglich beim Entwurf des Vibrationsmeßsystems mit frequenzselektivem mikromechanischen Vibrationssensor-Array (siehe Abschnitt 4.2.1, S. 79) geplant. Da aber in den getesteten VHDL-AMS Simulatoren („AdvanceMS“, „hAMSter“, „SMASH“) verschiedene, zur Kostenmodellierung notwendige Statements noch nicht implementiert sind, ist eine Anwendung beim Entwurf heterogener Systeme noch nicht möglich. Deshalb soll hier die Anwendung der Kostenmodellierung an einem abstrakten System unter VHDL demonstriert werden.

6.3.1 Ausgangsdaten

Das abstrakte fiktive System soll folgende Struktur besitzen:

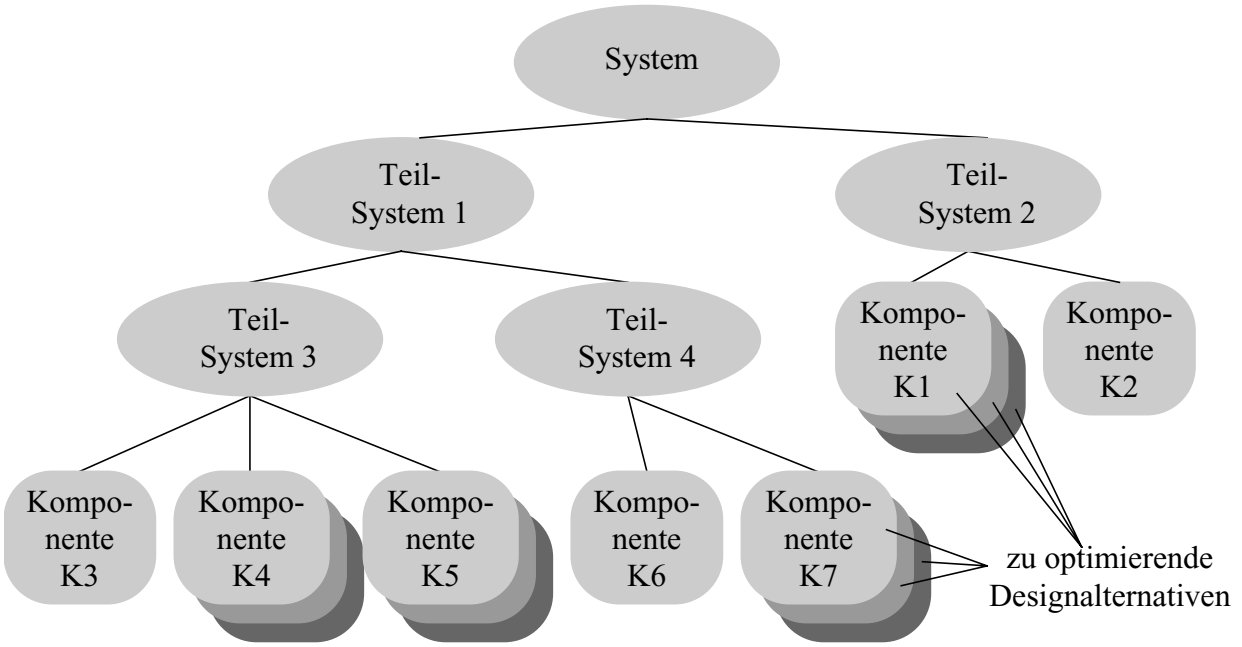


Bild 6-1 Struktur des Testsystems

Die zu optimierenden Kostenfaktoren (Kriterien) sind:

- Anschaffungskosten,
- Stromaufnahme,
- mittlere Ausfallzeit (MTBF),
- Platzbedarf.

Anschaffungskosten, Stromaufnahme und Platzbedarf der gesamten Struktur sollen dabei minimiert, die mittlere Ausfallzeit maximiert werden. Die gleichzeitige Maximierung und Minimierung von Kriterien als Ziel der Optimierung ist mit der aktuellen Version des vorgestellten Ansatzes nicht möglich. Daher muß man die mittlere Ausfallzeit durch Reziprokwertbildung in eine Ausfallwahrscheinlichkeit umwandeln, die dann minimiert werden kann. Tabelle 6-1 zeigt die für die einzelnen Komponenten des Systems angesetzten Grenzen und Tabelle 6-2 die Kostenwerte für die Komponenten 1 bis 4.

	K1	K2	K3	K4	K5	K6	K7
Anschaffungskosten [EURO]	100	50	80	5	1000	20	10
Stromaufnahme [mA]	10	5	100	-1	500	5	-1
Ausfallwahrscheinlichkeit [1/Std]	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Flächenbedarf [cm^2]	5	10	10	50	200	20	20

Tabelle 6-1 Kostengrenzen der einzelnen Komponenten

	K1 Variante 1	K1 Variante 2	K1 Variante 3	K2 Variante 1	K3 Variante 1
Anschaffungskosten	5	30	120	50	-1
Stromaufnahme	11	3	1	5	-1
Ausfallwahrscheinlichkeit	10^{-4}	10^{-6}	10^{-10}	10^{-4}	-1
Flächenbedarf	3	3	2	10	-1

Tabelle 6-2 Kostenfaktoren der Komponenten 1 bis 4

So stehen z. B. für Komponente K1 drei verschiedene Alternativen zur Verfügung, bei der die erste die Vorgabe bezüglich der Stromaufnahme überschreitet. Für die zweite und dritte Komponente steht jeweils nur eine Architektur zur Verfügung, wobei bei der dritten die Kostenwerte noch nicht feststehen. Für die weiteren Komponenten des Systems gelten ähnliche Vorgaben. Weiterhin sollen im Teilsystem 3 z. B. durch die Montage der Komponenten weitere Kosten hinzukommen, ohne daß sich die Kostengrenzen erhöhen.

6.3.2 Modellimplementierung

Die folgenden Modellausschnitte zeigen an Beispielen die Anwendung des Ansatzes beim Entwurf. Alle in Abschnitt 6.2 genannten Datentypen und Funktionen sind in einem Package „cost_support“ beschrieben, das bei Bedarf auf den jeweiligen Anwendungsfall (zu optimierende Kriterien, Reaktion auf Kostenüberschreitung) leicht angepaßt werden kann.

```
USE work.cost_support.ALL;
ENTITY Komponente1 IS
   --funktionale Generics des Modells
   …
   PORT( --Port für Kostenvektor
         cost:OUT criterion_vect:=(OTHERS=>(-1.0,-1.0));

         --funktionale Ports des Modells
         …
      );

   --Definition der Kostengrenzen
   CONSTANT cost_limit:criterion_limit_vect:=
           (Anschaffung    =>100.0,        --Euro
            Stromaufnahme  =>10.0,         --mA
            Fehlerrate     =>1.0e-4,       --1/Stunde
            Groesse        =>5.0);         --cm*cm
END;
```

Quelltext 6-2 Komponente 1, Entity mit Definition der Kostengrenzen

Bei der Anwendung der Kostenmodellierung in realen Modellen enthält die Portdeklaration neben dem Port `cost` die normalen funktionalen Ports des Modells. Der Port `cost` muß mit dem Defaultwert `-1` für Kostenwert und -grenze versehen werden, um falsche Kostenüberschreitungsmeldungen zu vermeiden.

```
ARCHITECTURE Version1 OF Komponente1 IS
   --Definition der Kostenwerte
   CONSTANT cost_val:criterion_val_vect:=
           (Anschaffung    =>5.0,
            Stromaufnahme  =>11.0,
            Fehlerrate     =>1.0e-4,
            Groesse        =>3.0);
BEGIN
   --Modellgleichungen
   ...
   --Zusammenfügen von Kostengrenze und -wert
   cost<=cost_limit & cost_val;
END;
```

Quelltext 6-3 Komponente 1, Architekturvariante 1 mit Definition ihrer Kostenwerte

Um die getrennt definierten Vektoren für Kostenwert und -grenze zu vereinigen, wurde im Package „cost_support“ der ″&″ Operator entsprechend überladen.

```
ARCHITECTURE Version2 OF Komponente1 IS
   --Definition der Kostenwerte
   CONSTANT cost_val:criterion_val_vect:=
           (Anschaffung   =>30.0,
            Stromaufnahme =>3.0,
            Fehlerrate    =>1.0e-6,
            Groesse       =>3.0);
BEGIN
   --Modellgleichungen
   …
   --Zusammenfügen von Kostengrenze und -wert
   cost<=cost_limit & cost_val;
END;
```

Quelltext 6-4 Komponente 1, Architekturvariante 2 mit Definition ihrer Kostenwerte

```
USE work.cost_support.ALL;
ENTITY Komponente3 IS
   --funktionale Generics des Modells
   …
   PORT( --Port für Kostenvektor
         cost:OUT criterion_vect:=(OTHERS=>(-1.0,-1.0));

         --funktionale Ports des Modells
         …
      );
END;

ARCHITECTURE Version1 OF Komponente3 IS
   --Definition der Kostenwerte
   CONSTANT cost_val:criterion_vect:=
           (Anschaffung =>(val=>-1.0, lim=>80.0),
           Stromaufnahme=>(val=>-1.0, lim=>100.0),
           Fehlerrate   =>(val=>-1.0, lim=>1.0e-4),
           Groesse      =>(val=>-1.0, lim=>20.0));
BEGIN
   --Modellgleichungen
   …
   cost<=cost_val;
END;
```

Quelltext 6-5 Komponente 3, gemeinsame Definition von Kostengrenzen und -werten

6.4 Resultate

Die Simulation wurde mit dem VHDL Simulator „Modelsim“ von Mentor Graphics durchgeführt. Der Ansatz der Kostenmodellierung hat dabei eine sehr gute Stabilität bewiesen. Die zur Berechnung der Kosten notwendige Rechenzeit ist vernachlässigbar. Die Kostenüberschreitung bei der Stromaufnahme der Version 1 der Komponente 1 wurde zuverlässig erkannt.

Das Bestimmen der optimalen Konfiguration erfolgte durch eine systematische Simulation der möglichen Systemkonfigurationen. Bei der Anwendung auf ein reales System müssen zur Bestimmung der Funktion ebenfalls viele Kombinationen aus verfügbaren Komponenten simuliert werden, so daß für die Bestimmung der Kosten kaum ein Mehraufwand entsteht.

Als kostenoptimales System haben sich bei einem Wichtungsvektor

```
CONSTANT Kostenwichtung: criterion_weight_vect:=
                        (Anschaffung   =>0.4,
                         Stromaufnahme =>0.3,
                         Fehlerrate    =>0.2,
                         Groesse       =>0.1);
```

die Konfigurationen entsprechend Tabelle 6-3 ergeben. Für die Komponenten 2, 3 und 6 steht jeweils nur eine Architektur zur Verfügung.

	Komponente K1	Komponente K4	Komponente K5	Komponente K7	skalares Gütemaß
Variante	2	2	1	1	0,61
Variante	2	3	1	1	0,61

Tabelle 6-3 Kostenoptimale Konfiguration des Testsystems

Die *worst case* Konfiguration des System ist in Tabelle 6-4 dargestellt.

	Komponente K1	Komponente K4	Komponente K5	Komponente K7	skalares Gütemaß
Variante	3	1	2	2	0,83

Tabelle 6-4 Worst case Konfiguration des Testsystems

Es zeigt sich, daß trotz größerer Wichtung auf den Kostenfaktor „Anschaffungskosten“ im System für Komponente 1 nicht die preiswerteste Version gewählt wird, da diese z. B. eine sehr hohe Stromaufnahme aufweist. Da zur Berechnung des skalaren Gütemaßes das Verhältnis von Kostenwert zu Kostengrenze ermittelt wird, spielt die absolute Größe eines Kostenwertes keine Rolle, sondern nur dessen Relation zur Kostengrenze.

Die interaktive Bestimmung des Optimums ist – insbesondere wenn viele Architekturalternativen zur Auswahl stehen – u. U. nicht mehr sinnvoll. Mit der hier vorgestellten Methode ist deshalb auch eine automatische Optimierung möglich. Dazu können prinzipiell Tools zur Optimierung des Verhaltens, wie z B. [125] oder [126] eingesetzt werden. Voraussetzung ist allerdings, das solche Werkzeuge in der Lage sind, die `CONFIGURATION` eines VHDL-Modells zu ändern und den entsprechenden VHDL-Compiler zu starten.

Solange kein Kostenparameter von einem Signal oder von einer Quantity abhängt, verursacht die Optimierung der Kosten gegenüber der reinen Optimierung des Verhaltens keine nennenswerte Verlängerung der Simulation, da in diesem Fall die Berechnungen zur Kostenoptimierung nur zum Zeitpunkt 0 s ausgeführt werden.

Ein weiterer Punkt der bei digitalen Modellen berücksichtigt werden muß, ist die Synthese des Modells. Es ist darauf zu achten, daß die für die Kostenoptimierung verwendeten Datenobjekte nicht mit in die Synthese einbezogen werden, da diese nicht synthetisierbar sind. Dies ist compilerspezifisch möglich, bei Tools der Firma Synopsys z. B. mittels:

```
--synopsys translate_off
Definition eines Kostenvektors
--synopsys translate_on
```

Für die Modellierung heterogener Systeme mit VHDL-AMS konnte diese Methode allerdings noch nicht angewandt werden, da die zur Verfügung stehenden Versionen der VHDL-AMS Simulatoren „AdvanceMS“ (Mentor Graphics) und „hAMSter“ bzw. „Simplorer“ (Ansoft) Teile der für diese Methode notwendigen Typdefinitionen noch nicht beherrschen.

6.5 Kombination von Kostenmodellierung und MAM

Die Kostenmodellierung für sich ermöglicht die interaktive Optimierung der Kostenparameter während der Simulation des Systems. Sie schafft eine Grundlage für ein automatisiertes Arbeiten mit Werkzeugen zur Verhaltensoptimierung und durch die gemeinsame Beschreibung von Kosten und Verhalten der Komponente wird außerdem die Konsistenz dieser Daten zueinander gewährleistet. Die Bestimmung der Kostenparameter ist, wie Abschnitt 6.1 bereits angedeutet, jedoch oft nicht möglich, da häufig keine Verfahren oder Methoden existieren, um Kostenparameter auf der Basis der Spezifikationsdaten abzuschätzen.

Verbindet man Kostenmodellierung und Multi-Architecture-Modellierung, so besitzen alle unterschiedlich abstrakten Modelle einer Komponente dieselbe Schnittstelle für die Kostenparameter und enthalten somit alle Bestandteile des zu optimierenden Kostenvektors. Damit besteht die Möglichkeit, die unterschiedlich abstrakten Modelle mit den jeweils bis zu dieser Entwurfsphase bekannten Kostenwerten auszustatten. Bis zu diesem Zeitpunkt unbekannte Kostenwerte können durch Setzen ihres Wert auf `-1` von der weiteren Berechnung ausgeschlossen werden. Damit sind die unbekannten Kostenwerte kein Bestandteil der Optimierung. Mit fortschreitendem Entwurf lassen sich aber immer mehr der Kostenwerte bestimmen, wodurch die Kostenvorhersage während des Entwurfs automatisch immer präziser wird und sich kostenbezogene Entwurfsentscheidungen damit so früh wie möglich treffen lassen. Bild 6-2 verdeutlicht dies nochmals anhand eines fiktiven Sensors. Während sich aus dem abstrakten Modell z. B. nur die Chipfläche im Vergleich zu bereits realisierten ähnlichen Strukturen abschätzen läßt, kann im Netzwerkmodell zusätzlich die Stromaufnahme berechnet werden. Im *Reduced Order Modell*, welches aus einem FEM-Modell des Sensors erzeugt wurde, stehen dann alle zur Kostenoptimierung notwendigen Informationen zur Verfügung

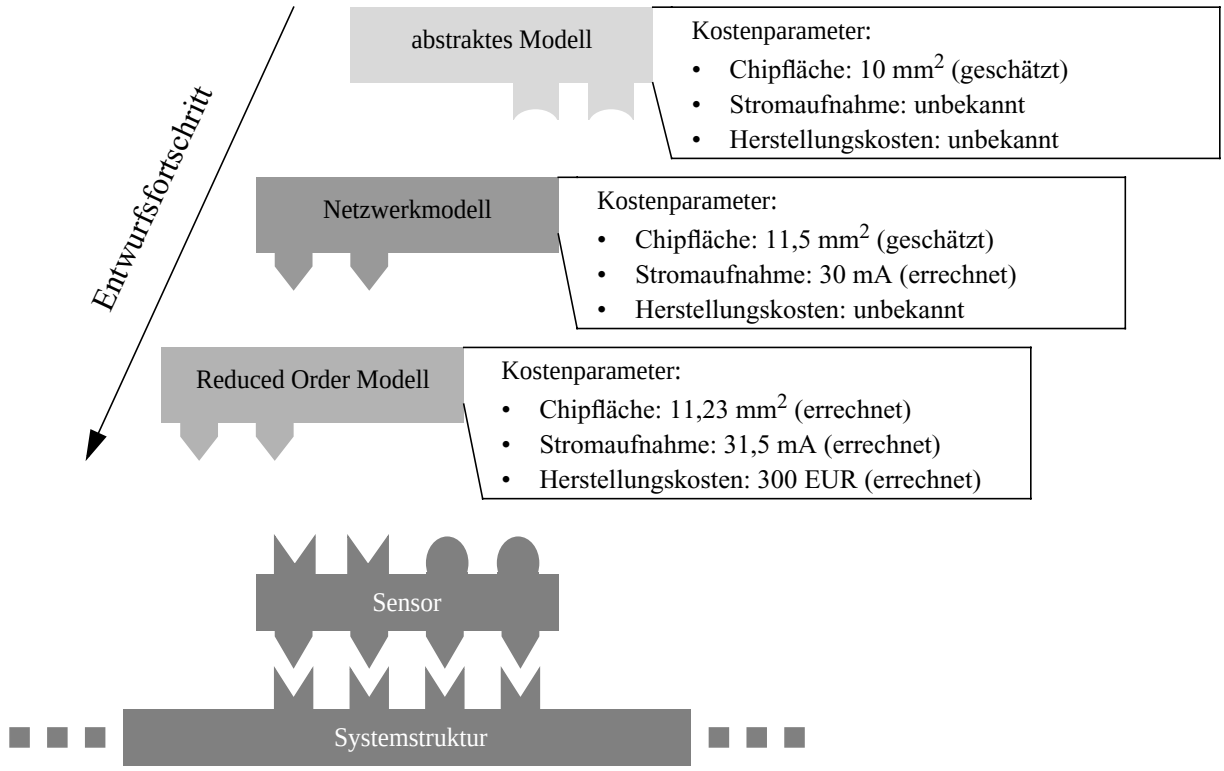


Bild 6-2 Kombination von Kostenmodellierung und MAM

6.6 Fazit

Das hier vorgestellte Verfahren der Kostenmodellierung ermöglicht die Abbildung nichtfunktionale Parameter einer Komponente wie Herstellungs- und Entwicklungskosten, Baugröße oder Fehlertoleranz in das Verhaltensmodell der Komponente. Dadurch wird zum einen die Konsistenz zwischen Verhaltens- und Kostenmodell gewährleistet. Zum anderen erlaubt dies die Optimierung der Kostenparameter gemeinsam mit den Verhaltensparametern. Durch die Kombination dieses Verfahrens mit der MAM lassen sich Kostenparameter in die Optimierung mit Einbeziehung, die erst im Laufe der Entwicklung der Komponente ermittelt werden können, wobei keine Änderungen an den Modellen notwendig sind. Bei der Anwendung der Kostenmodellierung wurde keine nennenswerte Verschlechterung der Simulationsperformance beobachtet.

7 Zusammenfassung

Die vorliegende Arbeit befaßt sich mit der Simulation heterogener Systeme, deren Komponentenmodelle auf unterschiedlichen Abstraktionsebenen vorliegen. Die gemeinsame Simulation unterschiedlich abstrakter Modelle ist notwendig, da Systemmodelle komplexer heterogener Systeme, die nur aus abstrakten Modellen bestehen, ungenaue Ergebnisse liefern. Modelle die nur aus detaillierten Modellen bestehen, weisen dagegen sehr lange Simulationszeiten auf. Es wurde daher im Rahmen dieser Arbeit eine Methode entwickelt, die es erlaubt, verschieden abstrakte Modelle (Multiple Architekturen) für dieselbe Komponente zu hinterlegen, um das Systemmodell selektiv zwischen Simulationsgenauigkeit und Simulationsgeschwindigkeit konfigurieren zu können. Allerdings bedingt in der Regel der Abstraktionsgrad des Modells die Art der verwendeten Schnittstelle, wodurch der Austausch der Modelle über Abstraktionsebenen hinweg deutlich erschwert wird. Daraus ergibt sich die Forderung, eine einheitliche Schnittstelle für unterschiedlich abstrakte Modelle zu schaffen. Für Hardwarebeschreibungssprachen wie VHDL-AMS wurden die folgenden Lösungsansätze erarbeitet:

1. Analoge Schnittstellen

   Für analoge elektrische und nichtelektrische Schnittstellen werden üblicherweise auf funktioneller Ebene nichtkonservative Knoten (`QUANTITY`), auf Netzwerkebene konservative Knoten (`TERMINAL`) verwendet. Da sich mit nichtkonservativen Knoten rückwirkungsbehaftete Verbindungen nur mit erheblichem Aufwand (je ein Knoten für Strom und Spannung und explizite Beschreibung der Kirchhoffschen Gesetze notwendig) herstellen lassen, ist es vorteilhaft wenn bereits auf hoher Abstraktionsebene konservative Knoten verwendet werden.

2. Einfache digitale Schnittstellen

   Digitale Interfaces verwenden sowohl auf algorithmischer als auch auf RT-Ebene Signale. Allerdings können sich die Datentypen der Signale auf den unterschiedlichen Abstraktionsebenen unterscheiden. So sind auf algorithmischer Ebene Real- und Integerwerte üblich, auf RT-Ebene oft `std_locic`-Vektoren. Da `std_logic`-Zustände wie `'Z'`,`'U'` oder `'X'` nicht in Integerwerte umgewandelt werden können, müssen auch die Schnittstellen auf hoher Abstraktionsebene `std_locic`-Vektoren verwenden.

3. Digitale Schnittstellen mit Kommunikationsprotokoll

   Digitale Schnittstellen werden nicht nur durch ihren Datentyp, sondern auch durch die Art und Weise wie Daten übertragen werden, charakterisiert. In abstrakten Modellen findet der Datenaustausch in Form von übergebenen Integer- oder Realwerten bzw. in Form von Datenpaketen statt, während auf RT-Ebene ein zeitgenaues Schema der Übertragung einzuhalten ist. Sollen die RT-Modelle ein bestimmtes Protokoll verwenden, so müssen auch die mit diesem Modell verbundenen abstrakten Modelle in der Lage sein, über dieses Protokoll zu kommunizieren. Dazu sind Prozeduren in das abstrakte Modell bei dessen Erstellung zu integrieren, die eine Konvertierung zwischen abstrakter Zahlendarstellung und zeitgenauer Kommunikation und zurück vornehmen. Da die

Automatisierung dieser Prozeduren nicht Gegenstand dieser Arbeit ist, eignet sich das hier vorgestellte Verfahren in erster Linie für Protokolle geringer bis mittlerer Komplexität. Für die im Entwurf heterogener System und Mikrosysteme nach wie vor häufig anzutreffende RS232-Schnittstelle und den I^2C-Bus wurden Beispielimplementierungen erstellt.

4. Analog/digital gemischte Schnittstellen

   Sollen im Rahmen der Gesamtsystemsimulation Untersuchungen zum exakten elektrischen Verhalten (z. B. Signallaufzeiten, Störfestigkeit) einer digitalen Schnittstelle durchgeführt oder digitale Modelle bis auf Transistor-Level verfeinert werden, so muß man Modelle miteinander verbinden, die auf hoher Abstraktionsebene Signale, auf niedriger Abstraktionsebene jedoch Terminals besitzen. Da digitale Signale nicht geeignet sind, elektrische Netzwerke miteinander zu verbinden, sind hier Terminals auf allen Abstraktionsebenen einzusetzen. Zur Anbindung der digitalen Signale an die Terminals des Interface wurden spezielle A/D- und D/A-Wandler erarbeitet, die in der Lage sind, sich technologiespezifisch zu verhalten (z. B. CMOS, TTL) oder, wenn mehrere Treiber auf ein Terminal schreiben, eine VHDL *resolution function* z. B. für den Datentyp `std_logic` nachzubilden.

   Die Notwendigkeit zur Verbindung von analogen mit digitalen Modellen besteht auch, wenn Komponenten abstrakt als analoges Modell beschrieben werden und man diese anschließend zu digitalen, synthetisierbaren Modellen verfeinert. Für diese Konfigurationen sind digitale Signale als Schnittstellen zu verwenden. Zur Verbindung von analogen Modellen mit den digitalen Signalen entstanden spezielle Konverter, die die Verbindung von Quantities mit Signalen erlauben. Dabei ist zu beachten, daß die Verwendung einer solchen Schnittstellen eine AC-Simulation der abstrakten analogen Modelle verhindert.

Aus diesen Ansätzen ergibt sich, daß die Schnittstelle, die ein Modell auf der niedrigsten – im Rahmen der Gesamtsystemsimulation zur Anwendung vorgesehenen – Abstraktionsebene benötigt, bereits zu Beginn des Entwurfes auf hoher Abstraktionsebene bekannt sein muß. Daraus resultiert eine Modifikation im Top-Down-Entwurfsprozeß. Ausgehend von der Systemspezifikation werden bei dem modifizierten Top-Down-Entwurf **die** Komponenten und Teilsysteme des Systems bestimmt, für die mehrere Architekturen hinterlegt werden sollen. Für diese Komponenten und die direkt mit ihnen verbundenen Komponenten erfolgt nun die Bestimmung der entsprechend notwendigen Schnittstellen. Die Anwendung des Verfahrens im Entwurf des Meßsystems „Demonstrator Vibrationssensor-Array“ hat gezeigt, daß dies in einem akzeptablen Rahmen möglich ist. Mit den so gewonnenen Informationen kann man dann den Top-Down-Design-Prozeß wie bisher fortsetzen, mit der Ausnahme, daß die zuvor bestimmten Schnittstellen anstelle der abstrakten Schnittstellen verwendet und die entsprechenden Anpassungsmaßnahmen zum abstrakten Systemverhalten hinzugefügt werden. Da diese Anpassungsmaßnahmen mit der Verfeinerung des Modells entfallen, hat die MAM keinen Einfluß auf Größe oder Simulationsverhalten des detaillierten Modells.

Die Simulationszeit eines abstrakten Modells mit detaillierter Schnittstelle kann sich bei Modellen mit wenig Verhaltensbeschreibung und viel Schnittstellenbeschreibung (*worst case* Szenarion für die MAM) gegenüber einem Modell mit abstrakter Schnittstelle deutlich erhöhen. Daraus ergibt sich, daß die Schnittstellenauslegung für die MAM nicht pauschal, sondern selektiv nur auf die im Rahmen des modifizierten Top-Down-Entwurfs bestimmten Modelle angewandt werden sollte. Wie die Simulation des mikroelektromechanischen Meßsystems zur Klassifizierung von Getriebeschäden und Werkzeugverschleiß jedoch gezeigt hat, fällt der Simulationszeitanstieg in einem solchen realen System – wie hier mit 16 % – eher gering aus. Im Vergleich zur Simulation eines Systems aus detaillierten Komponenten kann eine beträchtliche Einsparung an Simulationszeit, im vorliegenden Fall knapp 50 %, erzielt werden. Die frühzeitige Bestimmung der auf niedrigem Abstraktionsniveau notwendigen Schnittstelle hat sich dabei als praktikabel erwiesen.

Die MAM ermöglicht dem Entwerfer, Modelle über Hierarchieebenen hinweg ohne Anpassungsmaßnahmen auszutauschen. Dadurch entfallen die fehleranfälligen und zeitaufwendigen Anpassungsschritte im Entwurfsprozeß, und die Systemsimulation wird zwischen Genauigkeit und Geschwindigkeit skalierbar. Da die abstrakten Modelle der Systemsimulation gleichzeitig als Testbench für detaillierter Modelle dienen, spart dies dem Entwerfer Zeit bei der Erstellung der Modelle. Die Zeitersparnis sollte die u. U. längere Simulationszeit in der Regel kompensieren. Selbst wenn dies nicht der Fall sein sollte, ist die Zeitersparnis beim Entwurf des Modells höher zu bewerten als die längere Simulationszeit, da der Rechner die Simulation auch unabhängig vom Entwerfer durchführen kann, während sich der Entwerfer weiteren Aufgaben widmet.

Mit der direkten Anwendung der MAM im Entwurf mit SystemC-AMS konnte gezeigt werden, daß sich der Ansatz auch auf andere Beschreibungsformen übertragen läßt. Zusätzlich bieten die erweiterten Beschreibungsmöglichkeiten einer objektorientierten Sprache einen größeren Spielraum bei der Gestaltung der MAM. So unterstützt SystemC mit den Channels und *Interface Method Calls* bereits einen Ansatz zur Verbindung unterschiedlich abstrakter Modelle. In Kombination mit der MAM kann der in SystemC vorgesehene Einsatz von Adaptern zur Anpassung von noch nicht verfeinerten Modellen an verfeinerte Channels entfallen. Eine Anwendung der SystemC-Channels auf analoge Schnittstellen erfordert jedoch noch die Bereitstellung einiger weiterer Methoden seitens der Sprache SystemC-AMS.

Neben dem eigentliche Ziel der MAM, verschieden abstrakte Modelle für eine Komponente in der Modellierung einzusetzen, ermöglicht die Vereinheitlichung der Schnittstellen weitere Ansätze zur Optimierung des Entwurfs heterogener Systeme, wie am Beispiel der Kostenmodellierung gezeigt wurde. Mit dem Verfahren der Kostenmodellierung lassen sich nichtfunktionale Parameter wie Baugröße oder Herstellungs- und Entwicklungskosten in das Verhalten der Komponente abbilden, so daß diese Kosten gemeinsam mit dem Verhalten optimiert werden können. Die Kombination von MAM und Kostenmodellierung erlaubt es, Kostenparameter in die Optimierung einzubeziehen, die sich erst während der Entwicklung der Komponente ermittelt lassen. Dabei müssen keine Änderungen an den Modellen vorgenommen werden. Die Anwendung der Kostenmodellierung verursacht keinen nennenswerte Verschlechterung der Simulationsperformance.

8 Ausblick

Die im Rahmen dieser Arbeit entwickelte Methode der Multi-Architecture-Modellierung kann nach ihrem jetzigen Bearbeitungsstand als weitgehend abgeschlossen betrachtet werden. Die selektive Verwendung derjenigen Schnittstelle auf hohem Abstraktionsniveau, die auf niedrigem Abstraktionsniveau notwendig ist, hat sich als praktikabel erwiesen.

Da die zur Unterstützung der Methode notwendigen Funktionen und Prozeduren als *open source* VHDL-Code erstellt wurden, besteht jederzeit die Möglichkeit, diese an die konkreten Erfordernisse zukünftiger Entwürfe heterogener Systeme anzupassen und gegebenenfalls zu erweitern. Denkbar wäre z. B. die Schaffung eines CAN-Bus-Abstraktionswandlers für den Einsatz der Methode im Automotive-Sektor. Ebenfalls überlegenswert ist die Erweiterung der speziellen A/D- und D/A-Wandler zur Verbindung von Signalen mit Terminals für eine differentielle Datenübertragung, wie sie z. B. beim CAN-Bus oder der LVDS-Datenübertragung notwendig sind.

Im Rahmen dieser Arbeit wurde auch die Unterstützung der MAM durch neu zu entwickelnde Entwurfswerkzeuge diskutiert. So wäre es z. B. möglich, ein Tool zu erstellen, das die Schnittstellen der VHDL-Beschreibung automatisch von hoher auf niedrigerer Abstraktion modifiziert und die notwendigen Konvertierungsfunktionen oder -prozeduren bzw. entsprechende VHDL-Statements in das Modell integriert. Da aber zum einen die zu integrierenden Konvertierungsfunktionen oder -prozeduren nach wie vor von Hand erstellt werden müßten und zum anderen Informationen zur Schnittstelle auf niedrigerem Abstraktionsniveau in die Kommentarsyntax von VHDL implementiert werden müßte, steht der notwendige Aufwand zur Realisierung eines solchen Programms in keinem Verhältnis zur erzielbaren Vereinfachung der Anwendung dieser Methode. Darüber hinaus steht zu befürchten, daß bei Fertigstellung eines solchen Programms für eine bestimmte Hardwarebeschreibungssprache diese aufgrund der rasanten Entwicklung auf dem Gebiet der Entwurfsautomatisierung bereits veraltet ist, während die Methode an sich auch auf zukünftige Systembeschreibungssprachen anwendbar ist, sofern die in Abschnitt 4.3, S. 86 genannten Voraussetzungen erfüllt sind.

Für die Generierung synthesefähiger Modelle von Transceivern für digitale Protokolle steht mit SVE [80] ein leistungsfähiger Ansatz zur Verfügung. Mit diesem Ansatz ist es möglich, aus einer abstrakten Protokollbeschreibungen ein Modell in der Systembeschreibungssprache SystemC automatisch zu erzeugen. Das entstandene Modell ist somit auch im Rahmen der MAM unter SystemC-AMS nutzbar. Da für die Übersetzung von synthesefähigem SystemC nach VHDL ebenfalls Werkzeuge existieren, können die auf diesem Wege erzeugten Modelle prinzipiell auch im Rahmen der MAM unter VHDL-AMS eingesetzt werden. Voraussetzung dafür ist allerdings, daß die zur Verfügung stehenden VHDL-AMS-Simulatoren in naher Zukunft die im Rahmen eines Synthesesubsets der Sprachsyntax definierten Konstrukte vollständig beherrschen. Dadurch könnten auch komplexere digitale Schnittstellen für die MAM erschlossen werden.

Anhang A VHDL-AMS Quelltext des I²C Abstraktionswandlers

Abstraktionswandler-Prozeduren für I²C-Kommunikation

```
--Einbinden der IEEE std_logic_1164 Bibliothek
library ieee;
use ieee.std_logic_1164.all;

--Package mit Prozedurdeklarationen
package i2c is
   --Datenpuffer-Array fuer empfangene Daten
   type i2c_data is array (natural range <>) of integer range 0 to 255;
   --Uebertragungsart
   type i2c_speed is (std,fast);

   --Prozedur zum Daten senden fuer IIC_Master
   procedure master_send_data ( constant adress: in integer range 0 to 127;
                                constant data2send: in i2c_data;
                                signal   SCL: inout std_logic;
                                signal   SDA: inout std_logic;
                                constant speed: in i2c_speed:=std);

   --Prozedur zum Daten empfangen fuer IIC_Master
   procedure master_receive_data (constant adress: in integer range 0 to 127;
                                variable data_received: out i2c_data;
                                signal   SCL: inout std_logic;
                                signal   SDA: inout std_logic;
                                constant speed: in i2c_speed:=std);

   --Prozedur zum Daten senden und empfangen fuer IIC_Slave
   procedure slave (  constant adress: in integer range 0 to 127;
                      constant data2send: in i2c_data;
                      variable data_received: out i2c_data;
                      signal   SCL: inout std_logic;
                      signal   SDA: inout std_logic);

   --Prozedur zur Initialisierung der Schnittstelle
   procedure init (   signal   SCL: out std_logic;
                      signal   SDA: out std_logic);
end package i2c;
```

Prozedurdefinitionen

```
--Elemente des MAM-Packages verwenden
use work.mam_support.all;

package body i2c is
   ------------------------------------------------------------------------------
   --Prozedur zum Daten senden fuer I2C-Master
   ------------------------------------------------------------------------------
   procedure master_send_data ( constant adress: in integer range 0 to 127;
                                constant data2send: in i2c_data;
                                signal SCL: inout std_logic;
                                signal SDA: inout std_logic;
                                constant speed: in i2c_speed:=std) is
      variable delay:time:=0 ms;
      constant adress_bit:bit_vector(6 downto 0):=int2bit_vector(adress,7);
      variable data:bit_vector(7 downto 0);
   begin
   --auf legale Adresse pruefen
      if adress<8 or adress>119 then
         report "illegal adress";
      end if;
   --Uebertragungsgeschwindigkeit einstellen
      if speed=std then
         delay:=1 sec / (4*100e3); --4fach Oversampling, 100kbps
      else
         delay:=1 sec / (4*400e3); --4fach Oversampling, 400kbps
      end if;
   --Datenuebertragung
      if SCL='0' then
         report "Bus is busy";
      else
         wait for delay;
      --Start-Bit
         SDA<='0';
         wait for delay;
      --Adresse
         for i in 6 downto 0 loop
            SCL<='0';
            wait for delay;
            if adress_bit(i)='0' then
               SDA<='0';
            else
               SDA<='H';
            end if;
            wait for delay;
            SCL<='H';
            wait for 2*delay;
         end loop;
         SCL<='0';
         wait for delay;
      --r/w Bit
         SDA<='0';
         wait for delay;
         SCL<='H';
         wait for 2*delay;
         SCL<='0';
         wait for delay;
         SDA<='H';
         wait for delay;
```

```
      SCL<='H';
      wait for 2*delay;
   --Acknoledge lesen
      if SDA/='0' then
         --Slave nicht bereit
         report "Slave "& integer'image(adress) &" is busy or not present";
      else
   --Daten senden
         for i in data2send'range loop
            data:=int2bit_vector(data2send(i),8);
            for j in 7 downto 0 loop
               SCL<='0';
               wait for delay;
               if data(j)='0' then
                     SDA<='0';
               else
                     SDA<='H';
               end if;
               wait for delay;
               SCL<='H';
               wait for 2*delay;
            end loop;
            SCL<='0';
            wait for delay;
            SDA<='H';
            wait for delay;
            SCL<='H';
            wait for 2*delay;
   --Acknoledge lesen
            if SDA/='0' then
               --Slave nicht länger bereit
               if i<data2send'right then
                     report "slave "& integer'image(adress) &" is busy";
               end if;
               exit;
            end if;
         end loop;
         SCL<='0';
         wait for delay;
   --Stop-Bit
         SDA<='0';
         wait for delay;
         SCL<='H';
         wait for delay;
         SDA<='H';
         wait for delay;
      end if;
   end if;
end;
```

```
-------------------------------------------------------------------------------
--Prozedur zum Daten empfangen fuer I2C-Master
-------------------------------------------------------------------------------
procedure master_receive_data (constant adress: in integer range 0 to 127;
                               variable data_received: out i2c_data;
                               signal   SCL: inout std_logic;
                               signal   SDA: inout std_logic;
                               constant speed: in i2c_speed:=std) is
   variable delay:time:=0 ms;
   constant adress_bit:bit_vector(6 downto 0):=int2bit_vector(adress,7);
   variable data:bit_vector(7 downto 0);
begin
--auf legale Adresse pruefen
   if adress<8 or adress>119 then
      report "illegal adress";
   end if;
--Uebertragungsgeschwindigkeit einstellen
   if speed=std then
      delay:=1 sec / (4*100e3); --4fach Oversampling, 100kbps
   else
      delay:=1 sec / (4*400e3); --4fach Oversampling, 400kbps
   end if;
--Datenuebertragung
   if SCL='0' then
      report "Bus is busy";
   else
      wait for delay;
   --Start-Bit
      SDA<='0';
      wait for delay;
   --Adresse
      for i in 6 downto 0 loop
         SCL<='0';
         wait for delay;
         if adress_bit(i)='0' then
            SDA<='0';
         else
            SDA<='H';
         end if;
         wait for delay;
         SCL<='H';
         wait for 2*delay;
      end loop;
      SCL<='0';
      wait for delay;
   --r/w Bit
      SDA<='H';
      wait for delay;
      SCL<='H';
      wait for 2*delay;
      SCL<='0';
      wait for delay;
      SDA<='H';
      wait for delay;
      SCL<='H';
      wait for 2*delay;
   --Acknoledge lesen
      if SDA/='0' then
         --Slave nicht bereit
         report "Slave "& integer'image(adress) &" is busy or not present";
      else
```

```
    --Daten empfangen
         for i in data_received'range loop
            SCL<='0';
            wait for delay;
            SDA<='H';
            for j in 7 downto 0 loop
               wait for delay;
               SCL<='H';
               wait for delay;
               data(j):=to_bit(SDA);
               wait for delay;
               SCL<='0';
               wait for delay;
            end loop;
            data_received(i):=bit_vector2int(data);
            if i=data_received'right then
               --Datenpuffer voll
               SDA<='H'; --kein Acknowledge
               wait for delay;
            else
               SDA<='0'; --Acknowledge
               wait for delay;
            end if;
            SCL<='H';
            wait for 2*delay;
         end loop;
         SCL<='0';
         wait for delay;
      --Stop-Bit
         SDA<='0';
         wait for delay;
         SCL<='H';
         wait for delay;
         SDA<='H';
         wait for delay;
      end if;
   end if;
end;
```

```
------------------------------------------------------------------------------
   --Prozedur zum Daten senden und empfangen fuer I2C-Slave
------------------------------------------------------------------------------
procedure slave (    constant adress: in integer range 0 to 127;
                     constant data2send: in i2c_data;
                     variable data_received: out i2c_data;
                     signal   SCL: inout std_logic;
                     signal   SDA: inout std_logic) is
     variable delay,t1:time:=0 ms;
     variable adress_bit:bit_vector(6 downto 0);
     variable data:bit_vector(7 downto 0);
     variable rw:bit;
   begin
   --auf legale Adresse pruefen
     if adress<8 or adress>119 then
       report "illegal adress";
     end if;
     loop
   --auf start-bit warten
       wait on SDA until SDA='0' and SCL/='0';
   --aus dem Takt die Datenrate bestimmen
       wait on SCL until SCL='0';
       t1:=now;
       wait on SCL until SCL/='0';
       delay:=(now-t1)/2;
   --Adresse lesen
       for i in 6 downto 0 loop
         wait for delay;
         adress_bit(i):=to_bit(SDA);
         wait on SCL until SCL/='0';
       end loop;
       wait for delay;
   --r/w Bit lesen
       rw:=to_bit(SDA);
       wait on SCL until SCL='0';
       exit when adress=bit_vector2int(adress_bit);
     end loop;
     wait for delay;
   --Acknowledge senden
     SDA<='0';
     wait on SCL until SCL='0';
     if rw='0' then
   --Daten lesen
       for i in data_received'range loop
         wait for delay;
         SDA<='H';
         for j in 7 downto 0 loop
           wait on SCL until SCL/='0';
           wait for delay;
           data(j):=to_bit(SDA);
         end loop;
         data_received(i):=bit_vector2int(data);
         wait on SCL until SCL='0';
         if i=data_received'right then
           --Datenpuffer voll
           wait for delay;
           SDA<='H'; --kein Acknowledge
           wait on SCL until SCL='0';
         else
           wait for delay;
           SDA<='0'; --Acknowledge
```

```
            wait on SCL until SCL='0';
          end if;
        end loop;
      else
   --Daten schreiben
        for i in data2send'range loop
          data:=int2bit_vector(data2send(i),8);
          for j in 7 downto 0 loop
            wait for delay;
            if data(j)='0' then
              SDA<='0';
            else
              SDA<='H';
            end if;
            wait on SCL until SCL='0';
          end loop;
          wait for delay;
          SDA<='H';
          wait on SCL until SCL/='0';
          wait for delay;
   --Acknoledge lesen
          if SDA/='0' then
            --Master nicht länger bereit
            if i<data2send'right then
              report "master is busy";
            end if;
            exit;
          end if;
        end loop;
      end if;
   end;



-------------------------------------------------------------------------------
   --Prozedur zum Initialisieren der Schnittstelle
-------------------------------------------------------------------------------
   procedure init (  signal   SCL: out std_logic;
                     signal   SDA: out std_logic) is
   begin
      SCL<='H';
      SDA<='H';
      wait;
   end;

end package body i2c;
```

Anwendungsbeispiel

```
library ieee;
use ieee.std_logic_1164.all;

--I2C-Package sichtbar machen
use work.i2c;
use work.i2c.i2c_data;

--Test-Entity
entity tb is
end;

architecture a of tb is
   signal SDA,SCL:std_logic:='H';

begin

   master:process
      --Datenpuffer fuer ein Datenwort
      variable data:i2c_data(0 to 0);
   begin
      --Master soll ein Datenwort (data) von Slave mit Adresse 0x55 lesen
      i2c.master_receive_data(16#55#,data,SCL,SDA);
      wait;
   end process;

   slave:process
      --Datenpuffer fuer ein Datenwort
      variable data:i2c_data(0 to 0);
   begin
      --Slave mit Adresse 0x55 hat zwei Datenworte 0x33 und 0x66 zum Versenden
      i2c.slave(16#55#,(0=>16#33#,1=>16#66#),data,SCL,SDA);
      wait;
   end process;
end;
```

Anhang B VHDL-AMS-Package MAM_support

```
LIBRARY disciplines,ieee;
USE disciplines.electromagnetic_system.ALL;
USE ieee.std_logic_1164.ALL;
USE ieee.math_real.ALL;

PACKAGE MAM_support IS
   ------------------------------------------------------------------------------
   --einfache Funktionen zur Datentypskonvertierung
   ------------------------------------------------------------------------------
   FUNCTION bit_vector2int(in1: bit_vector) RETURN integer;
   FUNCTION int2bit_vector(in1: in integer;
                           size: in integer:=8) RETURN bit_vector;
   FUNCTION bit2real(in1:bit; low_level,high_level:real) RETURN real;
   FUNCTION real2bit(in1:real; threshold:real) RETURN bit;
   FUNCTION fix2real( CONSTANT data_in:std_logic_vector;
                      CONSTANT sig_wide:positive;
                      CONSTANT fix_point:natural) RETURN real;
   FUNCTION real2fix( CONSTANT data_in:real;
                      CONSTANT sig_wide:positive;
                      CONSTANT fix_point:natural) RETURN std_logic_vector;

   ------------------------------------------------------------------------------
   --Parameter für A/D- und D/A-Wandler für analog/digital gemischte Schnittstellen
   ------------------------------------------------------------------------------
   CONSTANT Rs:real;       -- Ausgangswiderstand für starke Treiber
   CONSTANT Rw:real;       -- Ausgangswiderstand für schwache Treiber
   CONSTANT Rl:real;       -- Eingangswiderstand: 0 < Rs < Rw << Rl
   CONSTANT U1:real;       -- Ausgangsspannung 'H' und '1'
   CONSTANT U0:real;       -- Ausgangsspannung 'L' und '0'
   CONSTANT Ux:real;       -- Ausgangsspannung 'X', 'W', 'U' und '-'
   CONSTANT Uz:real;       -- 'Z'-Testspannung Uz < U0 oder Uz > U1

   CONSTANT max_drv:real;  -- Maximale Anzahl von Treibern, um bei mehrfach getriebenen
                           -- Signalen am Eingang den richtigen Zustand wiederzuerkennen

   CONSTANT Ui1:real;      -- Minimale Eingangsspannung, um eine '1' zu erkennen,
                           -- wenn kein Treiber den Wert 'X' besitzt

   CONSTANT Ui0:real;      -- Maximale Eingangsspannung, um eine '0' zu erkennen,
                           -- wenn kein Treiber den Wert 'X' besitzt

   CONSTANT Nw_N1:real;    -- Verhältnis zwischen Treibern mit 'W' und '1', so daß
                           -- eine '1' erkannt wird
   CONSTANT Nl_N1:real;    -- Verhältnis zwischen Treibern mit 'L' und '1', so daß
                           -- eine '1' erkannt wird
   CONSTANT Nw_N0:real;    -- Verhältnis zwischen Treibern mit 'W' und '0', so daß
                           -- eine '0' erkannt wird
   CONSTANT Nh_N0:real;    -- Verhältnis zwischen Treibern mit 'H' und '0', so daß
                           -- eine '0' erkannt wird
```

```
-----------------------------------------------------------------------
-- Treiber Signal(std_ulogic)->Terminal(electrical)
-----------------------------------------------------------------------
COMPONENT MAM_Sstdulogic2T IS
   PORT (SIGNAL s_in:IN std_ulogic; TERMINAL t1:electrical);
END COMPONENT;

-----------------------------------------------------------------------
-- Treiber Signal(bit)->Terminal(electrical)
-----------------------------------------------------------------------
COMPONENT MAM_Sbit2T IS
   PORT (SIGNAL s_in:IN bit; TERMINAL t1:electrical);
END COMPONENT;

-----------------------------------------------------------------------
-- Treiber Signal(real)->Terminal(electrical)
-- Bei Signalen von Typ integer: die Konvertierungsfunktion real(integer_Wert)
-- verwenden
-----------------------------------------------------------------------
COMPONENT MAM_Sreal2T IS
   PORT (SIGNAL s_in:IN real; TERMINAL t1:electrical);
END COMPONENT;

-----------------------------------------------------------------------
-- Treiber Terminal(electrical)->Signal(std_ulogic)
-----------------------------------------------------------------------
COMPONENT MAM_T2Sstdulogic IS
   PORT (TERMINAL t1:electrical; SIGNAL s_out: OUT std_ulogic);
END COMPONENT;

-----------------------------------------------------------------------
-- Treiber Terminal(electrical)->Signal(bit)
-----------------------------------------------------------------------
COMPONENT MAM_T2Sbit IS
   PORT (TERMINAL t1:electrical; SIGNAL s_out: INOUT bit );
END COMPONENT;

-----------------------------------------------------------------------
-- Treiber Terminal(electrical)->Signal(real)
-- Bei Übergabe an Signale von Typ integer: die Konvertierungsfunktion
-- integer(real_Wert) verwenden
-- ACHTUNG: Wert von hmax beachten
-- bei getakteten Systeme MAM_T2Sreal_clk verwenden
-----------------------------------------------------------------------
COMPONENT MAM_T2Sreal IS
   GENERIC (delta: real:= 0.1); -- Werteänderung die einen event auslösen soll
   PORT (TERMINAL t1:electrical; SIGNAL s_out: OUT real);
END COMPONENT;

-----------------------------------------------------------------------
-- Treiber Terminal(electrical)->Signal(real) für getaktete Systeme
-- Bei Übergabe an Signale von Typ integer: die Konvertierungsfunktion
-- integer(real_Wert) verwenden
-----------------------------------------------------------------------
COMPONENT MAM_T2Sreal_clk IS
   PORT (TERMINAL t1:electrical; SIGNAL clk: IN bit;
         SIGNAL s_out: OUT real);
END COMPONENT;
```

```
--------------------------------------------------------------------------------
--Konverter zur Umsetzung von Signalen im Festkommaformat auf Quantities
--sig_wide:   Bit-Breite des Signals, Zeierkomplementdarstellung
--fix_point:  Kommapossition 0=integer, 1=1 Nachkommastelle ...
--ramp_time:  Zeit zwischen Signalwechseln auf data_in,
--            dient zur linearen Interpolation der Quantity zwischen
--            zwei Signalwechseln
--data_in:    Eingangssignal
--data_out:   Ausgangsquantity
--enable_in:  Eingang des Enable-Signals
--enable_out: Ausgang des Enable-Signals
--------------------------------------------------------------------------------
COMPONENT MAM_s2q IS
   GENERIC (  ramp_time: real:=1.0e-6;
              sig_wide: positive:=8;
              fix_point: natural:=0);
   PORT (  SIGNAL data_in: IN std_logic_vector(sig_wide-1 DOWNTO 0);
           QUANTITY data_out: OUT real;
           SIGNAL enable_in: IN std_logic;
           SIGNAL enable_out: OUT std_logic);
END COMPONENT;


--------------------------------------------------------------------------------
--Konverter zur Umsetzung von Quantities auf Signale im Festkommaformat
--sig_wide:   Bit-Breite des Signals, Zeierkomplementdarstellung
--fix_point:  Kommapossition 0=integer, 1=1 Nachkommastelle ...
--data_in:    Eingangsquantity
--data_out:   Ausgangssignal
--enable_in:  Eingang des Enable-Signals
--enable_out: Ausgang des Enable-Signals
--Die Wandlung erfolgt mit jeder positiven Flanke des Enable-Signals
--Enable immer '1' -> keine Wandlung!
--------------------------------------------------------------------------------
COMPONENT MAM_q2s IS
   GENERIC (  sig_wide: positive:=8;
              fix_point: natural:=0);
   PORT (     QUANTITY data_in: IN real;
              SIGNAL data_out: OUT std_logic_vector(sig_wide-1 DOWNTO 0);
              SIGNAL enable_in: IN std_logic;
              SIGNAL enable_out: OUT std_logic);
END COMPONENT;
```

```
  ------------------------------------------------------------------------------
  --Konverter zur Umsetzung von Quantities auf Signale im Festkommaformat
  --Erzeugung des Enable-Signals aus Abtasttakt
  --sig_wide:   Bit-Breite des Signals, Zeierkomplementdarstellung
  --fix_point:  Kommapossition 0=integer, 1=1 Nachkommastelle ...
  --clock_div:  0 enable immer '1', sonst enable=clk/clock_div
  --data_in:    Eingangsquantity
  --data_out:   Ausgangssignal
  --clk:        Takt-Eingang
  --enable_out: Ausgang des Enable-Signals
  ------------------------------------------------------------------------------
  COMPONENT MAM_q_clk2s IS
    GENERIC (  sig_wide: positive:=8;
               fix_point: natural:=0;
               clock_div: natural:=0);
    PORT (  QUANTITY data_in: IN real;
            SIGNAL data_out: OUT std_logic_vector(sig_wide-1 DOWNTO 0);
            SIGNAL clk: IN std_logic;
            SIGNAL enable_out: OUT std_logic);
  END COMPONENT;


  ------------------------------------------------------------------------------
  --Konverter zur Umsetzung von Quantities auf Quantities
  --Erzeugung des Enable-Signals aus Abtasttakt
  --clock_div:  0 enable immer '1', sonst enable=clk/clock_div
  --data_in:    Eingangsquantity
  --data_out:   Ausgangsquantity
  --clk:        Takt-Eingang
  --enable_out: Ausgang des Enable-Signals
  ------------------------------------------------------------------------------
  COMPONENT MAM_q_clk2q IS
    GENERIC ( clock_div: natural:=0);
    PORT (     QUANTITY data_in: IN real;
               QUANTITY data_out: OUT real;
               SIGNAL clk: IN std_logic;
               SIGNAL enable_out: OUT std_logic);
  END COMPONENT;

END PACKAGE MAM_support;
```

Package Body mit Prozedurdefinitionen

```
PACKAGE BODY MAM_support IS
------------------------------------------------------------------------------
--Konstanten fuer Wandlung Terminal-Signal, Signal-Terminal
------------------------------------------------------------------------------
   CONSTANT Rs:real:=1.0;       -- Ausgangswiderstand für starke Treiber
   CONSTANT Rw:real:=250.0;     -- Ausgangswiderstand für schwache Treiber
   CONSTANT Rl:real:=1.0e9;     -- Eingangswiderstand: 0 < Rs < Rw << Rl
   CONSTANT U1:real:=5.0;       -- Ausgangsspannung 'H' und '1'
   CONSTANT U0:real:=0.0;       -- Ausgangsspannung 'L' und '0'
   CONSTANT Ux:real:=2.5;       -- Ausgangsspannung 'X', 'W', 'U' und '-'
   CONSTANT Uz:real:=-1.0;      -- 'Z'-Testspannung Uz < U0 oder Uz > U1

   CONSTANT max_drv:real:=100.0;-- Maximale Anzahl von Treibern, um bei mehrfach
                                -- getriebenen Signalen am Eingang den richtigen
                                -- Zustand wiederzuerkennen

   CONSTANT Ui1:real:=ceil(( ((Ux*Rs/(max_drv-1.0)+U1*Rs)/(Rs+Rs/(max_drv-1.0)))
                    +0.001)*1000.0) / 1000.0;
                                -- minimale Eingangsspannung, um eine '1' zu
                                -- erkennen, wenn kein Treiber den Wert 'X' besitzt

   CONSTANT Ui0:real:=floor(( ((Ux*Rs/(max_drv-1.0)+U0*Rs)/(Rs+Rs/(max_drv-1.0)))
                    -0.001)*1000.0) / 1000.0;
                                -- maximale Eingangsspannung, um eine '0' zu
                                -- erkennen, wenn kein Treiber den Wert 'X' besitzt

   CONSTANT Nw_N1:real:=Rw/Rs * (Ui1-U1)/(Ux-Ui1);
                                -- Verhältnis zwischen Treibern mit 'W' und '1' , so
                                -- daß eine '1' erkannt wird
   CONSTANT Nl_N1:real:=Rw/Rs * (Ui1-U1)/(U0-Ui1);
                                -- Verhältnis zwischen Treibern mit 'L' und '1' , so
                                -- daß eine '1' erkannt wird
   CONSTANT Nw_N0:real:=Rw/Rs * (Ui0-U0)/(Ux-Ui0);
                                -- Verhältnis zwischen Treibern mit 'W' und '0' , so
                                -- daß eine '0' erkannt wird
   CONSTANT Nh_N0:real:=Rw/Rs * (Ui0-U0)/(U1-Ui0);
                                -- Verhältnis zwischen Treibern mit 'H' und '0' , so
                                -- daß eine '0' erkannt wird
```

```
--------------------------------------------------------------------------------
--Datentypskonvertierungsfunktionen bit_vector - integer
--------------------------------------------------------------------------------
   FUNCTION bit_vector2int(constant in1: bit_vector) RETURN integer IS
     VARIABLE x:integer:=0;
   BEGIN
     FOR i IN 0 TO in1'LENGTH-1 LOOP
       IF in1(i)='1' THEN
         x:=x+2**i;
       END IF;
     END LOOP;
     RETURN x;
   END FUNCTION bit_vector2int;


   FUNCTION int2bit_vector(constant in1: IN integer;
                           constant size: IN integer:=8) RETURN bit_vector IS
     VARIABLE in_var,x:integer:=0;
     VARIABLE retval : bit_vector(size-1 DOWNTO 0):=(OTHERS=>'0');
   BEGIN
     ASSERT in1<2**size REPORT "number ot of range" SEVERITY error;
     in_var:=in1;
     FOR i IN size-1 DOWNTO 0 LOOP
       x:=2**i;
       IF in_var-x>=0 THEN
         retval(i):='1';
         in_var:=in_var-x;
       ELSE
         retval(i):='0';
       END IF;
     END LOOP;
     RETURN retval;
   END int2bit_vector;




--------------------------------------------------------------------------------
--Datentypskonvertierungsfunktionen bit - real
--------------------------------------------------------------------------------
   FUNCTION bit2real(in1:bit; low_level,high_level:real) RETURN real IS
   BEGIN
     IF in1='0' THEN
       RETURN low_level;
     ELSE
       RETURN high_level;
     END IF;
   END;


   FUNCTION real2bit(in1:real; threshold:real) RETURN bit IS
   BEGIN
     IF in1>=threshold THEN
       RETURN '1';
     ELSE
       RETURN '0';
     END IF;
   END;
```

```
--------------------------------------------------------------------------------
--Datentypskonvertierungsfunktion std_logic_vector - real
--------------------------------------------------------------------------------
   FUNCTION fix2real( CONSTANT data_in:std_logic_vector;
                      CONSTANT sig_wide:positive;
                      CONSTANT fix_point:natural) RETURN real IS
      VARIABLE ret_val:real:=0.0;
      VARIABLE neg_val:boolean:=false;
      VARIABLE data_int:std_logic_vector(sig_wide-1 DOWNTO 0);
   BEGIN
      ASSERT data_in'LENGTH=sig_wide REPORT "vector length missmatch" SEVERITY error;

      IF data_in(sig_wide-1)='1' THEN
         neg_val:=true;
         data_int:=NOT data_in;
         data_int:=std_logic_vector(unsigned(data_int)+1);
      ELSE
         data_int:=data_in;
      END IF;

      FOR i IN sig_wide-2 DOWNTO 0 LOOP
         IF data_int(i)='1' OR data_int(i)='H' THEN
            ret_val:=ret_val+2.0**(i-fix_point);
         END IF;

         ASSERT data_int(i)='1' OR data_int(i)='0' OR data_int(i)='H' OR data_int(i)='L'
              REPORT "signal contains 'U', 'X', 'W' or 'Z'" severity warning;
      END LOOP;

      IF neg_val THEN
         RETURN -ret_val;
      ELSE
         RETURN ret_val;
      END IF;
   END;
```

```
-------------------------------------------------------------------------------
--Datentypskonvertierungsfunktion real - std_logic_vector
-------------------------------------------------------------------------------
   FUNCTION real2fix( CONSTANT data_in:real;
                      CONSTANT sig_wide:positive;
                      CONSTANT fix_point:natural) RETURN std_logic_vector IS

      VARIABLE data_int,x:real:=0.0;
      VARIABLE ret_val:std_logic_vector(sig_wide-1 DOWNTO 0):=(OTHERS=>'0');
      VARIABLE neg_val:boolean:=false;
   BEGIN
      ASSERT data_in<2.0**(sig_wide-1-fix_point) AND
             data_in>=-2.0**(sig_wide-1-fix_point)
           REPORT "number out of range" SEVERITY error;

      IF data_in<0.0 THEN
         data_int:=-data_in;
         neg_val:=true;
      ELSE
         data_int:=data_in;
      END IF;

      FOR i IN sig_wide-2 DOWNTO 0 LOOP
         x:=2.0**(i-fix_point);
         IF data_int-x>=0.0 THEN
            ret_val(i):='1';
            data_int:=data_int-x;
         ELSE
            ret_val(i):='0';
         END IF;
      END LOOP;

      IF neg_val THEN
         ret_val:=NOT ret_val;
         ret_val:=std_logic_vector(signed(ret_val)+1);
         RETURN ret_val;
      ELSE
         RETURN ret_val;
      END IF;
   END;

END PACKAGE BODY MAM_support;
```

Als Modell zu instantiierende Treiber

```
-------------------------------------------------------------------------------
-- Treiber Signal(std_ulogic)->Terminal(electrical)
-------------------------------------------------------------------------------
LIBRARY disciplines,ieee,mam_support;
USE disciplines.electromagnetic_system.ALL;
USE ieee.std_logic_1164.ALL;
USE mam_support.mam_support.ALL;

ENTITY MAM_Sstdulogic2T IS
   PORT (SIGNAL s_in:IN std_ulogic; TERMINAL t1:electrical);
END;

ARCHITECTURE behav OF MAM_Sstdulogic2T IS
   TERMINAL t2:electrical;
   QUANTITY u_r ACROSS i_r THROUGH t2 TO t1;
   QUANTITY u_out ACROSS i_out THROUGH t2;
BEGIN
   IF s_in='U' OR s_in='-' OR s_in='X'    USE  u_out==Ux; u_r==i_r*Rs;
   ELSIF s_in='1'                         USE  u_out==U1; u_r==i_r*Rs;
   ELSIF  s_in='0'                        USE  u_out==U0; u_r==i_r*Rs;
   ELSIF s_in='W'                         USE  u_out==Ux; u_r==i_r*Rw;
   ELSIF s_in='L'                         USE  u_out==U0; u_r==i_r*Rw;
   ELSIF s_in='H'                         USE  u_out==U1; u_r==i_r*Rw;
   ELSE                                        i_out==0.0; u_r==0.0;
   END USE;

   BREAK ON s_in;
END;
```

```
-------------------------------------------------------------------------------
-- Treiber Signal(bit)->Terminal(electrical)
-------------------------------------------------------------------------------
LIBRARY disciplines,mam_support;
USE disciplines.electromagnetic_system.ALL;
USE mam_support.mam_support.ALL;

ENTITY MAM_Sbit2T IS
   PORT (SIGNAL s_in:IN bit; TERMINAL t1:electrical);
END;

ARCHITECTURE behav OF MAM_Sbit2T IS
   TERMINAL t2:electrical;
   QUANTITY u_r ACROSS i_r THROUGH t2 TO t1;
   QUANTITY u_out ACROSS i_out THROUGH t2;
BEGIN
   IF s_in='1' USE
      u_out==U1; u_r==i_r*Rs;
   ELSE
      u_out==U0; u_r==i_r*Rs;
   END USE;

   BREAK ON s_in;
END;
```

```
--------------------------------------------------------------------------------
-- Treiber Terminal(electrical)->Signal(std_logic)
--------------------------------------------------------------------------------
LIBRARY disciplines,ieee,mam_support;
USE disciplines.electromagnetic_system.ALL;
USE ieee.std_logic_1164.ALL;
USE mam_support.mam_support.ALL;

ENTITY MAM_T2Sstdulogic IS
   PORT (TERMINAL t1:electrical; SIGNAL s_out: OUT std_ulogic );
END;

ARCHITECTURE behav OF MAM_T2Sstdulogic IS
   TERMINAL t2:electrical;
   QUANTITY u_in ACROSS t1;
   QUANTITY u_r ACROSS i_r THROUGH t1 TO t2;
   QUANTITY u_s ACROSS i_s THROUGH t2;
   SIGNAL s1,s0,sZ:boolean;
BEGIN
   i_r==u_r/Rl;
   u_s==Uz;

   s1<=u_in'ABOVE(Ui1);
   s0<=NOT u_in'ABOVE(Ui0);
   sZ<=NOT u_in'ABOVE(0.9*Uz) WHEN Uz<Ui0 ELSE
          u_in'ABOVE(0.9*Uz) WHEN Uz>Ui1 ELSE
          false;
   s_out<= '1' WHEN s1 ELSE
          '0' WHEN s0 AND NOT sZ ELSE
          'Z' WHEN sZ ELSE
          'X';
END;


--------------------------------------------------------------------------------
-- Treiber Terminal(electrical)->Signal(bit)
--------------------------------------------------------------------------------
LIBRARY disciplines,mam_support;
USE disciplines.electromagnetic_system.ALL;
USE mam_support.mam_support.ALL;

ENTITY MAM_T2Sbit IS
   PORT (TERMINAL t1:electrical; SIGNAL s_out: INOUT bit );
END;

ARCHITECTURE behav OF MAM_T2Sbit IS
   TERMINAL t2:electrical;
   QUANTITY u_in ACROSS t1;
   QUANTITY u_r ACROSS i_r THROUGH t1 TO t2;
   QUANTITY u_s ACROSS i_s THROUGH t2;
   SIGNAL s1,s0:boolean;
BEGIN
   i_r==u_r/Rl;
   u_s==Uz;

   s1<=u_in'ABOVE(Ui1);
   s0<=NOT u_in'ABOVE(Ui0);
   s_out<= '1' WHEN s1 ELSE
          '0' WHEN s0 ELSE
          s_out;
END;
```

```
--------------------------------------------------------------------------------
-- Treiber Signal(real)->Terminal(electrical)
-- Bei Signalen von Typ integer die Konvertierungsfunktion real(integer_Wert)
-- verwenden
--------------------------------------------------------------------------------
LIBRARY disciplines;
USE disciplines.electromagnetic_system.ALL;

ENTITY MAM_Sreal2T IS
   PORT (SIGNAL s_in:IN real; TERMINAL t1:electrical);
END;

ARCHITECTURE behav OF MAM_Sreal2T IS
   QUANTITY u_out ACROSS i_out THROUGH t1;
BEGIN
   u_out==s_in;
   BREAK ON s_in;
END;
```

```
--------------------------------------------------------------------------------
-- Treiber Terminal(electrical)->Signal(real)
-- Bei Übergabe an Signale von Typ integer die Konvertierungsfunktion
-- integer(real_Wert) verwenden
-- ACHTUNG: Maximale Simulationsschrittweite beachten
-- bei getakteten Systeme MAM_T2Sreal_clk verwenden
--------------------------------------------------------------------------------
LIBRARY disciplines;
USE disciplines.electromagnetic_system.ALL;

ENTITY MAM_T2Sreal IS
   GENERIC ( delta : real := 0.1);  -- Werteänderung die einen event auslösen soll
   PORT (TERMINAL t1:electrical; SIGNAL s_out: OUT real );
END;

ARCHITECTURE behav OF MAM_T2Sreal IS
   QUANTITY u_in ACROSS t1;
   SIGNAL s_intern:real:=0.0;
   SIGNAL x1,x2: boolean;
BEGIN
   x1<=u_in'ABOVE(s_intern+delta);
   x2<=u_in'ABOVE(s_intern-delta);
   s_out<=u_in WHEN x1 OR (NOT x2);
   s_out<=s_intern;
END;
```

```
-------------------------------------------------------------------------------
-- Treiber Terminal(electrical)->Signal(real) für getaktete Systeme
-- Bei Übergabe an Signale von Typ integer die Konvertierungsfunktion
-- integer(real_Wert) verwenden
-------------------------------------------------------------------------------
LIBRARY disciplines;
USE disciplines.electromagnetic_system.ALL;

ENTITY MAM_T2Sreal_clk IS
   PORT (TERMINAL t1:electrical; SIGNAL clk : IN bit; SIGNAL s_out: OUT real );
END;

ARCHITECTURE behav OF MAM_T2Sreal_clk IS
   QUANTITY u_in ACROSS t1;
BEGIN
   PROCESS(clk)
   BEGIN
      IF clk'EVENT AND clk'LAST_VALUE='0' THEN  --L-H Flanke von clk
         s_out<=u_in;
      END IF;
   END PROCESS;
```

```
-------------------------------------------------------------------------------
--Konverter zur Umsetzung von Signalen im Festkommaformat auf Quantities
--sig_wide:   Bit-Breite des Signals, Zeierkomplementdarstellung
--fix_point:  Kommapossition 0=integer, 1=1 Nachkommastelle ...
--ramp_time:  Zeit zwischen Signalwechseln auf data_in,
--            dient zur linearen Interpolation der Quantity zwischen
--            zwei Signalwechseln
--data_in:    Eingangssignal
--data_out:   Ausgangsquantity
--enable_in:  Eingang des Enable-Signals
--enable_out: Ausgang des Enable-Signals
-------------------------------------------------------------------------------
LIBRARY ieee,mam_support;
USE ieee.std_logic_1164.ALL;
USE mam_support.mam_support.ALL;
ENTITY MAM_s2q IS
   GENERIC (  ramp_time: real:=1.0e-6;
              sig_wide: positive:=8;
              fix_point: natural:=0);
   PORT (  SIGNAL data_in: IN std_logic_vector(sig_wide-1 DOWNTO 0);
           QUANTITY data_out: OUT real;
           SIGNAL enable_in: IN std_logic;
           SIGNAL enable_out: OUT std_logic);
END;

ARCHITECTURE conv OF MAM_s2q IS
   signal data_tmp: real:=0.0;
BEGIN
   data_tmp<=fix2real(data_in,sig_wide,fix_point);
   data_out==data_tmp'ramp(ramp_time,ramp_time);

   enable_out<=enable_in;
END;
```

```
--------------------------------------------------------------------------------
--Konverter zur Umsetzung von Quantities auf Signale im Festkommaformat
--sig_wide:   Bit-Breite des Signals, Zeierkomplementdarstellung
--fix_point:  Kommapossition 0=integer, 1=1 Nachkommastelle ...
--data_in:    Eingangsquantity
--data_out:   Ausgangssignal
--enable_in:  Eingang des Enable-Signals
--enable_out: Ausgang des Enable-Signals
--Die Wandlung erfolgt mit jeder positiven Flanke des Enable-Signals
--Enable immer '1' -> keine Wandlung!
--------------------------------------------------------------------------------
LIBRARY ieee,mam_support;
USE ieee.std_logic_1164.ALL;
USE mam_support.mam_support.ALL;
ENTITY MAM_q2s IS
   GENERIC (  sig_wide: positive:=8;
              fix_point: natural:=0);
   PORT (     QUANTITY data_in: IN real;
              SIGNAL data_out: OUT std_logic_vector(sig_wide-1 DOWNTO 0);
              SIGNAL enable_in: IN std_logic;
              SIGNAL enable_out: OUT std_logic);
END;

ARCHITECTURE conv OF MAM_q2s IS
BEGIN
   PROCESS(enable_in)
   BEGIN
      IF enable_in='1' THEN
         data_out<=real2fix(data_in,sig_wide,fix_point);
      END IF;
      enable_out<=enable_in;
   END PROCESS;
END;
```

```
--------------------------------------------------------------------------------
--Konverter zur Umsetzung von Quantities auf Signale im Festkommaformat
--Erzeugung des Enable-Signals aus Abtasttakt
--sig_wide:   Bit-Breite des Signals, Zeierkomplementdarstellung
--fix_point:  Kommapossition 0=integer, 1=1 Nachkommastelle ...
--clock_div:  0 enable immer '1', sonst enable=clk/clock_div
--data_in:    Eingangsquantity
--data_out:   Ausgangssignal
--clk:        Takt-Eingang
--enable_out: Ausgang des Enable-Signals
--------------------------------------------------------------------------------
LIBRARY ieee,mam_support;
USE ieee.std_logic_1164.ALL;
USE mam_support.mam_support.ALL;
ENTITY MAM_q_clk2s IS
   GENERIC (  sig_wide: positive:=8;
              fix_point: natural:=0;
              clock_div: natural:=0);
   PORT (  QUANTITY data_in: IN real;
           SIGNAL data_out: OUT std_logic_vector(sig_wide-1 DOWNTO 0);
           SIGNAL clk: IN std_logic;
           SIGNAL enable_out: OUT std_logic);
END;

ARCHITECTURE conv OF MAM_q_clk2s IS
BEGIN
   PROCESS(clk)
      VARIABLE clk_cnt:integer:=clock_div;
      VARIABLE assign:boolean:=true;
   BEGIN
      IF clock_div=0 THEN
         enable_out<='1';
         assign:=true;
      ELSIF clock_div=1 THEN
         enable_out<=clk;
         assign:=true;
      ELSE
         IF clk='1' THEN
            IF clk_cnt=1 THEN
               enable_out<='1';
               assign:=true;
            ELSE
               enable_out<='0';
               assign:=false;
            END IF;

            clk_cnt:=clk_cnt-1;
            IF clk_cnt=0 THEN
               clk_cnt:=clock_div;
            END IF;
         END IF;
      END IF;

      IF clk='1' THEN
         IF assign THEN
            data_out<=real2fix(data_in,sig_wide,fix_point);
         END IF;
      END IF;

   END PROCESS;
END;
```

```
--------------------------------------------------------------------------------
--Konverter zur Umsetzung von Quantities auf Quantities
--Erzeugung des Enable-Signals aus Abtasttakt
--clock_div:  0 enable immer '1', sonst enable=clk/clock_div
--data_in:    Eingangsquantity
--data_out:   Ausgangsquantity
--clk:        Takt-Eingang
--enable_out: Ausgang des Enable-Signals
--------------------------------------------------------------------------------
LIBRARY ieee,mam_support;
USE ieee.std_logic_1164.ALL;
USE mam_support.mam_support.ALL;
ENTITY MAM_q_clk2q IS
   GENERIC ( clock_div: natural:=0);
   PORT (     QUANTITY data_in: IN real;
              QUANTITY data_out: OUT real;
              SIGNAL clk: IN std_logic;
              SIGNAL enable_out: OUT std_logic);
END;

ARCHITECTURE conv OF MAM_q_clk2q IS
BEGIN
  data_out==data_in;

   PROCESS(clk)
      VARIABLE clk_cnt:integer:=clock_div;
   BEGIN
      IF clock_div=0 THEN
         enable_out<='1';
      ELSIF clock_div=1 THEN
         enable_out<=clk;
      ELSE
         IF clk='1' THEN
            IF clk_cnt=1 THEN
               enable_out<='1';
            ELSE
               enable_out<='0';
            END IF;

            clk_cnt:=clk_cnt-1;
            IF clk_cnt=0 THEN
               clk_cnt:=clock_div;
            END IF;
         END IF;
      END IF;

   END PROCESS;
END;
```

Anhang C Abstraktionswandler für eine RS232-UART unter SystemC

```
#ifndef SERIAL_TRANSMIT_H
#define SERIAL_TRANSMIT_H

#include "mixsigc.h"
#include "data_conv.h"


//-----------------------------------------------------------------------------
//Parameter
//-----------------------------------------------------------------------------
int  RS232_BAUD     = 9600;
char RS232_PARITY   = 'n';
int  RS232_STOPBITS = 1;
bool RS232_nowait   = true;



//-----------------------------------------------------------------------------
//Initialisierungsfunktion
//-----------------------------------------------------------------------------
void RS232_init(sc_signal<sc_bit> *TxD, sc_signal<sc_bit> *RTS, int baud=9600,
                char par='n', int stop=1, bool nowait=true)
{
   RS232_BAUD     = baud;
   RS232_PARITY   = par;
   RS232_STOPBITS = stop;
   RS232_nowait   = nowait;

   *TxD=(sc_bit)'1';
   if (RTS)
      *RTS=(sc_bit)'1';
   wait(int(1e9/RS232_BAUD), SC_NS);
}



//-----------------------------------------------------------------------------
//Sendefunktion
//-----------------------------------------------------------------------------
void RS232_senddata( sc_uint<8> data2send, sc_signal<sc_bit> *TxD,
                     sc_signal<sc_bit> *CTS)
{
   int one_count=0;
   int delay= int(1e9/RS232_BAUD);
   sc_bv<8> data;
   bool exit=false;

   *TxD=(sc_bit)'1';
   wait(delay,SC_NS);
   sc_uint<8> data_tmp=data2send;

   exit=false;
   if (CTS && (*CTS=='0') && !RS232_nowait)
   {
      while(!exit)
      {
```

```
      wait();
      if (*CTS==’0’)
         exit=true;
   }
}
if (!CTS || *CTS==’1’)
{
//Wenn Empfaenger bereit:
//Startbit
   *TxD=(sc_bit)’0’;
   uint2bv(&data_tmp, &data);
   wait(delay,SC_NS);

//Datenbits
   for (int i=0; i<8; i++)
   {
      *TxD=(sc_bit)data[i];
      if (data[i]==’1’)
         one_count++;

      wait(delay,SC_NS);
   }

//Parity-Bit
   if (RS232_PARITY==’o’)
   {
      if (one_count % 2 == 0)
         *TxD=(sc_bit)’1’;
      else
         *TxD=(sc_bit)’0’;

      wait(delay,SC_NS);
   }
   else if (RS232_PARITY==’e’)
   {
      if (one_count % 2 == 0)
         *TxD=(sc_bit)’0’;
      else
         *TxD=(sc_bit)’1’;

      wait(delay,SC_NS);
   }

//Stop-Bit
   if (RS232_STOPBITS==1 || RS232_STOPBITS==2)
   {
      *TxD=(sc_bit)’1’;
      wait(delay,SC_NS);
   }
   else if (RS232_STOPBITS==2)
   {
      *TxD=(sc_bit)’1’;
      wait(delay,SC_NS);
   }
}
else
{
   cout<< "Receiver not ready\n";
}
}
```

```
//-----------------------------------------------------------------------------
//Empfaengerfunktion
//-----------------------------------------------------------------------------
void RS232_receivedata( sc_signal<sc_uint<8> > *data_received,
                        sc_signal<sc_bit> *RxD, sc_signal<sc_bit> *RTS)
{
   int delay=int(1e9/RS232_BAUD);
   int one_count=0;
   sc_bit parity_bit;
   sc_bv<8> data;
   sc_uint<8> tmp;
   bool exit=false;

   //Empfaenger ist bereit Daten zu empfangen
   if (RTS)
      *RTS=(sc_bit)'1';
   exit=false;

   //auf Startbit warten
   while (!exit)
   {
      wait();
      if (*RxD=='0')
         exit=true;
   }

   //Datenbits
   wait(delay/2,SC_NS);
   for (int i=0; i<8; i++)
   {
      wait(delay,SC_NS);
      data[i]=*RxD;
      if (*RxD=='1')
         one_count++;
   }

   //Parity-Bit
   if (RS232_PARITY=='o' || RS232_PARITY=='e')
   {
      wait(delay,SC_NS);
      parity_bit=RxD;
      if ( ((one_count % 2 == 0) && (RS232_PARITY=='o')) ||
           ((one_count % 2 == 1) && (RS232_PARITY=='e')) )
         cout<<"parity error\n";
   }

   //Stop-Bits
   if (RS232_STOPBITS==1 || RS232_STOPBITS==2)
      wait(delay,SC_NS);
   if (RS232_STOPBITS==2)
      wait(delay,SC_NS);

   //Empfaenger ist nicht bereit Daten zu empfangen
   if (RTS)
      *RTS=(sc_bit)'0';

   bv2uint(&data,&tmp);
   *data_received=tmp;
   wait(delay/2,SC_NS);
}
#endif
```

Anhang D A/D- und D/A-Wandler: elec_wire – sc_port für SystemC-AMS

```
#ifndef MAM_DA_AD_H
#define MAM_DA_AD_H

#include "mixsigc.h"

//------------------------------------------------------------------------------
//Parameter
//------------------------------------------------------------------------------
const double Rs=1.0;            // Ausgangswiderstand
const double Rl=1.0e9;          // Eingangswiderstand 0 < Rs < Rw << Rl
const double U1=5.0;            // Ausgangsspannung ’1’
const double U0=0.0;            // Ausgangsspannung ’0’
const double Ux=2.5;            // Ausgangsspannung ’X’
const double Uz=-1.0;           // ’Z’ Testspannung Uz < U0 oder Uz>U1

const double max_drv=100.0;     // Maximale Anzahl von Treibern auf einem Signal,
                                // um eine korrekte Auflösung bei Konflikten zu
                                // erreichen

const double Ui1=ceil(( ((Ux*Rs/(max_drv-1.0)+U1*Rs)/(Rs+Rs/(max_drv-1.0)))
                        +0.001)*1000.0) / 1000.0;
                                //minimale Eingangsspannung um eine ’1’ zu erkennen,
                                //wenn kein Eingang den Wert ’X’ hat

const double Ui0=floor(( ((Ux*Rs/(max_drv-1.0)+U0*Rs)/(Rs+Rs/(max_drv-1.0)))
                         -0.001)*1000.0) / 1000.0;
                                //minimale Eingangsspannung um eine ’0’ zu erkennen,
                                //wenn kein Eingang den Wert ’X’ hat
```

```
//-------------------------------------------------------------
//Verbindung digital => SDF
//-------------------------------------------------------------
SDF_MODULE(MAM_outdrv_SDF)
{
   sc2sdf_inport<sc_logic> digital_in;
   sdf_outport<double> sdf_out;

   SDF_CTOR(MAM_outdrv_SDF);

   void init()
   {
      sdf_out=0.0;
   }

   void sig_proc()
   {
      if (digital_in.read()=='1') sdf_out=U1;
      if (digital_in.read()=='0') sdf_out=U0;
      if (digital_in.read()=='X') sdf_out=Ux;
      if (digital_in.read()=='Z') sdf_out=U0;
   }
};
```

```
//-------------------------------------------------------------
//Verbindung SDF => digital
//-------------------------------------------------------------
SDF_MODULE(MAM_input_SDF)
{
  sdf_inport<double> sdf_in;
  sdf2sc_outport<sc_logic> digital_out;

   SDF_CTOR(MAM_input_SDF);

   void init()
   {
      digital_out=(sc_logic)'0';
   }

   void sig_proc()
   {
      if (sdf_in>Ui1) digital_out=(sc_logic)'1';
      else if (sdf_in<Ui0) digital_out=(sc_logic)'0';
      else digital_out=(sc_logic)'X';
   }
};
```

```
//-------------------------------------------------------------
//digital<=>elec_wire
//Submodul zur Zuweisung der Parameter auf sdf_ports
//-------------------------------------------------------------
SDF_MODULE(MAM_outdrv_ELEC_sub)
{
   sc2sdf_inport<sc_logic> digital_in;
   sdf_outport<double> r_val;
   sdf_outport<double> u_val;

   SDF_CTOR(MAM_outdrv_ELEC_sub);

   void init()
   {
      r_val=0.0;
      u_val=0.0;
   }

   void sig_proc()
   {
      if (digital_in.read()=='1')
      {
         u_val=U1;
         r_val=Rs;
      }
      if (digital_in.read()=='0')
      {
         u_val=U0;
         r_val=Rs;
      }
      if (digital_in.read()=='X')
      {
         u_val=Ux;
         r_val=Rs;
      }
      if (digital_in.read()=='Z')
      {
         u_val=U0;
         r_val=1e12;
      }
   }
};
```

```
//-------------------------------------------------------------
//D/A-Wandler
//-------------------------------------------------------------
ELEC_SCMODULE(MAM_outdrv_ELEC)
{
   sc_in<sc_logic> digital_in;
   elec_port elec_out;

   sdf_signal<double> r_val,u_val;
   elec_gnd   gnd;
   elec_wire  t1;

   MAM_outdrv_ELEC_sub *part1;
   Vsdf                *part2;
   Rsdf                *part3;

   //Konstruktor
   ELEC_SCTOR(MAM_outdrv_ELEC)
   {
      part1=new MAM_outdrv_ELEC_sub("MAM_outdrv_ELEC_sub");
      part1->digital_in(digital_in);
      part1->r_val(r_val);
      part1->u_val(u_val);
      part1->r_val.T(Sample_Time,SDF_SEC);

      part2=new Vsdf;
      part2->a(t1);
      part2->b(gnd);
      part2->ctl(u_val);

      part3=new Rsdf;
      part3->a(t1);
      part3->b(elec_out);
      part3->ctl(r_val);

      end_module();
   }

   //Destruktor
   MAM_outdrv_ELEC::~MAM_outdrv_ELEC()
   {
      delete part1;
      delete part2;
      delete part3;
   }
};
```

```
//-------------------------------------------------------------
//Submodul Zuweisung Z-Testspannung auf sdf_port
//-------------------------------------------------------------
SDF_MODULE(MAM_input_ELEC_sub1)
{
   sdf_outport<double> Uz_sdf;

   SDF_CTOR(MAM_input_ELEC_sub1);

   void init()     { Uz_sdf=Uz; }
   void sig_proc() { Uz_sdf=Uz; }
};



//-------------------------------------------------------------
//Submodul Zustanderkennung
//-------------------------------------------------------------
SDF_MODULE(MAM_input_ELEC_sub2)
{
   sdf_inport<double> sdf_in;
   sdf2sc_outport<sc_logic> digital_out;

   SDF_CTOR(MAM_input_ELEC_sub2);

   bool s0,s1,sZ;

   void init()
   {
      digital_out=(sc_logic)'0';
   }

   void sig_proc()
   {
      s1=sdf_in>Ui1;
      s0=!(sdf_in>Ui0);
      if (Uz<Ui0)
         sZ=!(sdf_in > 0.9*Uz);
      else
         sZ=(sdf_in > 0.9*Uz);

      if (s1)
         digital_out=(sc_logic)'1';
      else if (s0 && !sZ)
         digital_out=(sc_logic)'0';
      else if (sZ)
         digital_out=(sc_logic)'Z';
      else
         digital_out=(sc_logic)'X';
   }
};
```

```
//-------------------------------------------------------------
//A/D-Wandler
//-------------------------------------------------------------
ELEC_SCMODULE(MAM_input_ELEC)
{
   elec_port elec_in;
   sc_out<sc_logic>  digital_out;

   elec_wire t1;
   elec_gnd  gnd;
   sdf_signal<double> uz;
   elecv2sdf t2;

   R                   *part1;
   MAM_input_ELEC_sub1 *part2;
   Vsdf                *part3;
   VCVS                *part4;
   MAM_input_ELEC_sub2 *part5;

   ELEC_SCTOR(MAM_input_ELEC)
   {
      part1=new R(Rl);
      part1->a(elec_in);
      part1->b(t1);

      part2=new MAM_input_ELEC_sub1("MAM_input_ELEC_sub1");
      part2->Uz_sdf(uz);
      part2->Uz_sdf.T(Sample_Time,SDF_SEC);


      part3=new Vsdf;
      part3->a(t1);
      part3->b(gnd);
      part3->ctl(uz);

      part4=new VCVS(1);
      part4->np(t2);
      part4->nn(gnd);
      part4->ncp(elec_in);
      part4->ncn(gnd);

      part5=new MAM_input_ELEC_sub2("MAM_input_ELEC_sub2");
      part5->sdf_in(t2);
      part5->digital_out(digital_out);

      end_module();
   }

   MAM_input_ELEC::~MAM_input_ELEC()
   {
      delete part1;
      delete part2;
      delete part3;
      delete part4;
      delete part5;
   }
};

#endif
```

Anhang E Dateien auf dem beigelegten Datenträger

Verzeichnis MAM_VHDL_AMS:

Package/Package Body für MAM-Unterstützung

MAM_support.vhd
MAM_support_pb.vhd

Package/Package Body des Abstraktionskonverters für IIC-Bus

i2c_transmit.vhd
i2c_transmit_pb.vhd

Package/Package Body des Abstraktionskonverters für RS232-Kommunikation

serial_transmit.vhd
serial_transmit_pb_bit.vhd
serial_transmit_pb_stdlogic.vhd

Verzeichnis MAM_SystemC_AMS:

MAM-Interface Template-Modelle

MAM.h

einfache Datentypskonvertierungen

data_conv.h

spezielle A/D- und D/A-Wandler

mam_da_ad.h

Funktionen des Abstraktionskonverters für RS232-Kommunikation

serial_transmit.h
serial_transmit.cpp

Eigene Veröffentlichungen

Begutachtete Veröffentlichungen mit Bezug zur Multi-Architecture- und Kosten-Modellierung

[1] Schlegel, M.; Bennini, F.; Mehner, J.; Herrmann, G.; Müller D.; Dötzel, W.: *Analyzing and Simulation of MEMS in VHDL-AMS Based on Reduced Order FE-Models*. IEEE Sensors Journal 2005, Volume 5, Issue 0, ISSN 1530-437X

[2] Schlegel, M.; Herrmann, G.; Müller, D.: *Erweiterte Kostenmodellierung mit VHDL/ VHDL-AMS.* GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen Kaiserslautern; Berichte aus der Informatik, Shaker Verlag, Aachen, 24.-25. Februar 2004, ISBN 3-8322-2486-6, pp. 147-155

[3] Schlegel, M.; Herrmann, G.; Müller, D.: *Kostenmodellierung mit VHDL/VHDL-AMS.* 6. Chemnitzer Fachtagung Mikromechanik & Mikroelektronik, Chemnitz, 29.-30. Oktober 2003, ISBN 3-00-011655-9, pp. 143-147

[4] Schlegel, M.; Bennini, F.; Mehner, J.; Herrmann, G.; Müller, D.; Dötzel, W.: *Analyzing and Simulation of MEMS in VHDL-AMS Based on Reduced Order FE-Models*. IEEE Sensors 2003, Second IEEE International Conference on Sensors, Toronto, Canada, 22.-24. October 2003, ISBN 0-7803-8133-5

[5] Schlegel, M.; Herrmann, G.; Müller, D.: *Application of the „Multi Architecture Modeling" Design Method to System Level MEMS Simulation.* Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS, dtip 2003, Mandelieu-La Napoule, France, 5.-7. May 2003, ISBN 0-7803-7066-X, pp. 149-153

[6] Schlegel, M.; Herrmann, G.; Müller, D.: *A system level model in VHDL-AMS for a micromechanic vibration sensor array*. IEEE Sensors 2002, The First IEEE International Conference on Sensors, Orlando, Florida, USA, 11.-14. June 2002, ISBN 0-7803-7454-1, pp. 1208-1213

[7] Schlegel, M.; Herrmann, G.; Müller, D.: *„Multi-Architecture-Modeling" Entwurfsmethode für Mixed-Signal- und Multi-Domain-Systemsimulation.* GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen Tübingen, 25.-27. Februar 2002, Berichte aus der Informatik, Shaker Verlag, Aachen 2002, ISBN 3-8265-9859-8, pp. 18-25

[8] Schlegel, M.; Herrmann, G.; Müller, D.: *Multi Architecture Modeling Design Method for Mixed Signal and Multi Domain System Simulation – First Solutions.* The International MEMS Workshop 2001, Singapore, 4-6 July 2001, ISBN 981-04-4165-7, pp. 662-667

Sonstige Veröffentlichungen zur Multi-Architecture-Modellierung

[9] Schlegel, M.; Herrmann, G.; Müller, D.: *Multi-Architektur-Modellierung im Systementwurf.* Zwischenkolloquium des SFB379, Chemnitz, 5. November 2002

Sonstige begutachtete Veröffentlichungen

[10] Schlegel, M.; Herrmann, G.; Müller, D.: *Eine neue Hardware-Komponente zur Fuzzy-Pattern-Klassifikation.* Dresdener Arbeitstagung Schaltungs- und Systementwurf DASS'04, Dresden, 19.-20. April 2004, pp. 21-26

[11] Schlegel, M.; Franke, M.; Herrmann, G.; Müller, D.: *Eine Bibliothek spice-kompatibler Bauelementemodelle in VHDL-AMS.* ANALOG 2003, ITG-Fachbericht Nr. 177, Heilbronn, 10.-12. September 2003, ISBN 3-8007-2778-1, pp. 215-221

[12] Schlegel, M.; Herrmann, G.; Müller, D.: *Ein System-Level Modell in VHDL-AMS eines mikromechanischen Vibrationssensor-Arrays.* Scientific Reports, Journal of the University of Applied Science Mittweida, 15th International Scientific Conference Mittweida, Nr. 10, 2002, Mittweida, 7.-11. November 2002, ISSN 1437-7624, pp. 144-149

[13] Schlegel, M.; Herrmann, G.; Müller, D.: *Entwicklung eines effizienten VHDL-AMS-Modells der verlustbehafteten Leitung.* 5. Chemnitzer Fachtagung Mikrosystemtechnik - Mikromechanik und Mikroelektronik, Chemnitz, 23.-24. Oktober 2001,
ISBN 3-00-008201-8, pp12-16

[14] Schlegel, M.; Irmisch, F.: *Standard mit Zukunft: VHDL-AMS.* Elektronik Automotive, Heft September/2001, pp. 88-95, ISSN 0013-5658

[15] Schlegel, M.; Irmisch, F.: *VHDL-AMS - ein Standard mit Zukunft.* Design&Elektronik, Heft 7/2001, pp. 49-51

[16] Schlegel, M.; Müller, D.: *Systementwurf und Systemsimulation mit VHDL-AMS am Beispiel der Rundumprojektion mit 2D-Mikrospiegelarray.* 3. ITG/GI/GMM-Workshop "Multi-Nature Systems: Optoelektronische, mechatronische und andere gemischte Systeme", Hamburg-Harburg, 22. Februar 2001, ISBN 3-930400-31-6, pp. 83-89

[17] Schlegel, M.; Müller, *D.: Gesamtsystemsimulation mit VHDL-AMS am Beispiel der Rundumprojektion mit 2D-Mikrospiegelarray.* Scientific Reports, Journal of the University of Applied Science Mittweida, 14th International Scientific Conference Mittweida, Nr. 10, 2000, Mittweida, 8.-11. November 2000, ISSN 1437-7624, pp. 87-94

[18] 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.* Nanotech 2005, Anaheim CA, USA, May 2005

[19] Markert, E.; Schlegel, M.; Michel, M.; Herrmann, G.; Müller, D.: *Untersuchung der Anwendbarkeit von SystemC-AMS bei der Beschreibung von MEMS,* Tagungsband 5. GMM/ ITG/GI-Workshop Multi-Nature Systems, Dresden 18. Februar 2005, pp. 13-18

[20] 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, Oktober 2004, ISSN 0947-1413, pp. 59-64

Literaturverzeichnis

[21] Müller-Glaser, K. D.; Rauch, H.: *Ablaufstrategien und Rechnerunterstützung beim Mikrosystementwurf.* Abschlußbericht des Verbundprojektes „Untersuchungen zum Entwurf von Mikrosystemen“, Reihe: Innovationen in der Mikrosystemtechnik, Band 19, November 1994

[22] Fowler, M.: *Refactoring Home Page.* http://www.refactoring.com

[23] Huss, S. A.; John, W.; Laur, R.; et al.: *Zukünftige Schwerpunkte für die Themen: Entwurf-Simulation-Modellierung-Test*. Arbeitskreis MST-Entwurfstechnik, Dezember 2001, www.vdivde-it.de/mst/entwurf/aktivit/ doks/positionspapier.pdf

[24] Sommer, R.; Rugen-Herzig, I.; et al.: *From system specification to layout: Seamless top-down design methods for analog and mixed-signal applications.* DATE’02, Paris, March 4-8, 2002, ISBN 0-7695-1471-5

[25] Mukherjee, T.; Fedder, G. K.: *Hierarchical Mixed-Domain Circuit Simulation, Synthesis and Extraction Methodology for MEMS*. Journal of VLSI Signal Processing, No. 21, Kluwer Acadamic Publishers, Netherlands, 1999

[26] Antao, B. A. A.: *Trends in CAD of Analog ICs*. IEEE Circuits & Devices, The Optoelectronics Magazine 12, 1996

[27] Haase, J.; Schwarz, P.; Trappe, P.; Vermeiren, W.: *Erfahrungen mit VHDL-AMS bei der Simulation heterogener Systeme*. GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Frankfurt a. M., 28.-29. Februar und 1. März 2000, VDE Verlag, ISBN 3-8007-2524-X

[28] Clauß, C.; Schneider, A.; Schwarz, P.: *Schaltungssimulation mit Modelica/Dymola*. 6. GMM/ITG-Diskussionssitzung, Analog’02, Bremen, 13.-14. Mai 2002, ISBN 3-8007-2695-9

[29] Otter, M.; Elmqvist H.: *Modelica Language, Libraries, Tools, Workshop and EU-Project RealSim*. http://www.modelica.org/

[30] Grimm, C.; Heupke, W.; Waldschmidt, K.; Meise, C.: *Entwurf analog/digitaler Systeme mit SystemC.* GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Tübingen, 25.-27. Februar 2002, Berichte aus der Informatik, Shaker Verlag, Aachen 2002, ISBN 3-8265-9859-8

[31] Einwich, K.; Schwarz, P.; Grimm, C.; Waldschmidt K.: *Mixed-Signal Extensions for SystemC*. FDL’02, Marseille, France, 24.-27. September 2002

[32] Klupsch, S.; Huss, S. A.: *Abstraktionsebenen und ihre Anwendung bei der Modellierung und Simulation von heterogenen Systemen*. Vorträge des 3. ITG/GI/GMM-Workshop „Multi-Nature Systems“, Hamburg-Harburg, 22. Februar 2001, ISBN 3-930400-31-6

[33] Christen, E.; Bakalar, K.; Dewey, A. M.; Moser, E.: *Analog and Mixed-Signal Modelling Using the VHDL-AMS Language*. 36th Design Automation Conference, http://www.eda.org/analog/ftp_files/documentation/tutdac99.pdf

[34] Haase, J.; Reitz, S.; Schwarz, P.: *FEM-basierte Generierung von Verhaltensmodellen*. 4. Chemnitzer Fachtagung Mikrosystemtechnik-Mikromechanik & Mikroelektronik, 11.–12. Oktober 1999, ISBN 3-00-004902-9

[35] Huss, S. A.; Klupsch, S.; Rosenberger, R.: *Modellierung gemischt analog/digitaler Schaltungen mit VHDL-AMS*. Nachtrag zum Tagungsband, 8. GMM-Workshop Methoden und Werkzeuge zum Entwurf von Mikrosystemen, Berlin, 2.–3. Dezember 1999, ISSN 0947-1413

[36] Hanna, J. P. ; Hillman, R. G.: *A Common Basis for Mixed-Technology Micro-System Modeling*. Technical Proceedings 2nd International Conference on Modeling and Simulation of Microsystems – MSM 99, San Juan, Puerto Rico, U.S.A., 19.–21. April 1999, ISBN 0-9666135-4-6

[37] Dewey, A.; et al.: *Energy-Based Characterization of Microelectromechanical Systems (MEMS) and Component Modeling Using VHDL-AMS*. Proc. International Conference on Modeling and Simulation of Microsystems MSM 99, San Juan, Puerto Rico, U.S.A., 19.–21. April 1999, ISBN 0-9666135-4-6

[38] Ginés, A.; Peralías, E.; Rueda, A.; Madrid, N.; Seepold, R.: *A Mixed-Signal Design Reuse Methodology Based on Parametric Behavioural Models with Non-Ideal Effects*. DATE'02, Paris, March 4-8, 2002, ISBN 0-7695-1471-5

[39] Wittmann, R.; Schardein, W.; Ouali, A. I.; Darianian, M.: *A Multi-Level Design Description Language enables parameterizable RF IP Development*. 6. GMM/ITG-Diskussionssitzung, Analog'02, Bremen, 13.-14. Mai 2002, ISBN 3-8007-2695-9

[40] Thronicke, W.: *A Toolset for Reuseable Parametrization Modules*. Proceedings 2nd GI/IGT/GMM-Workshop Reuse Techniques for VLSI Design, Karlsruhe, 14. September 1998

[41] Reutter, A.; Rosenstiel, W.: *An efficient Reuse System for Digital Circuit Design*. Design, Automation and Test in Europe (DATE'99), München, 9.–12. März 1999, ISBN 0-7695-0078-1

[42] Heuschen, F.; Grimm, C.; Waldschmidt, K.: *Aspects of Reuse in the Design of Mixed-Signal Systems*. Proceedings 2nd GI/IGT/GMM-Workshop Reuse Techniques for VLSI Design, Karlsruhe, 14. September 1998

[43] Meyer zu Bexten, V.; Stürmer, A.: *Design Reuse Experiment for Analog Modules.* Proceedings 2nd GI/IGT/GMM-Workshop Reuse Techniques for VLSI Design, Karlsruhe, 14. September 1998

[44] Seepold, R.: *A Hardware Design Methodology with Special Emphasis on Reuse and Synthesis*. Shaker Verlag, Aachen, 1997, ISBN 3-8265-3417-4

[45] Bastian, J.; Haase, J. Reitz, S.: *Verhaltensbeschreibung von Systemen mit verteilten Parametern durch Ordnungsreduktion.* GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen Tübingen, 25.-27. Februar 2002, Berichte aus der Informatik, Shaker Verlag, Aachen 2002, ISBN 3-8265-9859-8

[46] Benini, F.; Mehner J.; Doetzel, W.: *A Modal Decomposition Technique for Fast Harmonic and Transient Simulation of MEMS.* The International MEMS Workshop 2001, Singapore. 4-6 July 2001, ISBN 981-04-4165-7

[47] Tay F. E. H.; Andojo, O. O.; Yung C. L.: *Backpropagation-Based Generation of Dynamic Macromodels for Transient MEMS Devices Analysis.* The International MEMS Workshop 2001, Singapore. 4-6 July 2001, ISBN 981-04-4165-7

[48] Zaman, M. H.; Bart, S. F.; et al.: *A Technique for Extraction of Macro-Models in System Level Simulation of Inertial Electro-Mechanical Micro-System*. Technical Proceedings 2nd International Conference on Modeling and Simulation of Microsystems, Puerto Rico, San Juan, April 19–21 1999, ISBN 0-9666135-4-6

[49] Ramlaoui, H.: *System-On-Chip Design*. International Workshop on IP-Based Synthesis and SoC Design, Grenoble, December 14-15, 2000

[50] Seguine, D.: *Just Add Sensor – Integrating Analog and Digital Signal Conditioning in a Programmable System on Chip*. First IEEE International Conference on Sensors, Orlando, Florida, June 12-14 2002, ISBN 0-7803-7454-1

[51] Infineon Press Release: *Samsung und Infineon stellen Systemlösungen für Smartphones vor*. http://www.infineon.com/news/press/302_045d.htm

[52] Ganesan S.; Vemuri, R.: *Behavioral Partitioning in the Synthesis of Mixed Analog-Digital Systems*. 38th Design Automation Conference, Las Vegas, Nevada, June 18-22 2001, ISBN 1-58113-297-2

[53] Lauckner, S.; Kampe, J.: *Simulation automatisch synthetisierter Analogschaltungen auf unterschiedlichen Abstraktionsniveaus.* 6. GMM/ITG-Diskussionssitzung, Analog'02, Bremen, 13.-14. Mai 2002, ISBN 3-8007-2695-9

[54] Kampe, J: *Die formale Beschreibung des Strukturentwurfes analoger Systeme*. GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Frankfurt a. M., 28.-29. Februar und 1. März 2000, VDE Verlag, ISBN 3-8007-2524-X

[55] Verhaegen W.; Gielen, G.: *Efficient DDD-based Symbolic Analysis of Large Linear Analog Circuits.* 38th Design Automation Conference, Las Vegas, Nevada, June 18-22 2001, ISBN 1-58113-297-2

[56] Moser, V.; Amann, H. P.; Pellandini, F.: *Behavioural Modelling of Analogue Systems with ABSynth.* In: Current Issues in Electronic Modeling Vol. 10; Kluwer Academic Publishers, 1997, ISBN 0-7923-9875-0

[57] Zhou, N.; Zhu, B.; Agogino, A. M.; Pister, K. S. J.: *Evolutionary Synthesis of MEMS (Microelectronic Mechanical Systems) Design.* Proceedings of ANNIE 2001, Intelligent Engineering Systems through Artificial Neural Networks, Volume 11, ASME Press

[58] Clark, J. V.; Bindel, D.; et al.: *Addressing the Needs of Complex MEMS Design*, Proceedings of the 15th IEEE MEMS Conference, Las Vegas, January 20-24, 2002, ISBN-0-7803-7187-9

[59] Heuschen, F.; Grimm, C.; Waldschmidt, K.: *Top-Down Design of Mixed-Signal Systems with KANDIS*. Proceedings SDA'98 Workshop on System Design Automation, Dresden, 30.–31. März 1998

[60] Hedrich, L.: *Structural Synthesis on Transisto Level: Analysis and Modeling Approaches.* FDL'03, Frankfurt a. M., 2003 (CD zu den Proceedings)

[61] Tang, H.; Zhang, H.; Doboli, A.: *Towards High-Level Synthesis of Analog and Mixed-Signal Systems from VHDL-AMS Specifications. A Case Study for a ΣΔAnalog-Digita Converter*. FDL'03, Frankfurt a. M., 2003, ISSN 1636-9874

[62] Lehmann, G.; Wunder, B.; Selz, M.: *Schaltungsdesign mit VHDL. Synthese Simulation und Dokumentation digitaler Schaltungen*. Franzis-Verlag, Poing, 1994, ISBN 3-7723-6163-0

[63] Walker, R.; Thomas, D.: *A Model of Design Representation and Synthesis*. 22nd Design Automation Conference, Las Vegas, 1985, ISBN:0-8186-0635-5

[64] Singh, S.: *System Level Specification in Lava*. Design, Automation and Test in Europe, DATE'03, München, 2003, ISBN 0-7695-1870-2

[65] Reetz, R.; Schneider, K.; Kropf, T.: *Verificationbench: Formale Spezifikation in VHDL*. 3. GI/ITG/GME Workshop Hardwarebeschreibungssprachen und Modellierungsparadigmen, Holzhau, Deutschland, 1997, ISSN 0947-5125

[66] Rammig, F. J.: *Systematischer Entwurf digitaler Systeme.* B. G. Teubner Verlag, Stuttgart, 1989, ISBN 3-519-02265-6

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

[68] Ecker, W.; Hofmeister, M.: *The Design Cube – A New Model for Designflow Representation*. EURODAC/VHDL'92, Hamburg, 1992

[69] Ecker, W.; Hofmeister, M.; März, S: *The Design Cube – A Model for VHDL Designflow Representation and Its Application.* Modeling, Languages and Methods in Hardware Design: An International Journal, Dordrecht, Netherlands, Kluwer Academic Publishers, 1995

[70] Ecker, W.: *A Systematics for Design Steps and their Validation*. EURODAC/VHDL'95, Brighton, England, 1995, ISBN 0-8186-7156-4

[71] Zambaldi, M.; Ecker, W.: *Ein orthogonales Schema für die Klassifikation der Modellierungsabstraktion von digitalen Systemen*. GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen Kaiserslautern, Berichte aus der Informatik, Shaker Verlag, Aachen, 2004, ISBN 3-8322-2486-6

[72] RASSP, VHDL Modeling Terminology and Taxonomy: http://www.eda.org/rassp/

[73] Christen, E.; Bakalar, K.: *VHDL 1076.1: Analog and Mixed-Signal Extensions to VHDL*. In: Analog and Mixed-Signal Hardware Description Languages. Current Issues in Electronic Modeling Vol. 10; Kluwer Academic Publishers, 1997, ISBN 0-7923-9875-0

[74] Trihy, R.: *Analog Extensions to Verilog.* In: Analog and Mixed-Signal Hardware Description Languages. Current Issues in Electronic Modeling Vol. 10; Kluwer Academic Publishers, 1997, ISBN 0-7923-9875-0

[75] http://www.systemc.org/

[76] Grimm, C.; Oehler, P.; Meise, C.; Waldschmidt, K.: *Modellierung analoger Leistungselektronik auf Systemebene mit C++*. ITG-Fachbericht 164, Entwurf Integrierter Schaltungen, 10. E.I.S.-Workshop, Dresden, 2001, ISBN 3-8007-2608-4

[77] The MathWorks: http://www.mathworks.de/

[78] Ansoft Corporation: http://www.ansoft.com/

[79] Pelz, G.: *Entwurfs-Methodik für Automobil-Elektronik.* GI/ITG/GMM-Workshop: GI/ ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen Kaiserslautern; Berichte aus der Informatik, Shaker Verlag, Aachen, 2004, ISBN 3-8322-2486-6

[80] Siegmund, R.: *Ein Verfahren zur Spezifkation von Protokollen für die Verifkation und Synthese protokollzentrischer Systeme.* Dissertation, TU Chemnitz, eingereicht 2004

[81] Noullet, A. F.; Healey, F; et al.: *VerilogAMS language used in the Top-Down Methodology for wireless integrated circuit designs*. FDL'03, Frankfurt a. M., 2003, ISSN 1636-9874

[82] Herve, Y.: *Simple Models for Complex Systems: A-FSM Template*. FDL'03, Frankfurt a. M., 2003, ISSN 1636-9874

[83] Huss, S. A.; Klupsch, S.: *A new approach to mixed-signal model refinement by code refactoring methods*. FDL'03, Frankfurt a. M., 2003 (CD zu den Proceedings)

[84] The Institute of Electrical and Electronics Engineers, Inc.: *IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS changes).* IEEE Std 1076.1, 1999

[85] Ashenden, P. J.; Peterson, G. D.; Teegarden, D. A.: *The System Designer's Guide to VHDL-AMS*. Morgan Kaufmann Publishers, 2002, ISBN 1558607498

[86] Herrmann, G.; Müller, D.: *ASIC–Entwurf und Test*. Fachbuchverlag Leipzig, 2004, ISBN 3-446-21709-6

[87] Hansen, C.; Bringmann, O.;Rosenstiel, W.: *VHDL Component Model for Mixed Abstraction Level Simulation and Behavioral Synthesis*. Forum on Design Languages (FDL'99), Lyon, Frankreich, 1999

[88] Budkowski, S.; Dembinski, P.: *An Introduction to ESTELLE: A Specification Language for Distributed Systems.* Computer Networks and ISDN Systems, vol. 14, no. 1, 1987

[89] Wytrebowicz, J.; Budkowski, S.: *Communication Protocols Implementation in Hardware: VHDL Generation from Estelle.* In High-Level System Modeling: Specification Languages. Current Issues in Electronic Modeling Vol. 3; Kluwer Academic Publishers, 1995, ISBN 0-7923-9632-4

[90] Daveau, J. M.; Marchioro, G. F.; Ben-Ismail, T.; Jerraya, A. A.: *COSMOS: An SDL Based Hardware/Software Codesign Environment.* In: Hardware/Software Co-Design and Co-Verification. Current Issues in Electronic Modeling Vol. 8; Kluwer Academic Publishers, 1997, ISBN 0-7923-9689-8

[91] Vial, C.; Rouzeyre, B.: Hardware/Software Co-Synthesis: *Modeling and Synthesis of Interfaces Using Interpreted Petri Nets.* In: Hardware/Software Co-Design and Co-Verification. Current Issues in Electronic Modeling Vol. 8; Kluwer Academic Publishers, 1997, ISBN 0-7923-9689-8

[92] Thieser, M.: *PC-Schnittstellen*. Franzis-Verlag, München, 1994, ISBN 3-7723-4942-0

[93] Rosenberger, R.; Huss, S. A.: *A Systems Theoretic Approach to Behavioral Modeling and Simulation of Analog Functional Blocks*. Design, Automation, and Test in Europe DATE'98, Paris, 1998, ISBN 0-8186-8359-7

[94] Schubert, M.: *VHDL-basierte Mixed-Signal-Simulation.* Elektronik, Heft 18/2003, ISSN 0013-5658

[95] Dewey, A.: *Application of Discrete Event Hardware Description Languages to the Design and Documentation of Electronic Analog Systems.* In: Analog and Mixed-Signal Hardware Description Languages. Current Issues in Electronic Modeling Vol. 10; Kluwer Academic Publishers, 1997, ISBN 0-7923-9875-0

[96] Schlegel, M.; Sporbert, R.; Reinert, D.: *Abschlussbericht Verbundprojekt UNISIM*. Forschungsbericht der Fachhochschule für Technik und Wirtschaft Mittweida, 2000

[97] John, R.; Schaub, R.: *Werkzeugschwingungen im Ergebnis des Spanungsprozesses*. Dissertation, Technische Hochschule Karl-Marx-Stadt, 1973

[98] Strackeljan, J.; Behr, D.: C*ondition Monitoring of Rotating Machinery using a FuzzyPattern Recognition Algorithm*. 6. International Conference on Vibrations in Rotating Machinery, 9.-12. September, 1996, ISBN 1-86058-009-2

[99] Lipovszky, G.; Solyomvari, K.; Varga, G.: *Vibration Testing of Machines and their Maintenance*. Elsevier Science Publishers, Amsterdam 1990, ISBN 0-444-98808-4

[100] Mehner, J.; Scheibner, D.; Wibbeler, J.: *Silicon Vibration Sensor Arrays with Electrically Tunable Band Selectivity.* MICRO SYSTEM Technologies 2001, Düsseldorf, March 27-29, 2001, ISBN 3-8007-2601-7

[101] Eichhorn, K.: *Entwurf und Anwendung von ASICs für musterbasierte Fuzzy-Klassifikationsverfahren.* Dissertation, Technische Univerität Chemnitz, 21. März 2000

[102] Bocklisch, S. F.: *Prozeßanalyse mit unscharfen Verfahren*. Verlag Technik, Berlin 1987, ISBN 3-341-00211-1

[103] Romanowicz, B.; Ansel, Y.; et al.: *VHDL-1076.1 Modeling Examples for Microsystem Simulation*. In: Current Issues in Electronic Modeling Vol. 10; Kluwer Academic Publishers, 1997, ISBN 0-7923-9875-0

[104] Bennini, F.; Mehner, J.; Dötzel, W.: *Computational Methods for Reduced Order Modeling of Coupled Domain Simulations*. 11. International Conference on Solid State Sensors and Actuators (Transducers 01), Germany, 2001, ISBN 3-540-42150-5

[105] OSCI: *Functional Specification for SystemC 2.0*. http://www.systemc.org

[106] Müller, W.; Rosenstiel, W.; Ruf, H.: *SystemC Methodologies and Applications*. Kluwer Academic Publishers, Dordrecht, 2003, ISBN 1-4020-7479-4

[107] OSCI: *SystemC Version 2.0 Users Guide*. http://www.systemc.org

[108] Einwich, K.; Eichler, U.: *MixSigC Syntax version 0.8*. Fraunhofer Institut für Integrierte Schaltungen, Dokumentation, FhG IIS/EAS Dresden, 12. 8. 2003

[109] Einwich, K.: *SystemC-AMS Steps towards an Implementation.* FDL'03, Frankfurt a. M., 2003, ISSN 1636-987

[110] Michel, M.: *Untersuchung der Möglichkeit zur Beschreibung und Simulation heterogener Systeme unter SystemC-AMS im Vergleich zu VHDL-AMS*. Diplomarbeit an der TU Chemnitz, 2004

[111] Radetzki, M.; Putzke-Röming, W.; Nebel, W.: OO-VHDL: *What Is It, and Why Do We Need It?* Asia-Pacific Conference on Hardware Description Languages, APCHDL '97, Hsinchu, Taiwan, 1997

[112] Schumacher, G.; Nebel, W.: *Inheritance Concept for Signals in Object-Oriented Extensions to VHDL*. Proceedings of European Design Automation Conference, Brighton, England, 1995, ISBN 0-8186-7156-4

[113] Swanny, S.; Molin, A.; Govnot, B.: *OO-VHDL: Object-oriented extensions to VHDL*, IEEE Computer, vol. 28, no.10, October 1995

[114] Zippelius, R.; Müller-Glaser, K. D.: *An Object-oriented Extension of VHDL*. Proceedings of the Spring '92 Meeting of the VHDL-Forum for CAD, 1992

[115] Ecker, W.; Preis, V.; Schneider, C.: *Object Orientation and Structural Design*. In: Object-Oriented Modeling, Current Issues in Electronic Modeling Vol. 7; Kluwer Academic Publishers, 1996, ISBN 0-7923-9688-X

[116] Schumacher, G.; Nebel, W.: *Abstract Hardware Modelling using an Object-Oriented Language Extension to VHDL*. In: Object-Oriented Modeling, Current Issues in Electronic Modeling Vol. 7; Kluwer Academic Publishers, 1996, ISBN 0-7923-9688-X

[117] Schumacher, G.; Nebel, W.: *Survey on Languages for Object Oriented Hardware Design Methodologies*. In: High-Level System Modeling: Specification Languages. Current Issues in Electronic Modeling Vol. 3; Kluwer Academic Publishers, 1995, ISBN 0-7923-9632-4

[118] Siegmund, R.; Müller, D.: *SystemC^{SV}-Extension of SystemC for Mixed Multi-Level Communication Modeling and Interface-based System Design*. Design, Automation and Test in Europe DATE'01, Munich, Germany, 13-16 March 2001, ISBN 0-7695-0993-2

[119] Brem, D.: *Automatische Konvertierung von VHDL+ Systemspezifikationen nach SystemC*. Diplomarbeit, TU Chemnitz, 2001

[120] Proß, U.: *Modellierung und Systemsimulation des Universal Serial Bus Protokolls mit SystemC^{SV}*. Diplomarbeit, TU Chemnitz, 20024

[121] Teich, J.: *Digitale Hardware/Software-Systeme: Synthese und Optimierung*. Springer Verlag, 1997, ISBN 3-540-62433-3

[122] Heuschen, F.; Grimm, C.; Waldschmidt, K.: *Modellierung des Implementationsraumes im Analog/Digital Co-Design*. 3. ITG/GI/GMM-Workshop Rechnergesteuerter Schaltungs- und Systementwurf, Forschungs-Report Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Systemen und Schaltungen, Frankfurt a. M., 28.–29. Februar, 1. März 2000, ISBN 3-8007-2524-X

[123] Stammermann, A.; u. a.: *ORINOCO: Verlustleistungsanalyse und Optimierung auf der algorithmischen Abstraktionsebene*. 10. E.I.S.-Workshop, Dresden, 2001, ISBN 3-8007-2608-4

[124] Schmitt, H.: *High-Level Bewertung der Leistungsaufnahme sequentieller Schaltungen*. 10. E.I.S.-Workshop, Dresden, 2001, ISBN 3-8007-2608-4

[125] Schneider, A.; Schneider, P.; Bastian, J.: *MOSCITO-Ein modulares, internetbasiertes Programmsystem für die Optimierung von Mikrosystemen.* Statusseminar zum Verbundprojekt OMID Optimierung von Mikrosystemen für Diagnose- und Überwachungsanwendung, Bremen, 2001

[126] Peters, D.; Bolte, H.; Laur, R.: *Designoptimierung von Mikrosystemen mit MODOS*. Statusseminar zum Verbundprojekt OMID Optimierung von Mikrosystemen für Diagnose- und Überwachungsanwendung, Bremen, 2001

[127] Scholz, G.; Wilkes, W., Schlageter, G.: *A Methodology for the Specification of Transformations between Information Models*. In: Meta-Modeling : Performance and Information Modeling. Current Issues in Electronic Modeling Vol. 6; Kluwer Academic Publishers, 1996, ISBN 0-7923-9687-1

[128] Kumar, S.; Aylor; J. H.: et el.: *A Model for Exploring Hardware/Software Trade-Offs and Evaluating Design Alternatives.* In: Hardware/Software Co-Design and Co-Verification. Current Issues in Electronic Modeling Vol. 8; Kluwer Academic Publishers, 1997, ISBN 0-7923-9689-8

[129] Jakob, W.; Knorr, B.; Parodat, S.; Peters, D.; Uhlig, A.: *Optimierung von Mikrosystemen*. 8. GMM-Workshop Methoden und Werkzeuge zum Entwurf von Mikrosystemen, Berlin, 2.-3. Dezember 1999, ISSN 0947-1413

[130] Hartmann, R.; Palotas, L.: *Optimierungsverfahren für die Modellierung von Mikrosystemtechnikkomponenten*. 8. GMM-Workshop Methoden und Werkzeuge zum Entwurf von Mikrosystemen, Berlin, 2.-3. Dezember 1999, ISSN 0947-1413, S. 275–279

[131] Sporer, M.; Agsteiner, K.; Monjau, D.; Schwaar, M.: *Knowledge Based Specification and Modelling of Embedded Systems.* EUROMICRO'99, Workshop on Digital System Design, 8.-10. September 1999Milan, Italy, ISBN 0-7695-0321-7

[132] Heuschen, F.; Waldschmidt, K.: *Analog Digital Co-Design für signalverarbeitende eingebettete Systeme*. ITG-Fachbericht 164, Entwurf Integrierter Schaltungen, 10. E.I.S.-Workshop, Dresden, 3.-5. April 2001, ISBN 3-8007-2608-4

[133] Lippman, S. B.: *C++ Einführung und Leitfaden.* Addison-Wesley, 1995, ISBN 3-89319-375-8

[134] Vachoux, A.; Grimm C.; Einwich, K.: *Towards Analog and Mixed-Signal SoC Design with SystemC-AMS*. IEEE International Workshop on Electronic Design, Test and Application, DELTA 2004, Perth, Australia, 28-30 January 2004, ISBN 0-7695-2081-2

[135] Wikipedia: *Die freie Enzyklopädie*. http://de.wikipedia.org/wiki/Hauptseite

[136] Thomas, D. E.; Moorby, P. R.: *The Verilog Hardware Description Language.* Kluwer Academic Publishers, 1996, ISBN 0-7923-9723-1

[137] Schwarz, P.; Clauß, C.; Einwich, K.; Knöchel, U.; Matz, K.: *Hybride Simulation nachrichtentechnischer Systeme.* Simulationstechnik. 12. Symposium, Zürich, 1998

Verzeichnis der Abbildungen, Tabellen und Quellcodes

Abbildungen

Tabellen

Quellcodes

Selbständigkeitserklärung

Versicherung

Hiermit versichere ich, daß ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistungen von folgenden Personen erhalten:

keine

Weitere Personen waren an der Abfassung der vorliegenden Arbeit nicht beteiligt. Die Hilfe eines Promotionsberaters habe ich nicht in Anspruch genommen. Weitere Personen haben von mir keine geldwerten Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Chemnitz, den 28. Februar 2005 Michael Schlegel

Thesen

1. Die Gesamtsimulation großer, heterogener Systeme ist auf Grund begrenzter Speicher- und CPU-Ressourcen nicht oder nur sehr eingeschränkt möglich. Daher ist es notwendig, den Abstraktionsgrad der Modelle zu erhöhen, womit in der Regel aber auch eine Verringerung der Modellgenauigkeit einher geht.

2. Für eine schnelle und in den zu verifizierenden Teilsystemen genaue Simulation ist es vorteilhaft, für die Modelle der Systemkomponenten und Teilsysteme verschieden abstrakte und damit verschieden genaue und unterschiedlich simulationsaufwendige Modelle (Multiple Architekturen) zu hinterlegen. Das Gesamtsystemmodell kann dann mit diesen Teilmodellen partiell zwischen Genauigkeit und Geschwindigkeit skaliert werden.

3. Verfügen die Modelle des zu simulierenden Systems unabhängig von ihrem Abstraktionsgrad über einheitliche Schnittstellen, so können die abstrakten Modelle der Systemsimulation als Testbench für verfeinerte Modelle dienen. Eine separate Testbench kann dadurch unter Umständen entfallen, was dem Entwerfer wiederum Arbeitszeit spart.

4. Der Abstraktionsgrad eines Modelle bedingt in der Regel den Abstraktionsgrad der Schnittstelle des Modells. Unterschiedlich abstrakte Modelle lassen sich daher nicht oder nur schwer miteinander verbinden, da ihre Schnittstellen unterschiedlich abstrakt sind. Dies verursacht im Systementwurf unnötige, zeitaufwendige und fehleranfällige Anpassungsschritte. Dies gilt insbesondere dann, wenn sich die Abstraktion des Systems oder von Teilsystemen während des Entwurfsprozesses in mehreren Zyklen zwischen abstrakt und detailliert ändert.

5. Für einen effektiven Entwurfsprozeß ist es von großer Bedeutung, zu Beginn des Entwurfs einheitliche Schnittstellen für Teilsysteme und Komponenten zu definieren, soweit dies zu diesem Zeitpunkt des Entwurfsprozesses möglich ist.

6. Im Rahmen eines Top-Down-Entwurfes entstehen üblicherweise zu Beginn abstrakte Modelle mit abstrakten Schnittstellen. Die Beibehaltung dieser Schnittstellen für später entstehende, weniger abstrakte Modelle ist bei der Modellierung mit nicht objektorientierten Hardware- oder Simulationsbeschreibungssprachen in vielen Fällen nicht möglich, da die abstrakten Schnittstellen oft nicht in der Lage sind, Informationen oder Zustände zu übertragen, die erst auf weniger abstrakten Abstraktionsebenen eine Rolle spielen. Dies gilt sowohl für digitale als auch für analoge elektrische sowie nichtelektrische Schnittstellen.

7. Zur Schaffung einheitlicher Schnittstellen ist bereits auf hoher Abstraktionsebene abzuschätzen, bis auf welches Abstraktionsniveau der Entwurf der einzelnen Komponenten und Teilsysteme notwendig sein wird. Ausgehend von diesen Informationen können auf hoher Abstraktionsebene Schnittstellen geschaffen werden, deren Verwendung auf hoher Abstraktionsebene nur einen geringen bis mäßigen Modellierungs- und Simulationsmehraufwand hervorrufen, die aber auch für die Kommunikation weniger abstrakter Modelle geeignet sind. Bei diesem zusätzlichen Schritt im Entwurfsprozeß ist nicht nur der Abstraktionsgrad der gerade betrachteten Komponenten, sondern auch der Abstraktionsgrad der umgebenden

Komponenten zu berücksichtigen. Das Verhalten der Modelle wird entsprechend des jeweiligen Entwurfsschrittes unabhängig vom Abstraktionsgrad der Schnittstelle modelliert.

8. Die Verwendung detaillierter Schnittstellen bereits auf hoher Abstraktionsebene widerspricht dem allgemeinen Trend zur generellen Erhöhung des Abstraktionsniveaus von Modellen zur Beschleunigung der Simulation. Es konnte jedoch gezeigt werden, daß sich die Simulationszeit, abhängig vom verwendeten Simulator und der Art der Schnittstelle, gar nicht oder nur moderat und nur in wenigen Fällen deutlich erhöht.

9. Die Bestimmung der detaillierten Schnittstellen eines Systems ist insbesondere bei analogen, mixed-signal oder heterogenen Systemen oft möglich und praktikabel. Ist eine Bestimmung der detaillierten Schnittstellen auf hoher Abstraktionsebene nicht möglich oder mit einem erheblichen Aufwand verbunden, z. B. bei großen, komplexen digitalen Systemen, können die betroffenen Teilsysteme weiter nach der klassischen Top-Down- oder Meet-In-The-Middle-Strategie entworfen und deren Schnittstellen später im Entwurfsprozeß angepaßt werden.

10. Die MAM läßt sich auf verschiedene Hardware- und Systembeschreibungssprachen sowie auf unterschiedliche Simulationswerkzeuge anwenden, was anhand der Sprachen VHDL-AMS und SystemC-AMS demonstriert wurde. Die Verwendung objektorientierte Sprachen bietet dabei einen größeren Anwendungsraum für die Nutzung der MAM.

11. Die Verwendung einheitlicher Schnittstellen über Abstraktionsebenen hinweg ermöglicht die Ausstattung der Modelle mit sekundären Eigenschaften wie z. B. Kostenparametern, die mit fortschreitendem Entwurf stetig präzisiert werden können. Dadurch wird eine permanente Optimierung bezüglich solcher Parameter im Entwurfsprozeß ermöglicht.

12. Die Methode der Multi-Architecture-Modellierung – die Beschreibung des Systemverhaltens auf der Basis verschieden abstrakter Modelle mit vereinheitlichter Schnittstelle – ist leicht anwendbar, ermöglicht eine flexible Modellgestaltung, erzeugt nur begrenzten Modellierungs- und Simulationsmehraufwand.

Lebenslauf

Persönliche Daten

Vor- und Zuname: Michael Schlegel

Geburtsdatum: 07. 05. 1974

Schulausbildung:

1980-1990 Zehnklassige allgemeinbildende polytechnische Oberschule
Abschluß: mittlere Reife mit Prädikat „sehr gut“

1990-1992 Erweiterte Oberschule
Abschluß: Abitur mit Durchschnittsnote 1,5

Zivildienst:

1992-1993 im Klinikum „Küchwald“ (städtische Kliniken Chemnitz)

Studium:

1993-1998 Fachhochschule für Technik und Wirtschaft Mittweida
Studium der Mikrosystemtechnik

März bis Juli 1996 berufspraktisches Semester bei der „Gesellschaft für Mikroelektronikanwendungen Chemnitz mbH“

1994-1996 Anstellung als studentische Hilfskraft an der FH Mittweida

1998 Abschluß als Dipl.-Ing. (FH) für Mikrosystemtechnik mit dem Prädikat „Mit Auszeichnung“ (Durchschnittsnote 1,2)

Berufliche und akademische Laufbahn:

1998-1999 Mitarbeit im Verbundprojekt UNISIM zwischen der Fachhochschule Mittweida und der Firma SIMEC GmbH in Chemnitz,
Entwurf und Programmierung eines VHDL-AMS Compilers und Simulators

2000-2003 Weiterführung der Arbeiten zum VHDL-AMS Compiler bei der SIMEC GmbH (seit 2002 Ansoft Corp.)

2000-2004 Wissenschaftlicher Mitarbeiter an der Professur für Schaltungs- und Systementwurf der TU Chemnitz,
Bearbeitung von Aufgaben im Rahmen des SFB 379 zur Modellierung und Simulation heterogener Systeme mit VHDL-AMS und zur Entwurfsmethodik heterogener Systeme sowie die Entwicklung digitaler Hardwarekomponenten
Lehre: Vorlesung Systementwurf II – Entwurf heterogener Systeme mit VHDL-AMS

ab 2005 Übernahme der Entwicklungsabteilung der emc Elektronik und Mechanik GmbH