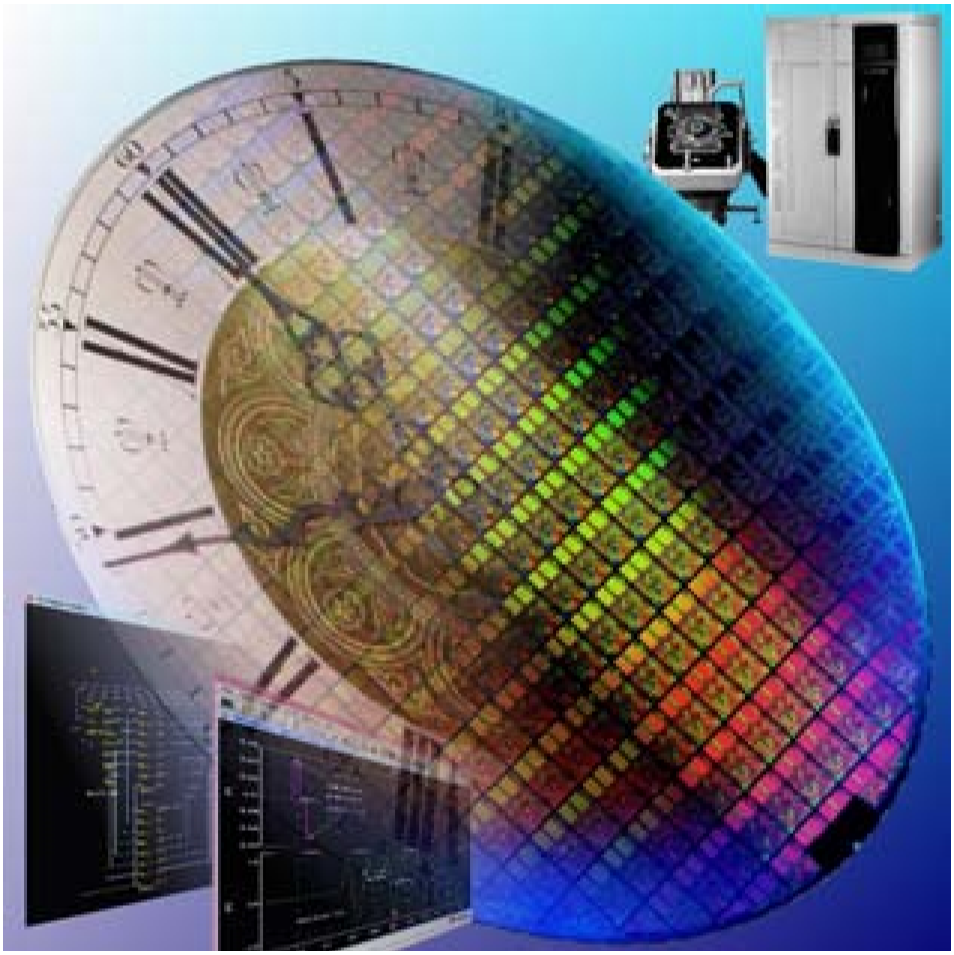
Dipl.-Ing. Harald Grams

# Modellierung und Simulation von Testautomaten für integrierte Schaltungen



Cuvillier Verlag Göttingen

# Modellierung und Simulation von Testautomaten für integrierte Schaltungen

Der Technischen Fakultät

der Unitversität Erlangen-Nürnberg

zur Erlangung des Grades

Doktor-Ingenieur

vorgelegt von

**Harald Grams**

**Erlangen - 2004**

**Bibliografische Information Der Deutschen Bibliothek**
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.
1. Aufl. - Göttingen : Cuvillier, 2004
Zugl.: Erlangen-Nürnberg, Univ., Diss., 2004
ISBN 3-86537-229-5

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

| | |
|---|---|
| Tag der Einreichung: | 28. Mai 2004 |
| Tag der Promotion: | 27. Juli 2004 |
| Dekan: | Prof. Dr. rer. nat. Albrecht Winnacker |
| Berichterstatter: | Prof. Dr.-Ing. Wolfram H. Glauert, |
| | Prof. Dr.-Ing. Dr.-Ing. habil. Robert Weigel |


Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de


1. Auflage, 2004
Gedruckt auf säurefreiem Papier

ISBN 3-86537-229-5

## Danksagung

Die vorliegende Arbeit ist im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Rechnergestützten Schaltungsentwurf der Friedrich-Alexander-Universität Erlangen-Nürnberg entstanden.

Ich bedanke mich an dieser Stelle herzlich bei Herrn Prof. Dr.-Ing. Wolfram H. Glauert für die Unterstützung, Betreuung und Begutachtung dieser Dissertation. Für die Übernahme des Zweitgutachtens geht mein Dank an Herrn Prof. Dr.-Ing. Dr.-Ing. habil. Robert Weigel. Mein weiterer Dank gilt Herrn Prof. Dr.-Ing. Reinhard German für die Übernahme des Amtes als Fachfremder Prüfer und Herrn Prof. Dr.-Ing. Lorenz-Peter Schmidt für die Übernahme des Prüfungsvorstizes.

Bei Herrn Dr. Wilfried Tenten von der Robert Bosch GmbH bedanke ich mich für sein Engagement und die Leitung des Projektes VIRTUS, das vom BMB+F gefördert wurde. Des weiteren geht mein Dank an Frau Waltraud Hartl und die Herren Georg Voit, Andreas Kasberger und Mike Hayn, die mich während der Kooperation mit der SZ Testsysteme AG hilfsbereit und kompetent mit Daten und Informationen unterstützten, sowie für die Diskussionen, die ich mit Ihnen führen durfte, und für die detaillierten Einblicke, die sie mir in die Produktionstestmaschinen gaben.

Ganz besonders will ich mich bei den Sekretärinnen Frau Barbara Becker und Frau Roswitha Rauch für ihren Einsatz, ihre Unterstützung bei allen organisatorischen Fragen sowie für ihr Engagement in der Endphase meiner Promotion bedanken.

*„Das Große Los blüht uns nicht oft,*
*wie man's auch dreht, nimmt, zieht und hofft."*
*[Heinz Erhardt]*

Abschließend geht ein ganz besonderer Dank an meine Kollegen Heiko, Klaus und Thomas, die mir in den vergangenen Jahren am Lehstuhl mehr als „nur" Kollegen waren und mit denen ich als Freunde und Kollegen mein Großes Los gezogen habe.

Stein, September 2004 Harald Grams

## Kurzfassung

Diese Arbeit beschreibt Modellierung und Simulation von Testautomaten, um Prüfprogramme zu simulieren und neue Architekturkonzepte für Testsysteme zu bewerten. Dazu werden zunächst Hard- und Software-Komponenten für derzeitig verfügbare Testsysteme analysiert. Die Analyse zeigt, daß die evolutionäre Weiterentwicklung der Hardware hin zu immer flexibleren Systemen zu Problemen bei der Modellierung und Programmierung der Instrumente führt. Die Untersuchung verfügbarer Virtual-Test-Produkte ergab, daß ein allgemeiner Ansatz fehlt, der auf die gesamte Aufgabe des Testingenieurs abzielt. Die in der Arbeit geschaffene virtuelle Testumgebung bietet die Möglichkeit der Prüfprogrammsimulation auf verschiedenen Abstraktionsebenen. Es wurde ein Verfahren zum netzwerkweiten Datenaustausch entwickelt, der nebenläufige Prozesse zeitgenau synchronisiert. Am Beispiel eines digitalen Pins werden Lösungswege gezeigt, wie komplexes Instrumentenverhalten abstrahiert wird, um zu schnellen Simulationsmodellen zu gelangen. Um dem Ziel, schnell und effektiv Testprogramme zu schreiben [ITR03], mit Hilfe einer system-unabhängigen High-Level-Sprache näherzukommen, ist aus heutiger Sicht das Entfeinern der Hardware nötig. Dazu wird ein Modellierungskonzept zur Entwicklung einer neuen Testsystemarchitektur vorgestellt, das das Zusammenspiel verschiedener Komponenten innerhalb einer Architektur verifiziert und verschiedene Architekturkonzepte miteinander vergleicht.

## Abstract

This work describes modelling and simulation of test systems for test program simulation and evaluation of new test system architectures. For this purpose hard- and software components of actual test systems were analyzed. This shows a problem in modelling and programming these instruments, because of the great flexibility caused by the evolutionary advancement of these systems. The analysis of commercial virtual test products shows the absence of a common approach, including the complete job of a test engineer. The presented virtual test environment provides an opportunity to simulate test programs on different abstraction levels. A method for a network wide date exchange was developed, which synchronizes concurrent processes. A digital pin gives the example in which way a complex behaviour of an instrument can be abstracted, to get fast simulation. Hardware should be simplified, to reach the goal of fast and effective test program development [ITR03]. Therefor a concept for modelling a new test system architecture will be introduced. It allows to verify the interaction of test system components and the comparison of different architecture concepts.

# Inhaltsverzeichnis

# Kapitel 1

# Einführung

## 1.1 Motivation

Zwei Hauptprobleme, denen sich die Halbleiterindustrie gegenüber sieht, sind fehlerfreie Designs sicherzustellen und defektfreie Chips zu liefern. Diese Probleme haben sich mit den jährlich zunehmenden Designgrößen nicht verringert. Design und Technologie geben Komplexität, Leistungsfähigkeit, Packungsdichte und Größe kommender IC-Generationen vor, deshalb ist es notwendig, neue Wege bei der Design-Verifikation, der Testprogrammentwicklung und der Architektur von Testsystemen einzuschlagen. Das Test Technology Technical Council (tttc) hat diese Problematik erkannt und deshalb Arbeitskreise (Technical Activity Comitee - TAC) gebildet, die sich mit „Virtual Test“ sowie mit „Verifikation und Test“ beschäftigen [TAC99]:

> „Although research in test and verification is exploring new directions which can deal with the complexities of emerging designs, the research communities in these areas have interacted very little. New techniques developed for verification are not applied to test, and sometimes long understood test strategies are not considered for formal verification. The goal of this TAC is to bring together the verification and test communities.“

Simulation ist das am häufigsten verwendete Verfahren, um heute die Richtigkeit eines Designs zu überprüfen. Dies muß der Ansatzpunkt sein, an dem zum ersten Mal Design, Verifikation und Test zusammenarbeiten. Dieses Zusammenspiel soll die Entwicklung von Werkzeugen zur Testgenerierung anspornen.

Die Überprüfung der Testprogramme kann derzeit erst erfolgen, sobald das „erste Silizium“, d.h. der erste Prototyp der Schaltung bereitsteht. Hier besteht noch Handlungsbedarf, an dem

das Konzept des virtuellen Tests ansetzt. Der Testingenieur soll nach Möglichkeit seine Arbeit parallel zum Designingenieur starten und seine Ergebnisse in die Entwicklung der Schaltung einbringen. Dazu sind für den Testingenieur eben solche Simulationsumgebungen für seine Arbeit verfügbar zu machen, wie sie heutzutage bereits für den Designingenieur existieren. Nur so lassen sich in Zukunft testbare Schaltungen entwickeln, die damit den hohen Qualitätsanforderungen genügen.

Der „Virtuelle Test“ stellt einen wichtigen Schritt zur Beschleunigung des Entwurfsprozesses und der Sicherung der Qualität integrierter Schaltungen dar. Diesem Vorgang wird in Zukunft noch eine viel größere Bedeutung zukommen, denn die Trends in der Halbleiterindustrie gehen aus Kostengründen weg von teuren und starren Universal-Testsystemen hin zu modularen Testsystemen, die je nach Kunde und Anwendung applikationsspezifisch zusammengestellt werden können. Diese Testsysteme müssen von der Architektur her optimal gestaltet sein, so daß umfangreiche Anpassungsarbeiten - wie bisher notwendig -, insbesondere bei der Betriebssoftware, Selbsttest und Kalibrierung, nicht mehr nötig sind. Deshalb ist es wichtig, zunächst die künftigen Architekturen vor der Realisierung zu simulieren, um Engstellen, Hardware-Konflikte und dergleichen aufzudecken. Mit einer derartigen Umgebung ist es einer Testsystem-Herstellerfirma dann auch möglich, mit dem Kunden in der Simulation ein modulares Testsystem aufzubauen, das ideal auf seine Bedürfnisse angepaßt ist. Kosten, Flexibilität und Einsatzgebiete sind durch die Simulation abschätzbar.

Im Rahmen dieser Arbeit wurde ein Verfahren entwickelt und implementiert, das Ablaufverifikation bei Testprogrammen mit verschiedenen Genauigkeitsstufen zuläßt. Weiterhin bietet das Verfahren Schnittstellen, die zur Optimierung von Testprogrammabläufen Verwendung finden. Anhand von Beispielen werden Lösungen gezeigt, wie komplexe Testervorgänge zeiteffizient simuliert und unter Einbindung vorhandener Werkzeuge in den bestehenden Arbeitsfluß integriert werden können.

Aufgrund der damit gesammelten Erfahrung war es möglich, eine Simulationsumgebung zu konzipieren, die zur Erprobung neuer Testerarchitekturen herangezogen werden kann. Durch die Entwicklung im virtuellen Raum ist es möglich, sich von alten Konzepten zu lösen. Dies ist heute auch nötig, um die Forderungen zu erfüllen bzw. den Problemen zu begegnen, die laut International Technology Roadmap for Semiconductors (ITRS) für die nächsten Jahre erwartet werden [ITR03]. Das auf den Erfahrungen des virtuellen Tests basierende System kann damit ebenso für die Planung und Simulation der modularen Testsysteme selbst herangezogen werden.

### 1.1.1 Entwicklungsfluß

Der Erfolg des Konzeptes des virtuellen Tests soll anhand des Entwicklungsflusses und der sich dabei ergebenden kritischen Zeitpfade erläutert werden. Diese Punkte gelten in ähnlicher Form auch für die Entwicklung einer neuen Testerarchitektur.

Aus der Spezifikation eines Mixed-Signal-ICs wird die Spezifikation der Testumgebung entwickelt. In ihr sind funktionale (Beschreibung der Schaltung) und nicht funktionale Angaben (Fertigungstechnologie, maximale Fläche, ...) enthalten [HEI99]. Die Spezifikation, die meist in Form von Texten, Tabellen oder Diagrammen vorliegt, bildet das zentrale Element des Entwicklungsflusses (Abbildung 1.1).

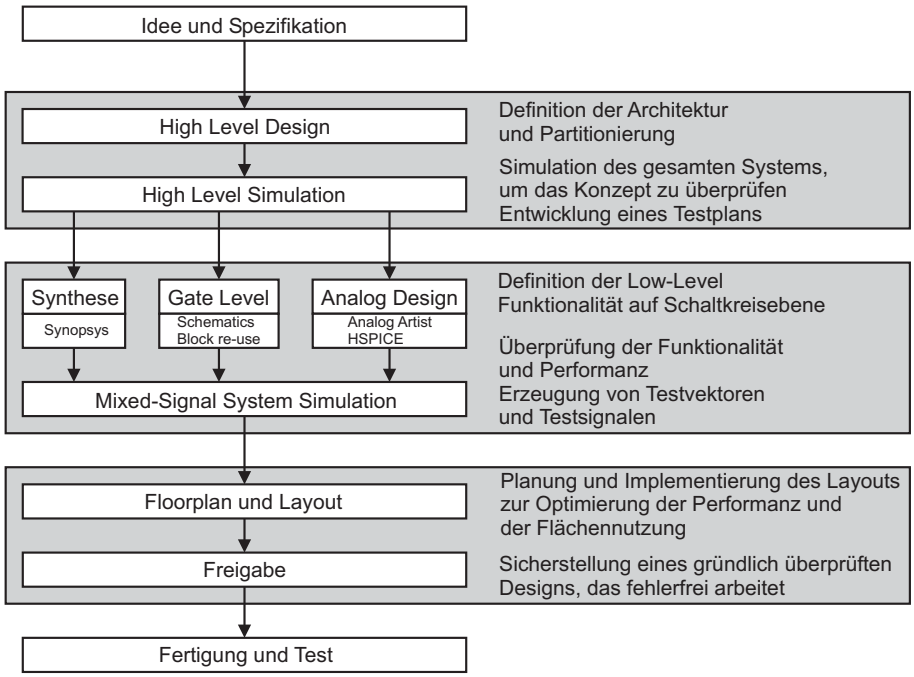



**Abbildung 1.1: Mixed-Signal Design Fluß [THY99]**

Auf abstrakter Ebene wird die Architektur und die Partitionierung der Schaltung festgelegt. Das Designteam verifiziert bei der Simulation auf diesem Niveau das Konzept des kompletten Systems. Anschließend wird in diesem Entwicklungsfluß ein Testplan entworfen. Die Verifikation des Testplans mit Hilfe von Software-Werkzeugen wird derzeit noch nicht eingesetzt, da diese Werkzeuge noch nicht homogen in den Entwicklungsfluß eingebunden sind.

Bei der Definition der Funktionalität auf Schaltkreisebene stehen dem Designteam je nach Anwendungsbereich verschiedene spezialisierte Arbeitsumgebungen zur Verfügung. Die Entwurfs-Verifikation der einzelnen Blöcke erfolgt durch Simulationen. Dazu sind in der Simulationsumgebung Testbenches aufzubauen, welche die Baugruppen mit den gewünschten Signalen stimulieren.

Bei digitalen Schaltungen werden nach der sich anschließenden Synthese der Netzliste Testmuster generiert, wobei mit möglichst wenigen Testmustern möglichst viele Fertigungsfehler entdeckt werden sollen. Es gibt mehrere Vorgehensweisen zur Gewinnung von Testmustern, zwei sollen exemplarisch vorgestellt werden [MOV03]:

- Funktionale Testmuster werden mit Hilfe von Logiksimulationen gewonnen. Hiermit können eher Entwurfsfehler gefunden werden, die unmittelbar zur Fehlfunktion führen. Sie sind für den Fertigungstest, der strukturelle Defekte entdecken muß, nur bedingt geeignet, stehen jedoch aus der Entwurfsarbeit zunächst zur Verfügung.
- Strukturale Testmuster sind speziell für den Zweck erzeugt, Strukturdefekte und daraus resultierendes fehlerhaftes Verhalten der Schaltung zu entdecken. Sie werden heute für digitale Schaltungen mit automatischen Testpatterngeneratoren erzeugt, in besonderen Fällen jedoch auch manuell geschrieben und mit Fehlersimulationen überprüft.

Die Tests für Analogschaltungen werden in der industriellen Praxis heute noch generell von Ingenieuren entworfen und mit Simulationen verifiziert. Im Forschungsbereich gibt es erfolgversprechende Ansätze, die Resultate sind jedoch noch nicht Stand der Technik.

Die so erzeugten Testmuster bzw. Testsignalbeschreibungen müssen in Formate konvertiert werden, mit denen die Zieltestsysteme arbeiten können. Hierbei können Fehler auftreten, die erst während der Testprogrammentwicklung auf dem Testsystem bemerkt werden. Es fehlt auch an dieser Stelle im Entwicklungsfluß an einer Verifikation der Testdaten.

Im letzten Abschnitt des Entwurfsflusses steht das Layout und dessen Verifikation mit Freigabe auf dem Plan. Danach erfolgt die Abgabe des Designs an die Fertigung, das sogenannte Tapeout. Test steht in diesem exemplarischen Entwicklungsfluß am Ende. Erst zu diesem Zeitpunkt wird der Testplan in ein Testprogramm umgesetzt und somit das Testkonzept an der Hardware verifiziert. Schlecht bzw. nicht testbare Komponenten und Schaltungsteile sowie fehlerhafte Einzeltests können daher momentan erst mit Hilfe der ersten Prototypen erkannt werden.

### 1.1.2 Kritische Zeitpfade im Entwicklungsfluß

Mit einer detaillierteren Darstellung dieses Entwurfsablaufs können kritische Zeitpfade, die den Bereich der Testprogrammentwicklung betreffen, lokalisiert werden (Abbildung 1.2).

Nach der Fertigung sind die ersten Prototypen verfügbar. Der Designer führt im Labor den Prototypentest durch. Hierbei ist es entscheidend, Entwurfs- und Herstellungsfehler zu entdecken, zu unterscheiden und zu lokalisieren.

Erst nach Erhalt erster funktionsfähiger Prototypen kann der Testingenieur mit der intensiven Phase der Testprogrammentwicklung und des Debuggings beginnen. Er hat erst jetzt die Möglichkeit das Testkonzept, die Testmuster und letztendlich das Testprogramm auf Richtigkeit zu überprüfen, denn erst jetzt erhält er analysierbare Antworten des realen Bausteins.

Zunächst ist der Charakterisierungstest zu entwickeln. Dabei wird festgestellt, welche Auswirkungen Variationen der Betriebsparameter (Versorgungsspannung, Temperatur, Taktfrequenz, usw.) haben und unter welchen Betriebsbedingungen der Baustein fehlerfrei arbeitet.

Im Produktionstestprogramm erfolgt der Test nicht im kompletten Parameterraum, sondern nur unter ausgewählten kritischen Betriebsbedingungen (Worst-Case). Das Produktionstestprogramm ist auf Kostenminimierung und daher Testzeitminimierung hin optimiert [BEY00].

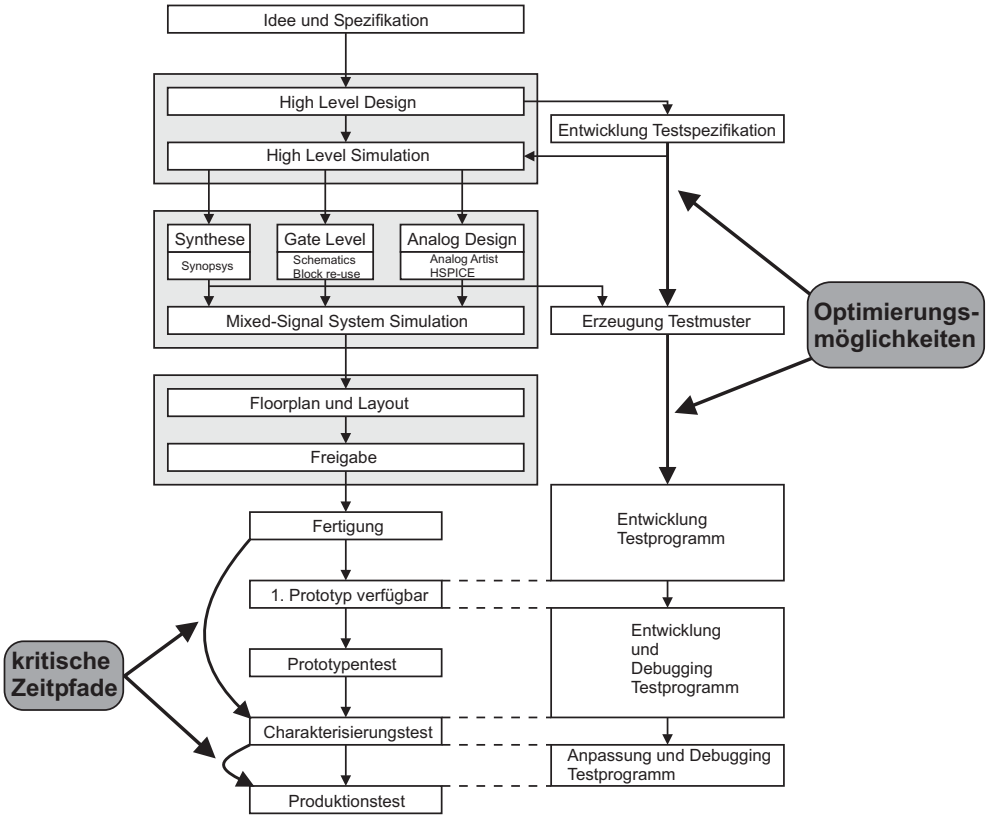



**Abbildung 1.2: Kritische Zeitpfade im Entwicklungsfluß**

Die kritischen Zeitpfade im Testbereich, die für die Markteinführung eines Produktes relevant sind, sind in Abbildung 1.2 angedeutet. Es handelt sich hierbei um den benötigten Zeitraum bis zur Verfügbarkeit der ersten verifizierten Testprogramme. Die Verifikation des Testprogramms kann aber erst mit dem ersten Silizium erfolgen. Es ist ersichtlich, daß an diesem Punkt ein Zeitverlust auftreten kann, der sich durch Wiederholungsschleifen oder durch Iterationen multipliziert, wenn festgestellt wird, daß Fehler im Design oder im Testkonzept vorliegen.

## 1.2 Virtueller Test

Auf der Designseite konnten erhebliche Zeiteinsparungen durch die schrittweise Einführung von Electronic Design Automation (EDA) erreicht werden. Auf der Testseite fehlen jedoch in weiten Bereichen noch automatisierte Verfahren und Softwarewerkzeuge, die den Testingenieur unterstützen. Testspezifikation oder Testplan sind momentan von Hand zu erstellen. Sie existieren, wie die Designspezifikation, in Form von Texten, Tabellen und Diagrammen. Ein einheitliches Format ist nicht vorhanden.

Eine Standardisierung der Testspezifikation mit Hinblick auf systemunabhängige Testbeschreibungssprachen, wie zum Beispiel TSDL (Test Signal Description Language), aus der Testcode generiert werden kann [KRA99] oder STIL (Standard Test Interface Language) [STI03], trägt zu einer Beschleunigung des Entwicklungsflusses auf Testprogrammseite bei. Jedoch stehen simulationsbasierte Verifikationsmethoden für Testspezifikation, testsystemunabhängige Testbeschreibungen und Testcodegenerierung erst am Anfang der Entwicklung.

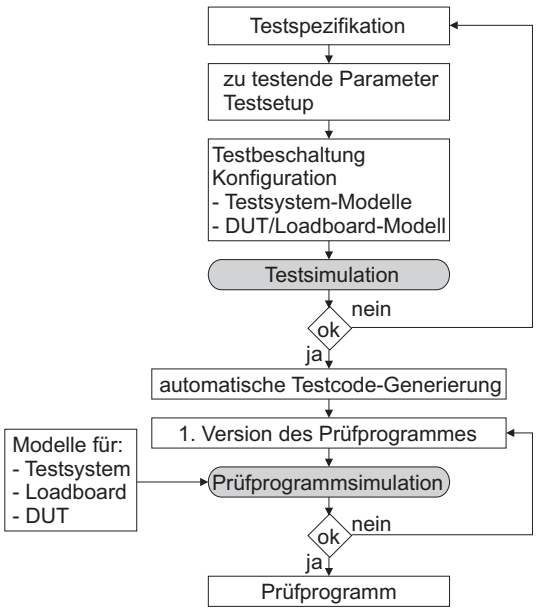



**Abbildung 1.3: Modifizierter Entwicklungsfluß für das Testprogramm**

Der modifizierte Entwicklungsfluß ist in Abbildung 1.3 gezeigt. Testspezifikation und maschinenunabhängige Testbeschreibung sind in einer simulationsbasierten Verifikationsumgebung zu überprüfen. Diese Verifikation nennt man Testsimulation [GOL98-1].

**Definition: „Testsimulation“**
Unter dem Begriff „Testsimulation“ wird die Simulation des Testverfahrens und seiner Parameter

verstanden. Ausgangspunkt für die „Testsimulation“ ist eine Testspezifikation mit zugehörigem Testsetup.

Aus der systemunabhängigen Testbeschreibung kann automatisch Testcode generiert werden. Dieser muß ebenso wie ein von Hand erstelltes Testprogramm verifiziert werden. Auch dies soll aus Gründen der Parallelisierung in einer Simulationsumgebung geschehen. Diesen Schritt nennt man Prüfprogrammsimulation [GOL98-1].

**Definition: „Prüfprogrammsimulation“**
Unter dem Begriff „Prüfprogrammsimulation“ wird die Simulation des maschinenspezifischen Testcodes zur Programmierung der Testschritte verstanden. Ziel der „Prüfprogrammsimulation“ ist der Nachweis des korrekten Testcodes im Zusammenspiel mit den Testerinstrumenten und dem Prüflingsverhalten.

Somit kann die gesamte Entwicklung und Verifikation des Testprogramms zeitlich nach vorne verlagert werden (Abbildung 1.4). Die Verifikation der Testspezifikation, der Testmuster und des Testprogramms während der Designphase bietet die Möglichkeit, Designfehler, mangelnde Testbarkeit, fehlerhafte Testkonzepte und nicht lauffähige Testmuster frühzeitig zu erkennen.

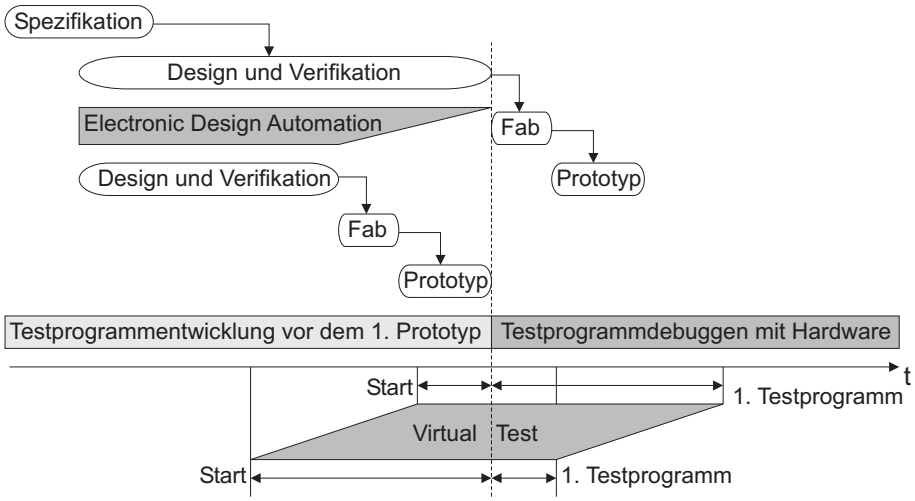



**Abbildung 1.4: Beschleunigung des Entwicklungsflusses durch Virtual Test**

Optimierungsmöglichkeiten gehen über den Begriff „Virtueller Test“ hinaus. Deshalb soll im Folgenden die erste vereinheitlichende Definition [GOL98-1] des Begriffes „Virtueller Test“ aufgrund der obigen Beschreibung für diese Arbeit erweitert werden.

**Definition: „Virtueller Test“**
Unter dem Begriff „Virtueller Test“ werden simulationsbasierte Methoden zur Verifikation des Tests und des für den Test erforderlichen Testcodes inklusive des Zusammenspiels mit Testmustern verstanden.

Verfahren zur Generierung von Testmustern bzw. Signalen (digital wie analog) werden damit ebensowenig berücksichtigt wie Testbeschreibungssprachen zur teilautomatischen Generierung des Testcodes.

## 1.3 Ergebnisse und Gliederung der Arbeit

### 1.3.1 Ergebnisse

Die untersuchte Vorgehensweise bietet die Möglichkeit der „Prüfprogrammsimulation“ und der Optimierung. Dazu wurde ein Verfahren zum netzwerkweiten Datenaustausch entwickelt, der nebenläufige Prozesse zeitgenau synchronisiert. Am Beispiel eines digitalen Pins werden Lösungswege gezeigt, wie in Simulationsmodellen mit großen Datenmengen, die beispielsweise durch Testmuster hervorgerufen werden, zu verfahren ist. Des weiteren werden Abstrahierungen aufgezeigt, um die Komplexität heutiger Instrumente modellierbar zu machen.

Eine eingehende Analyse momentan für Testsysteme verfügbarer Hard- und Software-Komponenten zeigte eine enorme kombinatorische Komplexität, die durch extrem viele Freiheitsgrade in der Verschaltung bestimmt wird. Die Ergebnisse dieser Untersuchung zeigen, daß eine Entfeinerung der Instrumente notwendig wird, um einfach modellier- und programmierbare Instrumente zu erhalten.

Des weiteren können mit den in der Arbeit vorgestellten Konzepten neue Testerarchitekturen entwickelt werden. Diese umfassen Hard- und Software. Damit kann, in Anlehnung an die Prüfprogrammsimulation, das Zusammenspiel verschiedener Komponenten innerhalb einer Architektur verifiziert und verschiedene Architekturkonzepte miteinander verglichen werden. Dies wird in Zukunft notwendig, um die aufgestellten Forderungen an die Testsystem-Ressourcen zu erfüllen.

### 1.3.2 Gliederung

Kapitel 2 befaßt sich mit den Grundlagen der Testsysteme im Hinblick auf Hardware- und Softwarearchitektur. Dabei werden Entwicklungen von Systemen und Instrumenten untersucht und die dabei auftretenden Vor- und Nachteile beleuchtet.

Kapitel 3 beschreibt und bewertet am Markt vorhandene Produkte und existierende wissenschaftliche Arbeiten zum „Virtuellen Test“

Darauf aufbauend wird in Kapitel 4 ein Anforderungskatalog für diese Arbeit aufgestellt. Es beschreibt vorhandene Werkzeuge, fehlende Komponenten und zeigt Lösungsmöglichkeiten und Implementierungen auf, die im Anhang vertieft werden.

Im Kapitel 5 wird ein Konzept vorgestellt, mit dem künftig neue Testsystemarchitekturen in einer Simulationsumgebung entwickelt werden können.

Kapitel 6 vergleicht Simulationsergebnisse mit der Realität und berichtet über Erfahrungen der Industriepartner im Rahmen von Evaluierungen.

Zusammenfassung und Ausblick befinden sich in Kapitel 7.

# Kapitel 2

# Grundlagen zum virtuellen Test

Um eine effektive und effiziente Simulation eines Prüfprogrammes zu gewährleisten, ist eine genaue Analyse der vorhandenen zu modellierenden Testsysteme und der künftig im virtuellen Raum zu entwickelnden Testsysteme notwendig. Es werden Testsysteme verschiedener Hersteller untersucht, um den Aufbau heutiger Mixed-Signal-Tester in allgemein gültiger Form darzustellen. Grundlegende Unterschiede werden herausgestellt, da sie bei der Modellierbarkeit eines Testsystems wichtige Kriterien darstellen können, was wiederum für die Entwicklung der Architektur neuer Testsysteme eine nicht zu vernachlässigende Rolle spielt. Das Ergebnis der Analyse liefert eine Basis für die Konzeption der Testsystem-Modelle.

## 2.1 Analyse der Hardware eines Mixed-Signal-Testers

In diesem Abschnitt wird der Aufbau heutiger Mixed-Signal-Testsysteme untersucht. Die Daten stammen aus den Spezifikationen der Testsysteme M3650 der Firma SZ, VistaVision und Quartet der Firma Credence und einem Eigenbau Tester eines Halbleiterherstellers. Es wird versucht, die Analyse jedoch so allgemein zu halten, daß wesentliche Erkenntnisse auch auf Systeme anderer Hersteller übertragbar sind. Das Eingehen auf instrumentenspezifische Lösungen des einen oder anderen Herstellers liefert keine zusätzlichen Gesichtspunkte in Bezug auf die Modellierung.

### 2.1.1 Strukturelle Gliederung eines Testsystems

Eine Testmaschine ist in drei Hauptkomponenten gegliedert (Abbildung 2.1): Die festinstallierte Haupteinheit, der bewegliche und schwenkbare Testkopf mit Device Interface Board (DIB) und die Programmierstation.

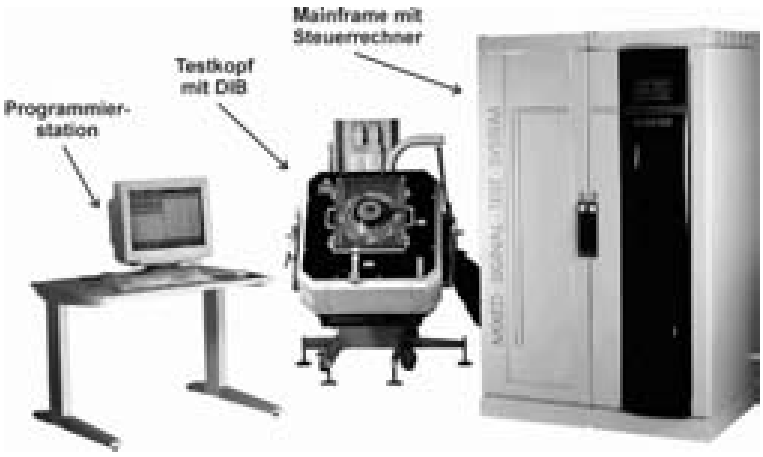



**Abbildung 2.1: Grundkomponenten eines Testsystems**

In der Haupteinheit, einem festinstalliertem Schrank, befinden sich alle zentralen Komponenten des Testsystems wie Strom- und Spannungsquellen, Steuerkarten, Busse, Mastertakt und Steuerrechner.

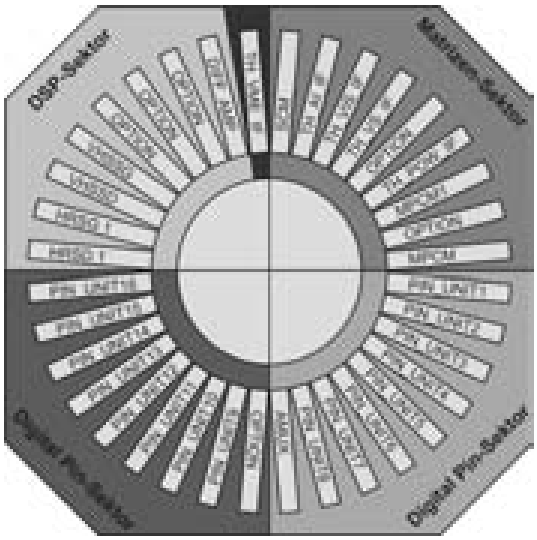



**Abbildung 2.2: Beispiel einer Sektor-Aufteilung im Testkopf**

Im Testkopf befinden sich alle Schaltungen, die zur Signalgenerierung und -erfassung an den Anschlüssen des Prüflings notwendig sind. Dadurch werden kurze Verbindungswege vom Prüfling zu den Pinelektroniken möglich. Die Aufteilung der Einsteckkarten unterliegt Kriterien der Kühlung, kurzer Verbindungswege zum DUT und hoher Packungsdichte [CHA98]. Die Berücksichtigung dieser Kriterien führt häufig zu einer radialen Anordnung. Abbildung 2.2 zeigt die prinzipielle Verteilung der einzelnen Meßkarten auf Sektoren im Testkopf. Die beiden „Digital Pin-Sektoren“ enthalten digitale Pinelektroniken, die zur Erzeugung und Erfassung digitaler Signale

und Pattern benötigt werden. Der „Matrizen-Sektor“ enthält verschiedene Relais-Schaltmatrizen, die analoge und digitale Signale sowie Stromversorgungen an den Prüfling anlegen und flexibel von und an die Meßmittel weiterleiten kann. Im „DSP-Sektor“ finden sich Instrumente, die zur Generierung und Erfassung analoger Signale (Waveforms) in der Lage sind. Zudem liegt in diesem Sektor die Interface-Karte, welche die Kommunikation des Steuerrechners mit den Instrumenten ermöglicht. Eine eingehendere Untersuchung dieser räumlichen Anordnungen bringt allerdings keine Erkenntnisse für die Modellierung eines vorhandenen Testsystems in Bezug auf den virtuellen Test.

### 2.1.2 Bussysteme

#### 2.1.2.1 Interne Busstruktur

Die Busse eines Testsystems erfüllen die Aufgaben, Daten zwischen den Geräten untereinander und dem Steuerrechner auszutauschen, Instrumente untereinander zu synchronisieren sowie analoge oder digitale Signale zwischen Baugruppen auszutauschen. Um diese Aufgaben zu erfüllen, unterscheidet man drei verschiedene Bustypen, die noch hinsichtlich ihrer Anordnung im Testkopf bzw. im Schrank (Hauptgestell) untergliedert werden können: [SZ00]

- Datenbus
  Der Datenbus verbindet die einzelnen Instrumente und den Steuerrechner im Hauptgestell miteinander, damit sie Befehlssequenzen sowie Meßergebnisse miteinander austauschen können. Standardisierte Busse (RS232, GPIB, VME) werden integriert und bieten Schnittstellen nach außen, um Meßgeräte dritter Anbieter (z.B. für Hochfrequenz-Messungen) einbinden und in der gleichen Umgebung programmieren zu können.
  Dabei verbindet der VME Bus (Versa Module Europe Bus) im Main Frame und im Testhead den Steuerrechner mit den zentralen Elektronikeinheiten und Unterstützungsmodulen. Er ist ein standardisierter Bus nach IEEE 1014 mit 16 Adress- und 16 Datenbits.

- Digitaler Signalbus (Ereignisbus)
  Triggersignale zur Synchronisation zwischen den Instrumenten bzw. zwischen Prüfling und Instrumenten werden über das digitale Bussystem geleitet. Derartige Busse finden sich sowohl im Testkopf als auch im Main Frame. Beide Busse werden über das „Digital I/O Interface“ miteinander verbunden. Für diese Konfiguration gibt es keine standardisierte Lösung.

- Analoger Signalbus
  Um analoge Signale von den Anschlußpins zu weiteren Meßgeräten zu schalten, ist ein

analoger Signalbus notwendig. Dieser Bus ist von Hersteller zu Hersteller unterschiedlich ausgelegt und ist nur im Testkopf vorhanden. Eine Weiterleitung der analogen Signale zum Mainframe würde diese Signale durch zu große Störungen auf Grund des Leitungsweges verfälschen.

Aufgrund der häufig verwendeten VME-Bus-Technologien und räumlichen Anordnungen ergeben sich eine Reihe von Vorgaben, Einschränkungen und Problemen. Der VME-Bus ist, wie beschrieben, ein E/A-Bus für Backplanes. Seine Reichweite liegt bei 50 cm. Die räumliche Trennung von Mainframe und Testkopf benötigt aber Leitungslängen von mehreren Metern. Deshalb ist eine Umsetzung der Daten vom VME-Bus des Mainframes auf den VME-Bus des Testheads mit Hilfe von zwei Interface-Karten sowie eines speziellen Busses für größere Leitungslängen (zum Teil Eigenentwicklungen der Testsystemhersteller) notwendig (Abbildung 2.3).

In Abbildung 2.3 erkennt man eine Vielzahl von zusätzlichen Verbindungen zwischen den Modulen im Mainframe und denen im Testkopf. Die Notwendigkeit dieser Verbindungen wird im Abschnitt 2.1.4 (DSP und DPIN) erläutert. Es ist allerdings zu überlegen, ob diese Verbindungen durch eine geeignetere Architektur verringert werden können. Eine derartige Reduktion würde erheblich Kosten im Leitungsbereich einsparen sowie zu einer massiven Gewichtsverminderung führen. Das Handling des Testheads würde durch derartige Einsparungen erheblich vereinfacht werden.

Diese zusätzlichen Verbindungen entstanden jedoch durch die Forderung nach immer neueren und flexibleren Instrumenten, die mit einer Testsystemarchitektur verbunden werden sollten, die zum Teil 20 Jahre alt ist. Durch dieses evolutionäre Vorgehen wurden modulare und hierarchische Strukturen der Architektur aufgeweicht.

#### 2.1.2.2 Triggering und Clocking für Mixed-Signals Tests

Die Synchronisation des digitalen Busses im Testkopf und dem Mainframe erfolgt über ein „Digital I/O Interface“. Diese Schnittstelle setzt die Signale der 32 Leitungen im Mainframe auf die 24 Synchronisationsleitungen (Test Head Digital Bus) im Testkopf um. Dabei handelt es sich nur um eine Punkt zu Punkt Verbindung über 6 m Kabel ohne jegliche Synchronisationsmechanismen (Abbildung 2.4).

Doch warum wird ein derartiger Aufwand für die Synchronisationsleitungen betrieben? Ein kurzer Ausflug in die Entwicklung der Synchronisationstechniken älterer Architekturen beschreibt die Problematik beim Mixed-Signal Test.

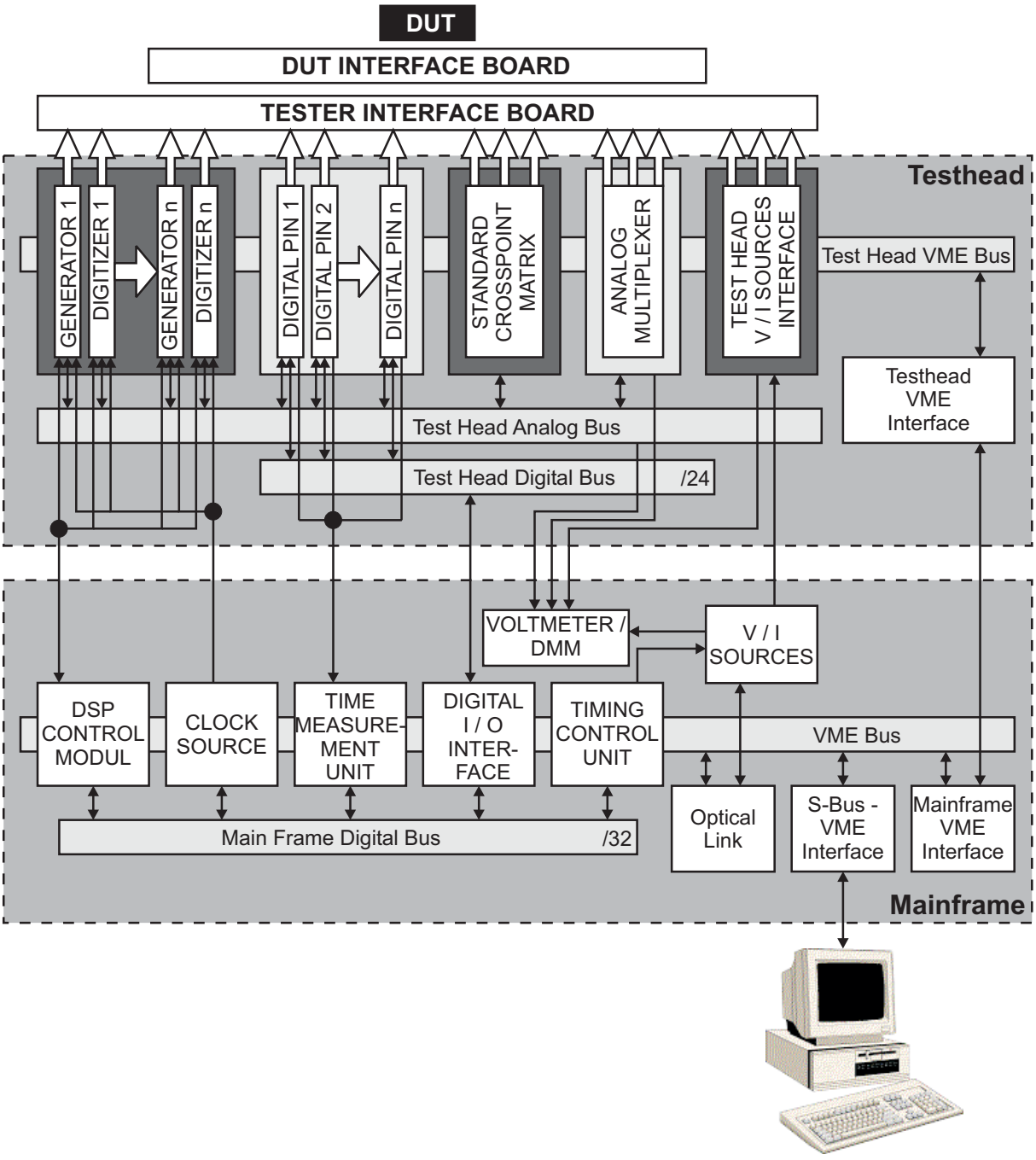



**Abbildung 2.3: Anbindung der Instrumente an die Busstruktur**

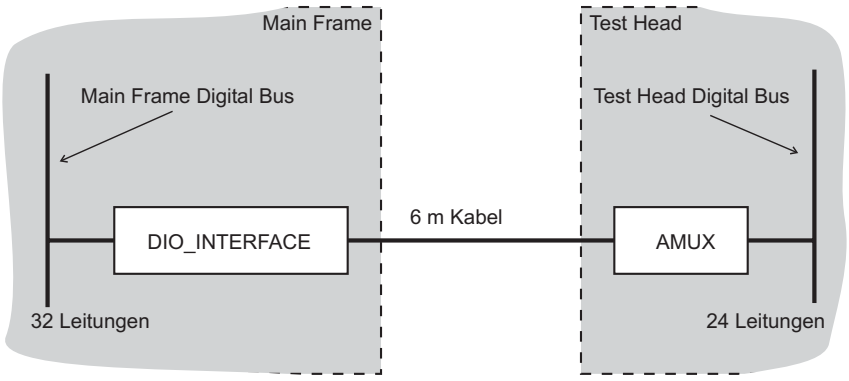



**Abbildung 2.4: Verbindungsproblematik Mainframe Testkopf am Beispiel Synchronisationsleitungen**

Bei digitalen Tests müssen nur digitale Daten verarbeitet werden. Diese werden aus dem Speicher über den digitalen Sequencer als Stimulus an das DUT geliefert. Die digitalen Bausteinantworten werden erfaßt und mit den Erwartungswerten, die im Speicher abgelegt sind, verglichen. Beim Mixed-Signal-Test müssen zusätzlich analoge „Daten“ behandelt werden. Diese werden entweder durch analoge Signale oder digitale Daten repräsentiert. Je nach Anwendung kommen beide Darstellungsformen zum Einsatz (z.B. ADC- oder DAC-Test). Für diese Anwendungsfälle ist eine genaue Synchronisation zwischen den analogen und digitalen Komponenten eines Testsystems extrem wichtig. In früheren Ausführungen hatten beide Komponenten ihren eigenen Takt. Die Synchronisation wurde nur durch Triggerleitungen hergestellt (Abbildung 2.5). Dies führt zu großen Ungenauigkeiten, da eines der beiden Systeme immer auf das andere warten mußte. Bei heutigen digitalen Subsystemen wird eine sehr genaue zeitliche Auflösung erreicht. Weiterhin ist es möglich die Patternfrequenz während des Testlaufs (on-the-fly) zu verändern. Diese Vorzüge bergen allerdings den Nachteil von erhöhtem Jitter und Nichtlinearität in sich, was aber gerade den Anforderungen bei analogen Bauelementen widerspricht.

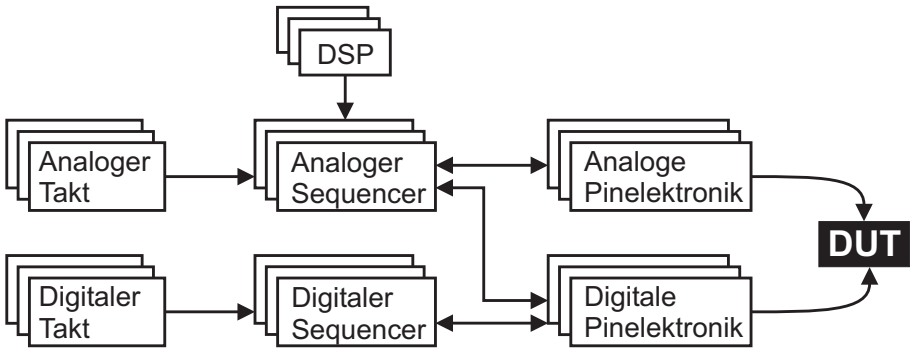



**Abbildung 2.5: Ältere Mixed-Signal-Testerarchitektur**

Somit ist es notwendig, die analogen Sequencer und die analoge Pinelektronik mit unabhängigen

Takt-Signalen zu versorgen, die in der Frequenz variabel sind und geringen Jitter zeigen. Diese aufwendige Taktgenerierung ergibt größtmögliche Flexibilität. Um die Wiederholpräzision von Meßergebnissen zu gewährleisten, müssen diese Taktgeneratoren phasenstarr an einen Referenztakt gekoppelt sein. Diesen Referenztakt liefert der digitale Master Clock (Abbildung 2.6). Mit dieser Implementierung ist es möglich, verschiedene analoge Takte durch digitale Events, die durch digitale Pattern hervorgerufen werden, zu synchronisieren. [ZAM98]

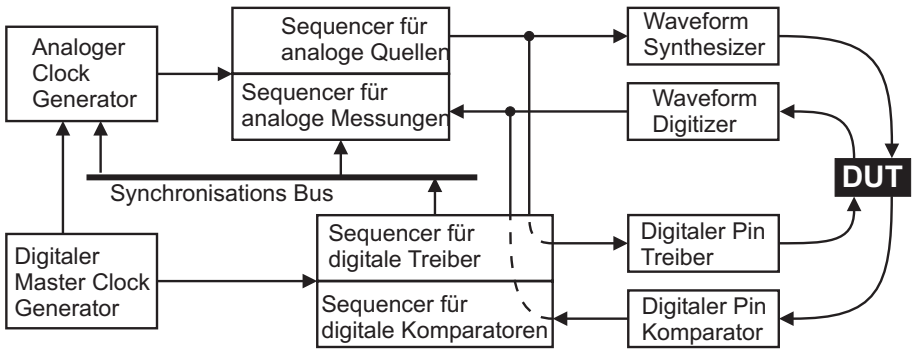



**Abbildung 2.6: Verbesserte Architektur**

### 2.1.3 Spannungsversorgung

Eine zentrale Spannungsversorgung (Abbildung 2.7) stellt eine weitere Schwierigkeit dar. Die Versorgung muß über eine mehrere Meter lange Leitung an die Instrumente im Testkopf gebracht werden. Über die gleiche Länge muß eine Senseleitung die korrekte Spannungsversorgung im Testkopf überprüfen. Je nachdem, an welcher Stelle im Testkopf die Senseleitung die Spannung abprüft, ist es möglich, daß andere Einsteckkarten mit zu großer bzw. zu geringer Spannung versorgt werden. Daher ist zu erwägen, die zentrale Versorgung durch eine dezentrale Versorgung zu ersetzen. Dabei liefert eine zentrale Einheit nur einen Basiswert, aus dem jedes Instrument selbständig die von ihm benötigte Spannung generieren kann.

### 2.1.4 Instrumente

Hier sollen am Beispiel von zwei Instrumenten (DSP und DPIN) die Unterschiede in den Generationen der Instrumente erläutert werden. Vor- und Nachteile alter und neuer Architekturen werden angedeutet.

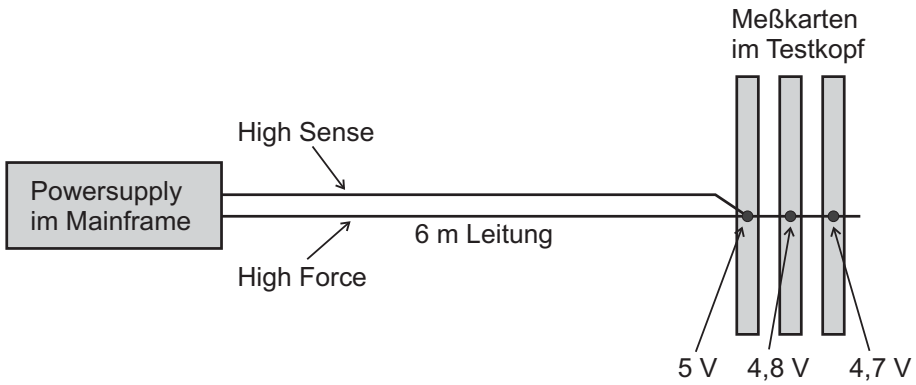



**Abbildung 2.7: Problem der zentralen Spannungsversorgung**

#### 2.1.4.1 DSP

Am Beispiel der DSP-Module kann die alte Hardware-Generation (DSP_CD, Abbildung 2.8) sehr gut mit der neuen (DSP_WAVE_CD, Abbildung 2.9) verglichen werden.

Der DSP_CD hat auf der Einsteckkarte exakt eine „Sende-“ und eine „Empfangseinheit“. Die Karte besitzt eine DSP-Einheit und einen Speicherblock. Über den VME-Bus erhält der DSP_CD die Programmierung sowie Daten über die zu generierenden Signalformen vom Steuerrechner. Die Daten über die Waveform werden aus dem Speicher ausgelesen und an den Generator geliefert. Hierzu ist eine direkte Verbindung zwischen DSP-Einheit und Generator implementiert. Der Digitizer tastet Signalformen ab, welche das DUT liefert, und schickt sie über eine zweite direkte Verbindung zur DSP-Karte im Mainframe zurück. Diese empfangenen Daten werden im Speicher abgelegt und in der DSP-Einheit verarbeitet. In der alten Architektur war es also möglich, mit einem DSP_CD einen Generator und einen Digitizer anzusteuern und parallel Signale zu generieren, abzutasten und zu prozessieren.

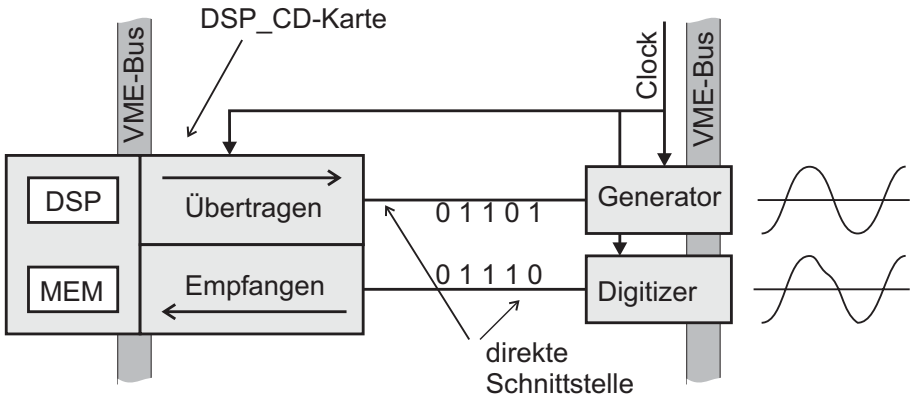



**Abbildung 2.8: DSP-Konfiguration**

Mit Hilfe des DSP_WAVE_CD (Abbildung 2.9) sollte ein Vielfaches an Flexibilität beim Gene-

rieren und Erfassen von Signalen erreicht werden. Auf einer DSP_WAVE_CD-Karte waren nicht mehr länger nur ein Übertragungs- und ein Empfangskanal, sondern zwei bidirektionale Kanäle. Diese Struktur sollte die Flexibilität bei der Konfiguration von Testsystemen erhöhen. Je nach Bedarf sollten an eine DSP_WAVE_CD-Karte zwei Digitizer oder zwei Signalgeneratoren oder jeweils ein Generator und ein Digitizer angeschlossen werden, die durch die bidirektionalen Kanäle unterstützt werden. Doch in der Praxis stellten sich einige Probleme ein, da auf der DSP_WAVE_CD-Karte ebenso wie auf dem Vorgängermodell nur ein DSP und ein Speicherbereich vorhanden ist. Man kann parallel Signale generieren sowie aufzeichnen. Allerdings müssen die Abtastwerte der analogen Signalformen sequentiell prozessiert werden, da nur ein DSP vorhanden ist. Der Betrieb der DSP-Karten im mixed-mode (wie es bei der alten Karte möglich war - ein Kanal sendet, während der andere empfängt) ist mit dieser Karte nicht möglich. Es müssen deshalb immer zwei DSP_WAVE_CD-Karten in das Testsystem eingebaut werden. Eine versorgt die Generatoren, während die andere an die Digitizer angebunden ist. Damit ist die erhoffte Flexibilität nicht erreicht worden.

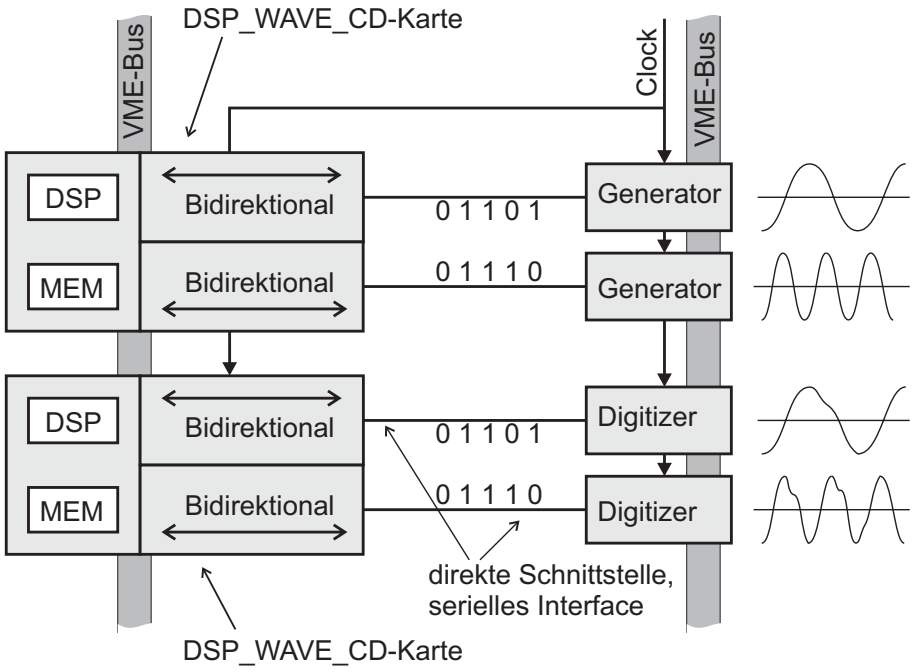



**Abbildung 2.9: Neue DSP-Konfiguration**

#### 2.1.4.2 Pinelektronik (MPIN - DPIN)

Im Bereich der Pinelektronik können unterschiedliche Architekturen am Beispiel des MPIN (Mixed-Signal-Pin) und des DPIN (Digital-Pin) untersucht werden. Die Namensgebung der Pins läßt unterschiedliche Einsatzbereiche vermuten. Es sind zwar Unterschiede in Spannungs-

hüben sowie in Frequenzbereichen vorhanden, doch vom Prinzip her sind beide Pins für ähnliche Applikationen einsetzbar.

Bevor auf die spezifischen Eigenheiten der beiden Pinelektroniken eingegangen wird, soll ein kurzer allgemeiner Überblick über den prinzipiellen Aufbau gegeben werden. Die digitale Pinelektronik hat in erster Linie die Aufgabe, Logiksignale zu generieren und zu erfassen. Unter Umständen stehen ihr noch Baugruppen zur Vermessung der Ausgangstreiber eines digitalen Pins zur Verfügung. Abbildung 2.10 zeigt den prinzipiellen Aufbau, der folgende Komponenten enthält:

- Signaltreiber
  Die Treibereinheit versorgt das zu testende Bauelement mit digitalen Signalfolgen. Diese setzen sich aus dem Pattern und dem Timeset zusammen. Das Pattern (siehe Abschnitt 2.2.4.2) beinhaltet für jeden Pin die Logik-Werte. Eine Zeile nennt man Vektor. Zu welchem exakten Zeitpunkt der jeweilige Wert eines Vektors am Pin anliegt, wird durch das Timeset bestimmt.
  Das Pattern wird mit dem Timeset im Timing Edge Generator kombiniert. Den logischen Werten werden dann die elektrischen Treiberpegel zugewiesen, die dann am DUT-Pin anliegen.

- Signalempfänger
  Die vom DUT gesendeten digitalen Signale werden in den beiden Komparatoren auf High- und Low-Pegel verglichen. Der daraus resultierende Logikwert wird zu dem im Timing Edge Generator berechneten Zeitpunkt mit dem Vergleichswert aus dem Pattern verglichen.

- Aktive Lasten
  Die zur Belastung des Prüflings während des Tests vorgesehene programmierbare Last muß für High- und Low-Ausgangspegel getrennte Einstellungen des Laststroms erlauben.

- PMU
  Zur Messung der U/I-Charakteristik der Prüflingsanschlüsse stehen hochgenaue Spannungs- und Strommeßgeräte zur Verfügung. Die Meßeinheiten, die eine verhältnismäßig hohe Eingangskapazität ausweisen, müssen vom Prüfling getrennt werden, um kapazitive Lasten zu minimieren.

In Abbildung 2.10 sind zwei pinspezifische Unterschiede markiert. Auf die sowie auf weitere Eigenheiten soll im Folgenden näher eingegangen werden.

Der MPIN baut auf einer sogenannten „Shared-Ressource-Architecture“ auf. Hierbei teilen sich mehrere Kanäle die gleichen Ressourcen (Abbildung 2.11).

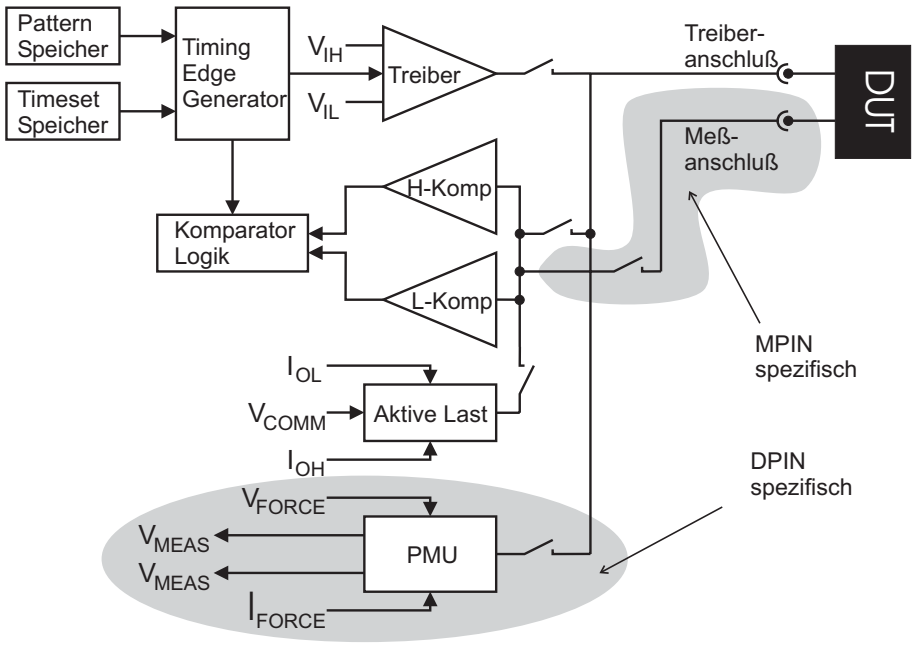



**Abbildung 2.10: Prinzipieller Aufbau einer Pinelektronik (MPIN, DPIN)**

Vier MPIN-Kanäle finden Platz auf einer MSPU (Mixed-Signal-Pin-Unit). Für diese vier Pins stehen nur ein einziges Mal die Spannungen für die Treiber- und Compare-Pegel (Voltage-Set) zur Verfügung. Eine MSPU ist eine Einsteckkarte im Testkopf. 16 dieser Karten finden im Testkopf Platz, damit kann eine Testmaschine dieser Bauart mit maximal 64 Mixed-Signal-Pins bestückt werden. Diese 64 MPIN-Kanäle greifen wiederum auf gemeinsame Ressourcen zu, nämlich den DSS_MEM und den DSS_SEQ. In diesen beiden Komponenten, die sich im Mainframe des Testsystems befinden, werden die Patterndaten und Zeitsteuerungen (Timeplates) für alle MPINs zentral gespeichert. Die Patterndaten werden dann über direkte Verbindungen vom DSS_MEM an die MSPUs geliefert. Der Sequencer verwaltet die Timeplates und steuert den Timegenerator. Dieser kontrolliert seinerseits über direkte Leitungen die exakte zeitliche Abarbeitung der Patterndaten in den MPIN-Kanälen. (Eine detaillierte Erklärung über die Abarbeitung von Pattern und dem Zusammenspiel zwischen Timeplates und Patterdaten findet sich im Abschnitt 2.2.4.2.) Der MPIN verfügt über zwei getrennte Anschlüsse zum Treiben und Erfassen digitaler Signale. Dadurch ist es möglich, den MPIN in folgenden Modi zu betreiben:

- Bidirectional mode
  Treiber und Komparator sind über den gleichen Anschluß mit einem bidirektionalem Pin am DUT verbunden.

- Drive only mode
  Nur der Treiber ist mit einem Eingangspin des DUT verbunden. Der Komparator wird

durch Relais abgetrennt.

- Sense over sense path
  Der Komparator (inklusive Laststrombrücke) des MPINs ist mit einem Ausgangspin des DUT über den Meßanschluß verbunden. Der Treiber wird durch ein Relais abgeschaltet. Dieser Modus wird vor allem dann verwendet, wenn die Kapazität des Treibers oder der Strom, der im Tristate-Zustand fließt, die Messung zu stark beeinflußt.

- Sense over drive path
  Der Komparator (inklusive Laststrombrücke) des MPINs ist mit einem Pin des DUT über den Treiberanschluß verbunden. Der Treiber wird durch ein Relais abgeschaltet. Dieser Modus wird dann verwendet, wenn die Laststrombrücke zeitweise mit einem Pin verbunden wird, der nur mit einem Treiber verbunden ist.

- Splitted mode
  Der Treiber ist mit einem Eingangspin des DUT verbunden, während der Komparator mit einem Ausgangspin des DUT verbunden ist. Auf diese Weise können zwei verschiedene Pins des DUT mit dem MPIN verbunden werden. Dieser Modus findet vor allem dann Anwendung, wenn die Anzahl der DUT-Pins größer ist als die Anzahl der MPINs im ATE.

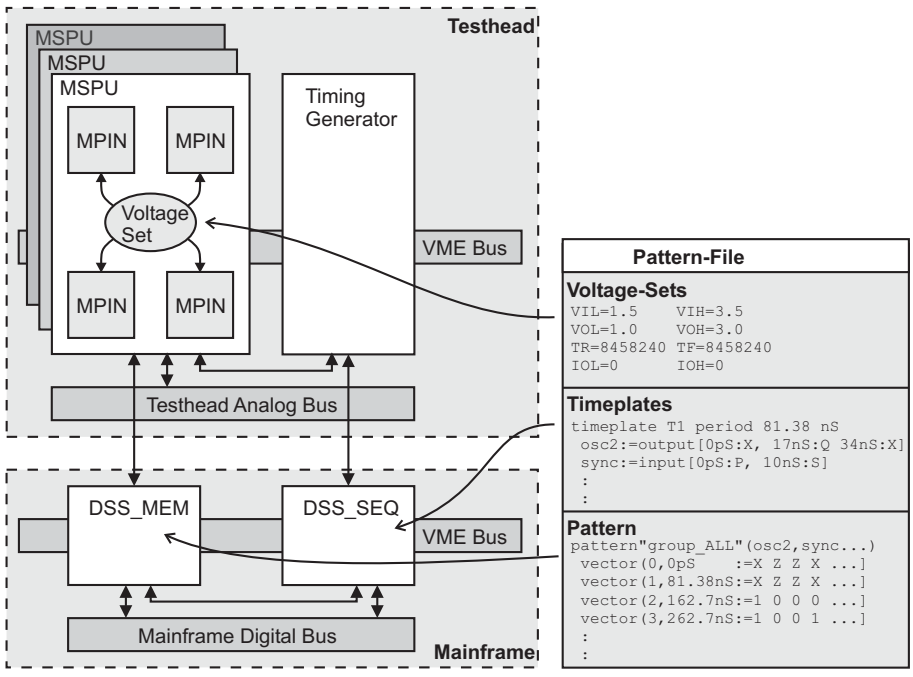



**Abbildung 2.11: Shared-Ressource-Architecture beim MPIN**

Im Gegensatz dazu stellt der zeitlich später entwickelte DPIN eine echte „Per-Pin-Architecture“ dar. Das heißt, sämtliche Ressourcen sind jeweils für jeden Pin real in der Hardware verfügbar.

Auf einer DPIN-Einsteckkarte befinden sich acht DPIN-Kanäle. Damit ist bei gleicher Testkopfausführung das System mit 128 digitalen Kanälen ausgerüstet. Auf einer DPIN-Karte befinden sich alle Ressourcen, die jeder Kanal benötigt. Zum Beispiel Patternspeicher, der beim MPIN zentral für alle Komponenten im Mainframe untergebracht war oder die PMU, die beim MPIN nicht vorhanden ist. Die Verfügbarkeit sämtlicher Ressourcen für jeden Kanal hebt bisherige Einschränkungen auf. Um die damit gewonnene Flexibilität noch weiter zu treiben, wurden zusätzliche Steuerungsmöglichkeiten implementiert, wie z.B. das patternabhängige Zuschalten der „Aktiven Last“. Genau diese zusätzlichen Freiheitsgrade und deren Kombinatorik erschweren die Modellierung erheblich.

Abschließend sollen die wichtigsten Vor- bzw. Nachteile beider Pinelektroniken, die sich aufgrund der unterschiedlichen Architektur ergeben, gegenübergestellt werden. Die Flexibilität der Per-Pin-Architecture des DPIN wird mit einer Reihe von Nachteilen erkauft:

- Im Vergleich zu anderen Instrumenten benötigt der DPIN mehr unterschiedliche Spannungsversorgungen
- 2 W Leistungsaufnahme pro Kanal (16 W pro Einsteckkarte)
- Platine mit 14 Lagen
- Aufgrund der standardisierten Anschlüsse zum Loadboard steht dem DPIN kein separater Treiber- und Meßanschluß zur Verfügung
- Die Komplexität und Flexibilität des DPIN kann praktisch nicht mehr in der Software nachgebildet werden (Bis jetzt wurden 200 Bugs in der Software behoben)

Ein Vorteil für den DPIN ist, daß weniger Leitungen vom Mainframe zum Testhead gezogen werden, da er seine Daten und Programmierung komplett vom VME-Bus bezieht.
Demgegenüber steht der Vorteil des MPIN, der Patterndaten erheblich schneller laden kann, da die Daten nicht per Pin aufgeschlüsselt und gesendet werden müssen.

Für die Modellierung des DPIN stellt seine enorme Flexibilität eine erhebliche Schwierigkeit dar. Um den DPIN zu programmieren stehen dem Testingenieur 179 Befehle zur Verfügung. Alleine der Befehl, um Steuerleitungen an den Pintreiber anzulegen, hat 30 Optionen. 10 Befehle mit ähnlicher Optionenvielfalt gibt es zur Steuerung des Treibers, Comparators und der aktiven Last. Daraus entsteht eine extreme Vielfalt an möglichen Kombinationen, die eine Modellierung erschweren.

**Schlußfolgerung:**
Aus diesem Vergleich ist, wie auch schon beim DSP, ersichtlich, daß eine neue, flexiblere Architektur nicht nur Vorteile bringt. Von daher drängt sich immer mehr die Frage auf, ob derartige

Nachteile nicht schon im Vorfeld mit Hilfe einer Simulation umschifft werden können. Dies würde eine ausreichende Flexibilität der Instrumente bei gesteigerter Software-Qualität und schnellen Softwaremodellen bedeuten.

### 2.1.5 Unterschiede in der Steuerungs-Architektur

Die bisher vorgestellten Komponenten finden sich in dieser oder ähnlicher Form in allen Testsystemen wieder. Hinzukommen spezielle Meßgeräte wie zum Beispiel Iddq.

Ein wesentlicher Unterschied ergibt sich zwischen Testsystemen mit zentraler und dezentraler Steuerung.

In einem Testsystem mit zentraler Steuerung (Abbildung 2.12) liefert der Steuerrechner sequentiell Testprogrammblöcke an die einzelnen Instrumente. Diese arbeiten den empfangenen Datenblock ab und senden nach erfolgter Ausführung ein Acknowledge Signal an den Steuerrechner. Daraufhin sendet dieser den nächsten Datenblock. Externe Geräte wie Handler, Prober oder zusätzliche Meßgeräte werden direkt an die Standardschittstellen (RS232, GPIB) des Steuerrechners angeschlossen.

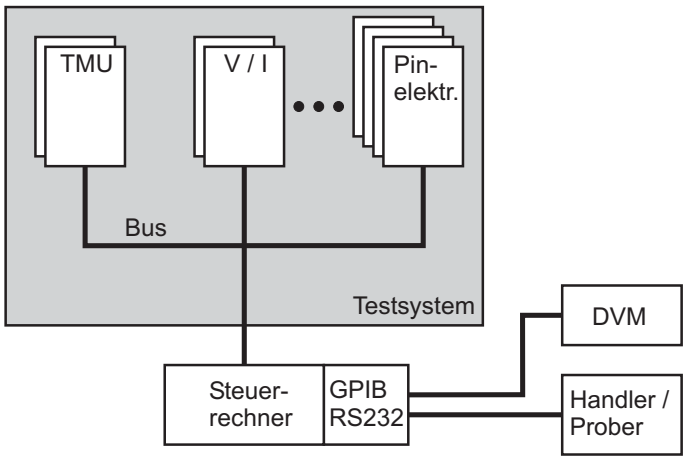



**Abbildung 2.12: Testsystem mit zentraler Steuerung**

Der Vorteil der zentralen Steuerung liegt in der Debugfähigkeit. Das Testprogramm kann vom Steuerrechner aus zu jedem Zeitpunkt angehalten werden. Auf diese Weise kann der Benutzer eingreifen und interaktiv Daten mit dem Testsystem kommunizieren, um so Instrumenteneinstellungen zu ändern.

Die Nachteile liegen in der Geschwindigkeit. Jeder Befehlsblock wird über den Bus geschickt bevor er ausgeführt werden kann. Somit bildet die Kommunikation zwischen Steuerrechner und

Testsystem einen limitierenden Faktor, da sämtliche Daten erst zur Programmlaufzeit ausgetauscht werden. Die Leistungsfähigkeit des Steuerrechners beeinflußt ebenso die Performance des Systems. Bei der Online-Auswertung muß der Rechner Meßergebnisse analysieren, bevor der nächste Programmblock abgesetzt werden kann.

Bei Testsystemen mit dezentraler Steuerung (Abbildung 2.13) sendet die Programmierstation (Hostrechner) das gesamte Testprogramm innerhalb der Setup-Phase an den Sequencer des Testsystems. Dieses verteilt die einzelnen instrumenten-spezifischen Programmblöcke an die DSPs der jeweiligen Instrumente.
Wird das Testprogramm abgearbeitet, so übernimmt nicht mehr die Programmierstation die zentrale Position. Der Sequencer übernimmt die Synchronisation der einzelnen Karten, in dem er die Instrumenten-Karten in der im Testprogramm festgelegten Abfolge antriggert und ihnen mitteilt, welcher Programmblock momentan auszuführen ist. Die Karten führen daraufhin die in ihrem Speicher festgelegte Befehlsfolge des aktuellen Blocks aus.
Der Sequencer übernimmt ebenso die Steuerung externer GPIB-Geräte, Handler und Prober.

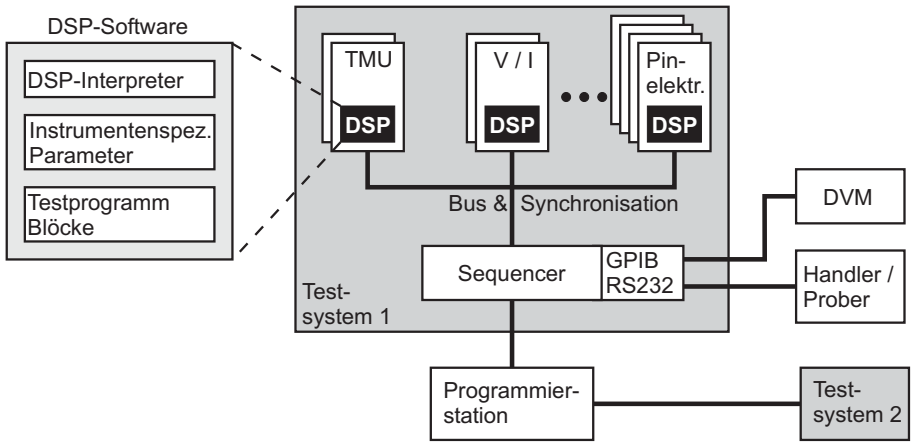



**Abbildung 2.13: Testsystem mit dezentraler Steuerung**

Dieses Vorgehen beschleunigt die Ausführungszeit, da keine Befehlssequenzen über Bussysteme zur Laufzeit des Testprogramms verschickt werden, sondern nur noch Triggersignale. Damit ist der Datenaustausch nicht mehr der limitierende Faktor.
Die gemessenen Daten werden zur Programmierstation während Bearbeitungspausen zurückgesandt. Die Performance der Programmierstation ist somit kein kritischer Faktor bezüglich der Ausführungsgeschwindigkeit. Außerdem ist es möglich, mehrere Testsysteme mit einer Programmierstation zu verbinden.
Nachteilig ist die mangelnde Debugfähigkeit, da das Programm getrennt von der Programmierstation läuft.

In den meisten Testsystemen findet sich ein Gemisch aus zentraler und dezentraler Steuerung.

Voltmeter, Strom- und Spannungsquellen unterliegen dem Prinzip der zentralen Steuerung. Sie erhalten einen Befehlssatz über den Bus und führen diesen sofort aus. Demgegenüber stehen digitale Pinelektroniken (siehe Abschnitt 2.1.4.2) und DSP-Einheiten (siehe Abschnitt 2.1.4.1). Auf diese Karten werden zur Initialisierungsphase Pattern oder DSP-Routinen geladen, die eine Abfolge von verschiedenen Funktionen festlegt. Diese Programm-Module werden zur Laufzeit des Testprogramms aufgerufen und laufen dann selbständig auf den entsprechenden Instrumenten.

### 2.1.6 Hierarchische Aufteilung der Testsystemressourcen

Aus der Untersuchung der Steuerungs-Architektur 2.1.5 und den Betrachtungen einzelner Instrumente 2.1.4 kann eine hierarchische Aufteilung der Testsysteme und Instrumente vorgenommen werden.

Diese Aufteilung erfolgt nach folgendem Gesichtspunkt: Sind Ressourcen oder Controller pro Pin, pro Baugruppe, pro Sequencer oder pro Rechner vorhanden? Das Prinzip der Befehlsübertragung läßt sich auf diese Struktur abbilden. Werden einzelne Befehle, Befehlsblöcke, Unterprogramme, Testschritte oder ganze Testprogramme an die Instrumente des Testsystems gesendet?

Man stellt fest, daß sich in einem Testsystem eine Kombination aus all diesen Hierarchie-Typen findet (Tabelle 2.1). Die sich daraus ergebende Kombinatorik macht die Systeme komplex und damit eine Modellierung extrem schwierig.

| **Typ** | **Instrument** | **Programmcode** |
|---|---|---|
| pro Pin | DPIN | Pattern |
| pro Baugruppe | DSP | Programm-Module |
| pro Sequencer | MPIN | Pattern |
| pro Rechner | Voltmeter | Befehlsblock |

**Tabelle 2.1: Hierarchie der Testsysteme**

Im Anwendungs-Bereich „Virtueller Test“ kann ein Ausweg in der Modellierung solcher Strukturen auf hoher Abstraktions-Ebene gefunden werden. Hier ist es beispielsweise nicht essenziell, daß ein Patternspeicher pro Pin nachgebildet wird. Es kann über ein Patterdatenfeld, auf das jedes Modell einer Pinelektronik zugreifen kann, für den Anwender die gleiche Verhaltensweise modelliert werden. Im Gegensatz dazu ist es bei der Untersuchung neuer Testsystemarchitekturen wichtig, diese Strukturen zu untersuchen. Nur so können Engpässe in den Systemen aufgedeckt und vermieden werden.

### 2.1.7 Trends

Gemäß der ITRS (International Technology Roadmap for Semiconductors) werden die bestehenden Testsystemarchitekturen in den nächsten fünf Jahren nicht mehr mit der komplexer und schneller werdenden Technik der Prüflinge Schritt halten können. Die Erforschung völlig neuer, modular aufgebauter und dezentral organisierter Architekturalternativen stellt deshalb eine der dringendsten Aufgaben für die Wissenschaft und Industrie dar. Bei den jeweiligen Testsystemherstellern entsteht erfahrungsgemäß ca. alle 10 bis 20 Jahre eine grundlegend neue Testsystemgeneration. Die Marktanalyse zeigt deutlich, daß die Architekturen aller aktuell auf dem Markt befindlichen Testsysteme auf Grundlagenforschungen und Entwicklungen beruhen, die bereits zehn bis zwanzig Jahre zurückliegen.
Ein Vergleich der Leistungsmerkmale und Qualitäten der derzeit auf dem Markt verfügbaren Testsysteme gestaltet sich äußerst schwierig, da es - im Gegensatz zu den bekannten Benchmarks für Computer - keine objektiven und allgemein zugänglichen Verfahren bzw. Benchmarks zur Bewertung gibt. Die Halbleiterfirmen führen daher im eigenen Hause stichprobenartige Leistungsbewertungen von Testmaschinen nur am Beispiel ganz konkreter IC-Typen durch. Aus diesem Grunde kann ein objektiver Vergleich der Leistungsmerkmale aller auf dem Markt befindlicher Testsysteme weder in der Halbleiterindustrie noch in dieser Untersuchung durchgeführt werden.

Heutige Testsysteme sind bezüglich ihrer grundsätzlichen Architektur ähnlich aufgebaut (Abbildung 2.3): Sie enthalten im Testkopf verschiedene Steckplatzbereiche für bestimmte Modularten. Die Steckplätze sind in der Regel spezifisch für die Kartenart Pinelektronik, DSP-Frontends, programmierbare Strom-Spannungsquellen oder spezielle Funktionen vorgesehen. Die Karten werden von einem gemeinsamen Kommunikationsbus gesteuert und haben Zugriff auf Synchronisationssignale (Master Clock und weitere Synchronisationsleitungen). Die Spannungsversorgung erfolgt über eine Reihe gemeinsamer Netzteile. Die Ein- und Ausgänge der Module werden über ein Testerinterface-Board an der Prüflingsschnittstelle des Testers angeboten. Dieses Testerinterface-Board stellt zudem Querverbindungen zwischen den Modulen her.

Die heute geforderten Leistungsmerkmale können von diesen Testsystemen nicht mehr in befriedigendem Umfang erfüllt werden:

- Die Systeme sind unzureichend modular aufgebaut und können deshalb kaum flexibel an die oft sehr speziellen Anforderungen der Prüflinge angepaßt werden. Konkret ergeben sich durch eine derartige Struktur folgende Einschränkungen:
  - Die Konfigurationsmöglichkeiten sind auf die jeweiligen Slot-Sektionen limitiert
  - Das Testerinterface-Board begrenzt die Flexibilität ebenfalls

  - Auf diese Grundstruktur ist der Selbsttest-Adapter abgestimmt, der meist auch für die Grundkalibrierung verwendet wird
  - Die installierte Stromversorgung ist meist für eine bestimmte Konfiguration ausgelegt und damit nicht modular

- Die interne und externe Kommunikation der Testmaschinen erweist sich als limitierender Faktor, sie wird der zu erwartenden Datenflut nicht mehr gerecht.
  - Die Kommunikationskonzepte sind völlig ungeeignet für den Einsatz in einer modularen Architektur.
  - Die nicht ausreichende Geschwindigkeit und Breite der Datenübertragung führt zu Wartezeiten und verlängert damit die Laufzeit der Testprogramme und erhöht dadurch die Testkosten.

- Die starre Architektur aller auf dem Markt angebotenen Testsysteme unterstützt nur in unzureichendem Maß parallele Testlösungen (vor allem Multisite-Tests); damit können zunehmend Meßkanäle nicht effizient genutzt werden.

- Die traditionelle Testkopfarchitektur der Testsysteme bereitet bei den zukünftigen Technologien folgende Probleme:
  - Der Testkopf wird bei der geforderten zunehmenden Pinanzahl zu groß
  - Der Testkopf produziert zu viel Wärme, die mit traditionellen Methoden (Luftkühlung) sehr schlecht abgeführt werden kann
  - Die Kommunikation zwischen Testkopf und Hauptgestell des Testsystems ist durch die Signallaufzeit aufwändig und beeinträchtigt die Signalintegrität

Im gesamten Markt gibt es heute noch keine Testerfamilie, die auf diese neuen Anforderungen ausgerichtet ist. Verfügbare Systeme sind architektonisch ähnlich aufgebaut und weisen somit alle die gleichen Schwächen auf. Alle Testerhersteller sind folglich unter Druck, Neukonstruktionen auf den Markt zu bringen.

Um die Erfüllung zukünftiger Testaufgaben optimal und kostengünstig zu gewährleisten, ist ein neues Architekturkonzept zu realisieren, welches folgende Forderungen erfüllen soll:

- Ein hoher Grad an Modularität ist zu unterstützen. Damit kann eine konkrete Testmaschine je nach Kunde und Anwendung für einfache sowie komplexe Meßaufgaben applikationsspezifisch zusammengestellt werden, ohne daß - wie bisher - umfangreiche Anpassungsarbeiten, insbesondere bei der Betriebssoftware (auch Selbsttest und Kalibrierung), nötig werden.

- Die ITRS (International Technology Roadmap for Semiconductors) fordert die Erhöhung der Zahl der Testkanäle auf über 1000. Dazu sind die Kosten pro Kanal bis zum Jahr 2005 um mindestens 25 % zu senken, für Hochfrequenzkanäle sogar um mindestens ein drittel gegenüber dem heutigen Preisniveau. Für integrierte Schaltungen mit einer kleineren Zahl an Anschlüssen ist diese dennoch recht hohe Investition nur nutzbar, wenn mehrere Chips gleichzeitig getestet werden können (Multisite). Die Ressourcenzuordnung muß deshalb völlig neu geregelt werden, und zwar weitgehend automatisiert. Bisher ist es Stand der Technik, daß ein Prüfprogramm inklusive Loadboard zuerst einfach (Single Site) erstellt und getestet wird. Bei der Umstellung auf Multisite muß es komplett überarbeitet und auch ein neues Loadboard entwickelt werden. Dies bedeutet einen erheblichen Kosten- und Zeitaufwand. Ziel ist es, ein Testprogramm schnell - und zwar durch weitgehende Automatisierung - von Einfach- auf Mehrfachtest umstellen zu können.

- Die Entwicklung der Halbleiter im Mixed Signal-Bereich führt zur Kombination von hohen Auflösungen im Analogen bei gleichzeitigem Einsatz umfangreicher und leistungsfähiger digitaler Funktionen auf einem Chip. Außerdem steigt die Komplexität der verwendeten Messungen stark an, was beides zu größeren Datenmengen führt. Die Leistungsfähigkeit der internen Verbindungen zwischen den Baugruppen der Maschine ist deshalb um mindestens eine Größenordnung zu verbessern. Dazu reicht es nicht aus, nur die Datenraten zu erhöhen, es sind auch völlig neue Konzepte für die Verbindungsstrukturen selbst zu entwickeln (z.B. parallele Verarbeitung) und im Hinblick auf die neuen Testaufgaben zu optimieren.

- Im Bereich der mobilen Kommunikation sind bereits heute die vorhandenen Frequenzbänder ausgelastet. Zukünftige Standards werden sich bei Frequenzen über 10 GHz, kombiniert mit größeren Bandbreiten, bewegen. Dies erfordert neben der allgemeinen Verbesserung der Meßtechnik insbesondere deutliche Fortschritte im Bereich der Signalqualität und Signalführung zwischen Tester und Prüfling sowie neuartige Kalibrierkonzepte.

- Das Schreiben von Testprogrammen durch den Anwender muß wesentlich erleichtert und beschleunigt werden, um im Fertigungshochlauf neuer ICs von vornherein höchste Qualitätsanforderung bei geringsten Kosten zu erfüllen. Der Entwicklungsaufwand für diese Phase soll von den typischen sechs Personenmonaten auf etwa die Hälfte reduziert werden. Die Grundlage dafür ist eine neue wesentlich übersichtlichere und benutzerfreundlichere Architektur.

### 2.1.8 Zusammenfassung der Hardware-Analyse

Die evolutionäre Weiterentwicklung der Hardware führt nicht nur, wie noch gezeigt wird, zu Problemen bei der Modellierung. Die Hardware-Realisierung flexibelster Systeme führt vielmehr

zu Problemen bei der Software, denn:

- Ein hoher Grad an Flexibilität erhöht die Fehlerquote in der Treiberstruktur.
- Ein hoher Grad an Flexibilität kann nur mit Low-Level-Befehlen programmiert werden.
- Ein hoher Grad an Flexibilität ist nicht mit einer High-Level-Sprache (z.B. TSDL) darstellbar.

Um dem Ziel näherzukommen, schnell und effektiv Testprogramme zu schreiben, ist eine systemunabhängige High-Level-Sprache notwendig. Dies ist nur durch das Entfeinern der Hardware mit gleichzeitiger Unterstützung von Modularität möglich. Dazu müssen Interfaces der Prüflinge, wie zum Beispiel FireWire, USB usw. unterstützt werden.

Es wurde gezeigt, daß flexible Systeme eine Per-Pin-Architektur besitzen. Diese zieht einen extrem hohen Modellierungsaufwand nach sich und verlängert Simulationszeiten auf ein nicht abschätzbares Maß. Deshalb müssen solche Strukturen bei der Modellierung derartig abstrahiert werden, daß sich der Modellierungsaufwand reduziert, das Verhalten aber wie in der Realität erscheint. Bei der Simulation und Bewertung neuer Testsystem-Architekturen ist aber gerade auf die Modellierung dieser Aufteilung besonderer Wert zu legen, weil dadurch Engpässe in künftigen Systemen erkannt werden können.

Die angesprochene Per-Pin-Architektur kann als dezentrale Steuerung angesehen werden. In verfügbaren Testsystemen ist jedoch eine Mischform aus zentraler- und dezentraler Steuerung vorhanden, deren Berücksichtigung den Modellierungsaufwand ebenfalls erhöht.

## 2.2 Analyse der Software-Architektur

Die Steuerung eines Testsystems (beim virtuellen Test der Simulation) geschieht mit Hilfe von Prüfprogrammen, die in einer testsystemspezifischen Software-Architektur eingebetet sind. Um Testsysteme effektiv zu modellieren ist eine Analyse der Software-Struktur notwendig. Dies geschieht in den folgenden Abschnitten.

### 2.2.1 Bedienungsumgebung

In der Bedienungsumgebung muß der Testingenieur das Testprogramm entwickeln. Dazu stehen ihm eine Reihe von Hilfsmitteln (Abbildung 2.14) zur Verfügung, die er aus der zentralen

Bedienungsumgebung heraus aufrufen kann. Diese Werkzeuge dienen zum Debuggen des Testprogramms, zum Erzeugen von digitalen Pattern und analogen Signalen, zum Variieren von Parametern und der Abfolge von Testschritten, zur Unterstützung der Programmierung mittels graphischer Repräsentation der Instrumenten-Einstellungen und dergleichen mehr. Diese Werkzeuge liefern interaktiv ihre Daten an das Testprogramm bzw. stellen Werte dar, die sie aus den Meßergebnissen des Testprogramms beziehen.
Die Konzepte der Bedienungsumgebung spiegeln die Philosophie der Testsystemhersteller wieder. So ist es Geschmacks- und Gewohnheitssache des Testingenieurs, welche Art der Benutzerführung (graphisch oder textuell) er favorisiert. Es ist zu bedenken, daß es sich bei der beschriebenen Bedienungsumgebung nicht nur um die graphische Darstellung des Testsystems handelt. Vielmehr handelt es sich in seiner Gesamtheit um das Betriebssystem der Testmaschine, mit all seinen Steuerungs-, Treiber- und Verwaltungsstrukturen.

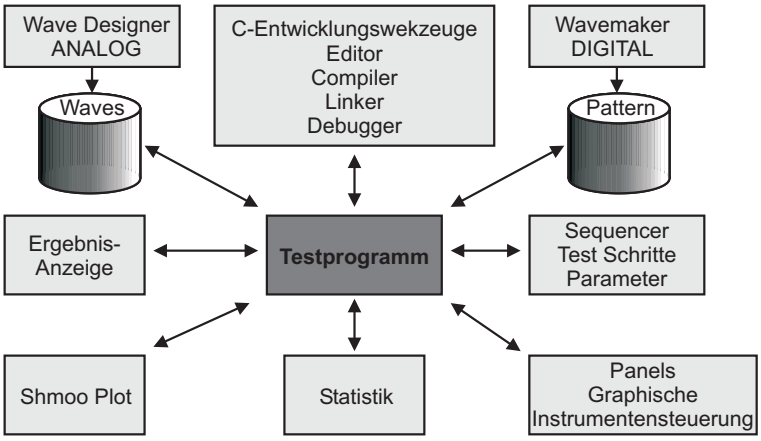



**Abbildung 2.14: Werkzeuge zum Entwickeln eines Testprogramms**

### 2.2.2 Hierarchischer Aufbau der Programmierumgebung

Das Testprogramm wird bei Credence und SZ in der Programmiersprache C entwickelt. Der Sprachumfang wird durch Bibliotheken mit testsystemspezifischen Befehlen erweitert. Beide Testsystemhersteller nehmen bei den Tester-Befehlssätzen eine Aufteilung in High- und Low-Level-Befehle vor. SZ unterscheidet zwischen Basic-, Master- und Super-Instrumenten, Credence zwischen Elementals und Primitives.

Die Befehle dieser Bibliotheken - egal aus welchem Level - können wie jede andere Funktion der Programmiersprache C verwendet werden. Dabei rufen High-Level-Befehle während ihrer Ausführung die Low-Level-Befehle auf. Im Gegensatz zu den Low-Level-Befehlen, die ebenfalls zur Programmierung verwendet werden können, bieten High-Level-Befehle folgende Vorteile:

- High-Level-Befehle beinhalten Break-Punkte, die vom Debugger aus angesprochen werden können
- Viele High-Level-Befehle beinhalten pass/fail - Entscheidungen, die mitprotokolliert werden können

Das Zusammenspiel zwischen den High- und Low-Level-Befehlen sowie die Einbettung zusätzlich notwendiger Software-Komponenten, wie Pattern und Debugger, zeigt Abbildung 2.15.

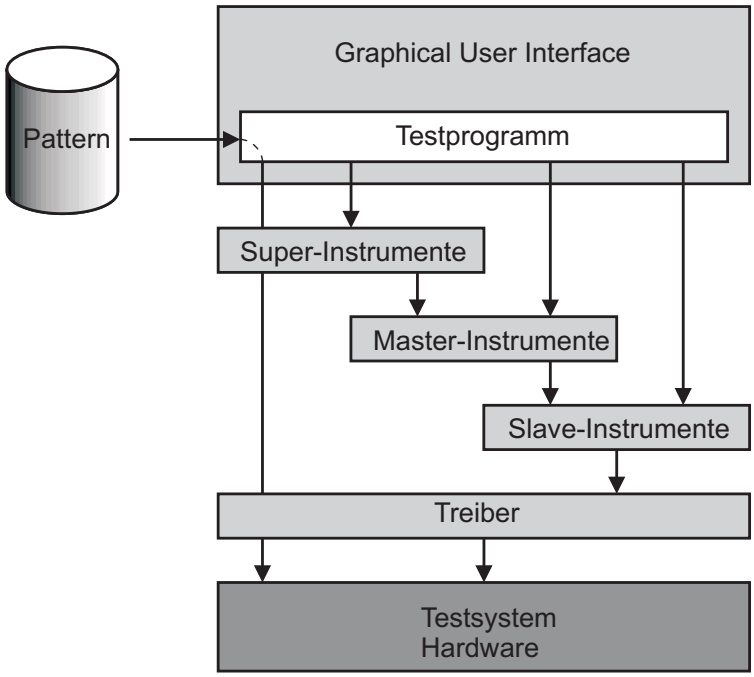



**Abbildung 2.15: Hierarchie der Software-Architektur**

Eine Beschreibung des hierarchischen Aufbaus der Software-Architektur am Beispiel der SZ M3650 findet sich im Anhang A.1.

### 2.2.3 Aufbau eines Testprogramms

Das Testprogramm wird, wie erwähnt, in der Programmiersprache C geschrieben. Zusätzlich wird auf testsystemspezifische Befehle zugegriffen, die dem Testingenieur in Bibliotheken zur Verfügung stehen. Auf diese Weise ist es nicht notwendig, sich mit der Kommunikation zwischen Steuerrechner und Testsystem auseinanderzusetzen. Der Anwender kann sich vollkommen auf die Programmierung der Instrumente konzentrieren. Die Abarbeitung der Instrumentenprogrammierung erfolgt rein sequentiell. Es gibt in der Programmierung keinerlei Möglichkeiten, parallele Vorgänge zu definieren und ausführen zu lassen. Dies gilt sowohl für den Bereich der

Instrumentenbefehle als auch für die Ausführung der verschiedenen Testschritte. Die Architektur unterstützt in der momentanen Ausführung keine parallelen Vorgänge.

Das Testprogramm untergliedert sich in mehrere Hauptgruppen, die sequentiell abgearbeitet werden. Diese Gruppen lassen sich wie folgt untergliedern:

- Initialisierung
- Setup
- Testschritte
- Shutdown

Diese Gruppen werden im Anhang A.2 näher erklärt.

### 2.2.4 Programmierung der Pinelektronik (DPIN)

Der DPIN, der digitale Signalfolgen generiert und auswertet, fällt in die Gruppe der Master- und Super-Instrumente. Im Folgenden soll ein Überblick über die softwaretechnische Ansteuerung dieser Pinelektronik gegeben werden.

#### 2.2.4.1 Befehle und Panels

Das Panel (Abbildung 2.16) stellt die grundlegenden Funktionen des DPIN dar. Die Steuerung der digitalen Treibereinheit kann über den Patternspeicher, DIO-Lines, ... erfolgen. Die Steuerung der aktiven Last kann über logische Signale oder Triggerleitungen durchgeführt werden. Im digitalen Komparator werden die Prüflingsantworten mit Sollwerten im digitalen Pattern verglichen. Mit der Auswertung dieser Signale können weitere Instrumente über Triggerleitungen gesteuert werden.

Digitale Signale zum Prüfling bzw. vom Prüfling sind aber ebenso elektrische Größen, die mit Hilfe anderer Meßgeräte (z.B. Voltmeter) ausgewertet werden können. Um derartige Verschaltungen zu diesen Instrumenten zu ermöglichen, steht der analoge Bus zur Verfügung (Abschnitt 2.1.2).

Die LPMU (Local Parametric Measurement Unit) kann über alle vier Quadranten Ströme und Spannungen forcieren sowie messen. Pro Pin ist eine PMU verfügbar. Aus Gründen der Übersichtlichkeit wird ihre Funktionalität durch ein eigenes Panel, das in das DPIN-Panel integriert ist, repräsentiert (Abbildung 2.17).

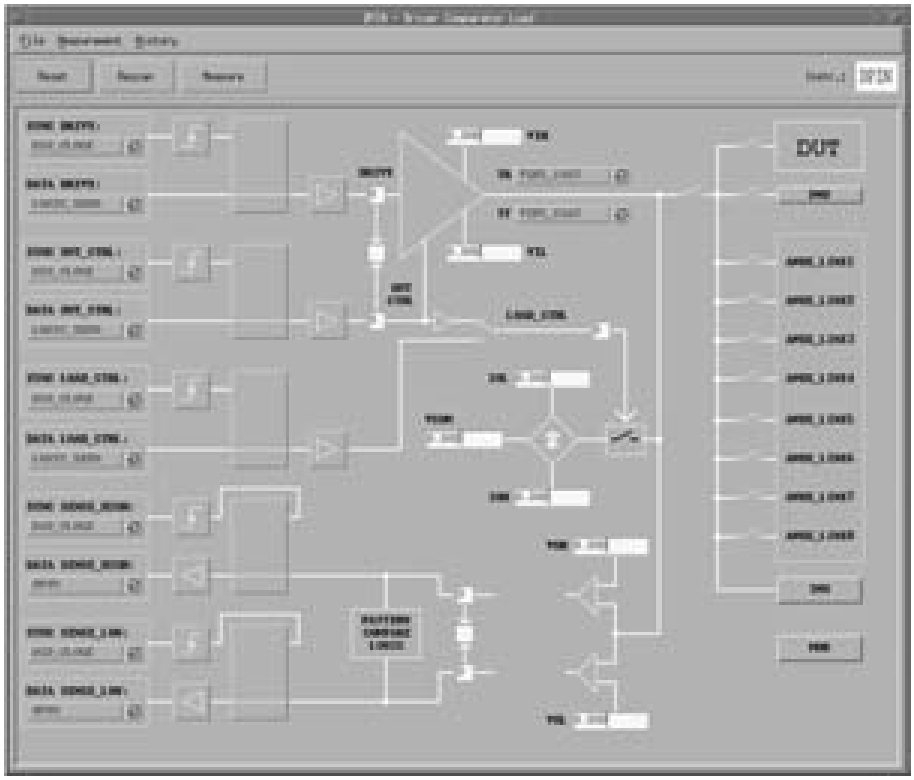



**Abbildung 2.16: DPIN-Panel**

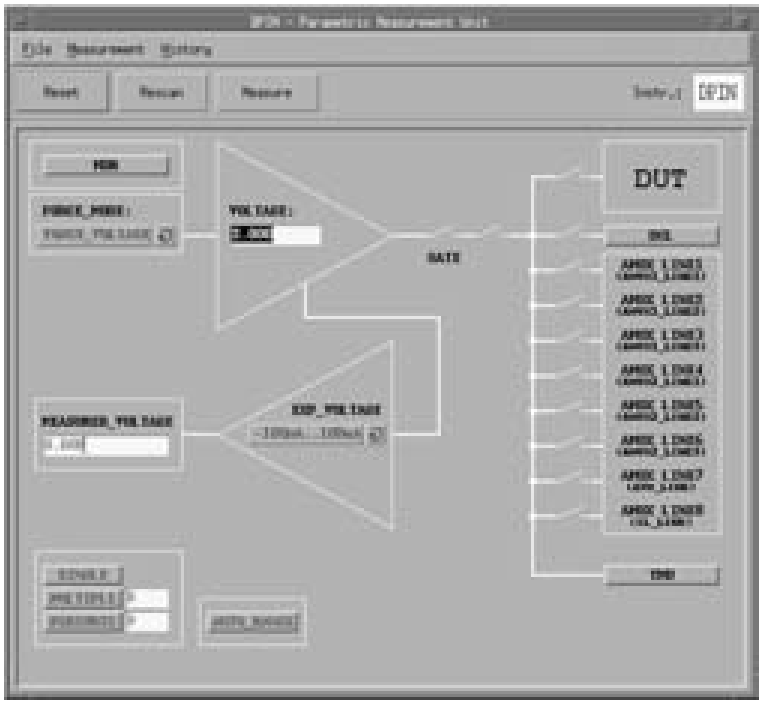



**Abbildung 2.17: PMU-Panel**

Eine Besprechung sämtlicher Befehle, die durch die Panels erzeugt werden können, wäre hier zu umfangreich und kann im Handbuch [SZ00] nachgeschlagen werden. Im Folgenden soll aber näher auf die Steuerung mit Hilfe digitaler Pattern eingegangen werden.

#### 2.2.4.2 Patterndaten

Um heutige Bausteine mit digitalen Signalfolgen zu stimulieren, sind große Datenmengen notwendig. Diese Daten werden in sogenannten Pattern zusammengefasst. Das sind Dateien, in denen die zeitliche Abfolge von High- und Low-Pegeln sowohl für die Treiber als auch für die Komparatoren festgelegt sind.

Zunächst werden allgemeingültige Formate erklärt, mit deren Hilfe Signalformen beschrieben werden. Anschließend wird auf Dateiformate zur Speicherung der Patterndaten eingegangen.

Standardformate zur Signalform-Beschreibung
In der Begriffswelt der Testsysteme für digitale Bausteine sind 'Format'-Bezeichnungen zur Klassifizierung von Signalverläufen üblich. Treiberformate werden klassifiziert aus der Abfolge der Signalzustände während des Prüftaktes ('Return-to-x'), Empfängerformate aus der Dauer des Signalvergleichs ('compare event / window').

- Treiber-Formate
  - Non-return-to-zero-Format (NRZ)
    Während der gesamten Dauer einer Prüftakt-Periode wird ein konstanter Pegel an den Prüfling angelegt. Ein eventueller Signalübergang erfolgt nur zu Beginn oder Ende der Prüftaktperiode.
  - Delayed-non-return-to-zero-Format (DNRZ)
    Genauso wie beim NRZ-Format liegt das Signal für die Dauer einer Prüftaktperiode an, allerdings werden die Flanken um einen programmierbaren Wert verschoben.
  - Return-to-zero-Format (RZ)
    Der logische Wert 1 des Testvektors liegt nicht während der ganzen Dauer einer Prüftaktperiode an. Das Prüfsignal beginnt mit dem Logikwert 0, nimmt kurzfristig den logischen Wert 1 an und kehrt nach einer bestimmten Dauer wieder auf den Logikwert 0 zurück. Der Logikwert 0 in einem Testvektor führt dazu, daß das Prüfsignal während der gesamten Prüftaktperiode auf dem Wert 0 bleibt.
  - Return-to-one-Format (RO)
    Es gilt sinngemäß die Beschreibung des RZ-Formats auf die logische 1 bezogen.

  - Return-to-one-inverted-Format (ROI)
    Der negierte Wert des RO-Formats wird geliefert. Der Unterschied zum RZ-Format besteht darin, daß im Falle einer logischen 0 im Testvektor beim RZ-Format kein Impuls erzeugt wird. Beim ROI-Format hingegen tritt ein Impuls auf, da die logische 0 im Testvektor wegen der RO-Definition im ROI-Format zu einer 1-0-1-Folge und nach Invertierung zu einer 0-1-0-Folge führt.
  - Surround-by-Complement-Format (SBC)
    Der logische Wert des Testvektors wie bei der Prüfsignalaufbereitung mit den jeweils invertierten Werten umschlossen.

- Empfänger-Formate
  Die Reaktionen des Prüflings werden vom Testsystem erfaßt und mit den vorgegebenen Sollwerten verglichen.

  - Strobe-Compare-Format
    Die Abtastung eines Prüfsignals erfolgt zu einem Zeitpunkt. Liegt zu diesem Zeitpunkt das Ausgangssignal des Prüflings im Sollbereich, ergibt dieser Testschritt eine 'Pass'-Meldung, ansonsten eine 'Fail'-Meldung.
  - Window-Compare-Format
    Es wird sichergestellt, daß das Prüflingsausgangssignal während eines ganzen Zeitintervalls stabil ist und den beim Vergleich vorgegebenen Pegel-Schwellwert nicht über- bzw. unterschreitet. Verläßt das Prüfsignal den zulässigen Pegelbereich auch nur einmal während des vorgegebenen Zeitintervalls, ergibt dieser Zeitfenster-Vergleich eine 'Fail'-Meldung

Die Patterndaten werden in Dateien gespeichert. Hierzu stehen verschiedene Formate zur Verfügung:

- WDB (Waveform-Data-Base) [SZ00]
  Die Waveform-Data-Base (WDB) ist eine maschinenunabhängige Datenbasis. Sie stellt die zentrale Basis bei der Arbeit mit Pattern dar, denn von hier aus werden testsystemspezifische Patterncodes sowie menschenlesbare systemunabhängige Austauschformate generiert. Die WDB ist keine einzelne Datei, sondern eine ganze Verzeichnisstruktur mit mehreren Verzeichnisebenen und Dateien.

- WGL (Waveform-Generation-Language) [SZ00]
  Das WGL-Format als Form der Patterndarstellung ist relativ übersichtlich und auch vom Menschen lesbar. Die Daten liegen in diesem Format im ASCII-Code vor.

- PPS (Pattern-Pin-Sequencer) [SZ00]
  Das PPS-Format ist ein maschinenspezifisches Format, das speziell für den DPIN von SZ entwickelt wurde. Es ist nicht vom Menschen lesbar.

Eine eingehendere Beschreibung der Dateiformate findet sich im Anhang A.3.

### 2.2.5 Zusammenfassung der Software-Analyse

In der Software spiegelt sich der eigene Stil der Hersteller wider. Deshalb ist es schwierig die ATE-Hersteller auf ein einheitliches Konzept zu einigen. Vorteilhaft ist, daß Testsystemhersteller verschiedene Beschreibungsebenen unterstützen und dabei die Programmierung auf hoher Ebene forcieren, die allerdings im Vergleich zu allgemeinen Beschreibungssprachen noch zu detailliert ist. Das zeigt, daß die bisherigen Standardisierungsvorschläge wie TSDL oder STIL nicht ausreichend ausdrucksfähig und damit nicht attraktiv sind. Diese Sprachen decken nur allgemeine Fälle ab [MIE99]. Detailuntersuchungen am Prüfling sprechen Feinheiten der Hardware an, die jeder Testingenieur auf seine eigene Weise löst. Hierbei verwendet er nicht einmal die abstrakte Programmierung des Testsystems. Somit ist dies mit einer High-Level Beschreibungssprache nicht programmierbar. Ein Testprogramm, das in einer High-Level-Sprache wie STIL vorliegen würde, müsste auf eine testsystemspezifische Sprache umgesetzt werden. So eine Konvertierung kann nur lückenhaft sein, solange sich die Testsystemhardware nicht an der Sprache orientiert und die beschriebene Flexibilität und Kombinatorik vorhanden ist.

# Kapitel 3

# Stand der Technik

## 3.1 Vergleich kommerzieller Virtual Test Produkte

### 3.1.1 Untersuchungskriterien

#### 3.1.1.1 Benutzerumgebung

Der „Virtuelle Test“ wurde in erster Linie zur Verifikation von Testprogrammen konzipiert. Die Zielgruppe sind also Testingenieure, und sie sollten einen leichten Zugang zu diesen Werkzeugen finden. Dies kann nur mit Hilfe einer komfortablen oder gewohnten Benutzerumgebung geschehen. Ein Testingenieur ist in der Regel kein Experte auf dem Gebiet der Simulation. Interviews mit Applikateuren ergaben sogar, daß oftmals eine gewisse Scheu und Ablehnung gegen Simulationsumgebungen vorhanden ist. Deshalb ist es notwendig eine Softwareumgebung zu schaffen, die eine leichte Anwendung des virtuellen Tests gestattet. Wünschenswert ist eine Umgebung, bei welcher der Simulator komplett im Hintergrund steht und aus der gewohnten Arbeitsumgebung des Testingenieurs wie ein realer Tester bedient werden kann. Die zusätzlichen Möglichkeiten, die ein Testprogrammlauf in einem Simulator bietet, können, aber müssen nicht genutzt werden. Viel wichtiger ist es, alle Funktionen der Testersoftware zu unterstützen. Dies ist die Ausgangsbasis, um Simulation im Bereich der Testprogrammentwicklung zu etablieren.

Das Untersuchungskriterium „Benutzerumgebung“ bewertet den Bedienungskomfort, die Nähe zur Testsystemsoftware und die Einbindung des virtuellen Tests in ein Gesamtsimulationspaket.

#### 3.1.1.2 Funktionsumfang

Damit die virtuelle Testumgebung in der Praxis effizient eingesetzt werden kann, muß sie einen gewissen Funktionsumfang abdecken. Dieser läßt sich nach verschiedenen Gesichtspunkten untersuchen.

Werden alle Instrumente der realen Hardware in Modellen unterstützt?
Eine Testmaschine setzt sich aus vielen verschiedenartigen Instrumenten zusammen (siehe Abschnitt 2.1). In der Simulationsumgebung müssen alle Instrumente durch Software-Modelle repräsentiert werden. Fehlt für ein Instrument das entsprechende Modell, kann dessen Programmierung im Testprogramm nicht verifiziert werden.

Wird der testsystemspezifische Sprachumfang abgedeckt?
Für jedes Instrument gibt es spezifische Befehle, die in einer Bibliothek zusammengefaßt sind (siehe Abschnitt 2.2). Jeder dieser Befehle ruft in einem Instrument eine Reaktion hervor. Dieses Verhalten muß im entsprechenden Software-Modell der Instrumente nachgebildet werden.

Werden die Funktionen der Bedienungsumgebung abgedeckt?
In der Bedienungsumgebung eines ATE gibt es Funktionen, die dem Debugging dienen (Panels, Scope, Result-Display, Shmoo-Plots, ...)(siehe Abschnitt 2.2.1). Diese Funktionen basieren auf einem intensiven Datenaustausch zwischen Bedienungsumgebung und realer Hardware. Der Simulator muß folglich den Funktionen die benötigten Daten in geeigneter Weise zur Verfügung stellen können, um deren Benutzbarkeit sicherzustellen.

Werden zusätzliche Informationen über Signale angeboten?
In einem Simulator sind alle berechneten Signale grundsätzlich verfügbar. Damit sind nicht nur Signale auf dem Loadboard gemeint, sondern auch bauteil- und testerinterne Signale. Diese zusätzlichen Daten stellen ein großes Informationspotential bei der Fehlersuche im Testprogramm dar. Diese Daten müssen dem Testingenieur aber durch Simulatorwerkzeuge zur Verfügung gestellt werden.

Werden zusätzliche Funktionen speziell für den virtuellen Test angeboten?
Da in der Simulationsumgebung mehr Informationen als an einem realen Testsystem verfügbar sind ist es möglich, diese in der Bedienoberfläche zur Verfügung zu stellen. Dies stellt die komfortable Erweiterung des vorangegangenen Punktes dar.

Das Untersuchungskriterium „Funktionsumfang“ bewertet die Vollständigkeit der Simulationsumgebung im Verhältnis zur realen Testumgebung und die Verfügbarkeit von zusätzlichen Simulatorinformationen.

#### 3.1.1.3 Simulationstiefe

Simulationsgenauigkeit steht im Widerspruch zu Simulationsgeschwindigkeit. Je genauer Detaileffekte simuliert werden, desto länger dauert eine Simulation und je schneller simuliert werden soll, desto ungenauer oder abstrakter sind die Simulationsergebnisse.

Das Untersuchungskriterium „Simulationstiefe“ bewertet die Genauigkeit der Simulationsergebnisse. Daraus kann eine Festlegung der Simulationstiefe bzw. der Abstraktionsebene erfolgen.

#### 3.1.1.4 Modellgenauigkeit

Die Genauigkeit, mit der die Modelle die Realität abbilden, konnte nicht untersucht werden. Nur aufgrund der Demos, die auf Messen und Workshops vorgeführt werden, ist das nicht möglich. Weitergehende Informationen waren jedoch von den Anbietern nicht zu erhalten.

#### 3.1.1.5 Performance

Die Simulationsgeschwindigkeit wäre ein ideales Kriterium, um die verschiedenen Werkzeuge in Abhängigkeit von der Simulationstiefe zu vergleichen. Doch auch hier kann man die Vorführungen nicht als Maßstab anwenden, denn bei jedem Anbieter liegen unterschiedliche Testprogramme in den Demonstratoren zugrunde. Einheitliche Benchmarks standen nicht zur Verfügung, somit sind die Vorführungen nicht vergleichbar.

### 3.1.2 Produkte

Die Produkte verschiedener Anbieter werden vorgestellt und anhand der Untersuchungskriterien bewertet. Abbildung 3.1 gibt einen ersten Überblick, in welchem Stadium der Testprogrammentwicklung die Produkte Einsatz finden.

#### 3.1.2.1 Dantes (IMS)

Integrated Measurement Systems, Inc. - gegründet 1983 - entwickelt, produziert, vermarktet und betreut eine komplette Produktlinie von verschiedenen Hochleistungs- und kosteneffektiven IC-Testmaschinen, die vor allem im Bereich der Validierung, Verifikation und Charakterisierung komplexer IC's eingesetzt werden.
Zusätzlich entwickelt IMS Virtual Test Produkte, die im Bereich digitale Test-Pattern-Generierung und Verifikation sowie zu deren Debugging eingesetzt werden.

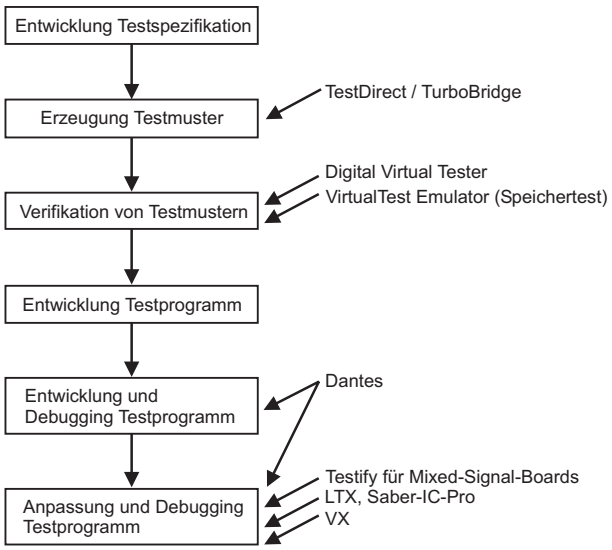



**Abbildung 3.1: Einsatzgebiete der Virtual-Test-Produkte**

> „Dantes ist eine Test-Entwurfs- und Verifikationsumgebung, die die graphische Übernahme der Tests, das Erstellen von Modellen, das Überprüfen der Design-Regeln, die Simulation, die ATE-spezifische Code-Generierung, die Adapter-Entwicklung sowie die Verifikation und die Dokumentation von Mixed-Signal-Tests unterstüzt. Integriert in die EDA-Technologie von Cadence Design Systems ermöglicht Dantes dem Testingenieur, die Design-Datenbasis zu erweitern. Zusätzlich können gemeinsame Tools während des gesamten Test-Entwicklungsprozesses verwendet werden.“ [DAN95]

Im Rahmen der Toolbox-Entwicklung für die Credence Testsysteme VISTA-VISION und DUO wurde am LRS eine Evaluierung des Software-Pakets durchgeführt. Durch die ausgiebige Untersuchung des Softwarepakets können detaillierte Angaben über Bedienung und Performance gegeben werden.

Der Testingenieur gibt in Dantes jeden Testschritt als Schematic ein. Dazu ist das gewünschte Instrument aus der Bibliothek verfügbarer Testerinstrumente auszuwählen. Das zugehörige Symbol wird auf der graphischen Eingabeoberfläche plaziert und mit dem zugehörigen Pin des DUT-Modells „verdrahtet“. Auf diese Weise werden alle für den Testschritt notwendigen Instrumentenmodelle angeschlossen. Es entsteht ein Schematic, das die Beschaltung eines Testschritts repräsentiert (Abbildung 3.2).

Das Setup der Instrumente nimmt der Testingenieur über symbol- und damit instrumentenspezifische Einstellungs-Formulare vor (Abbildung 3.3). In diesen Formularen wird auch der Import analoger oder digitaler Daten spezifiziert, die im testerspezifischen Format vorliegen.

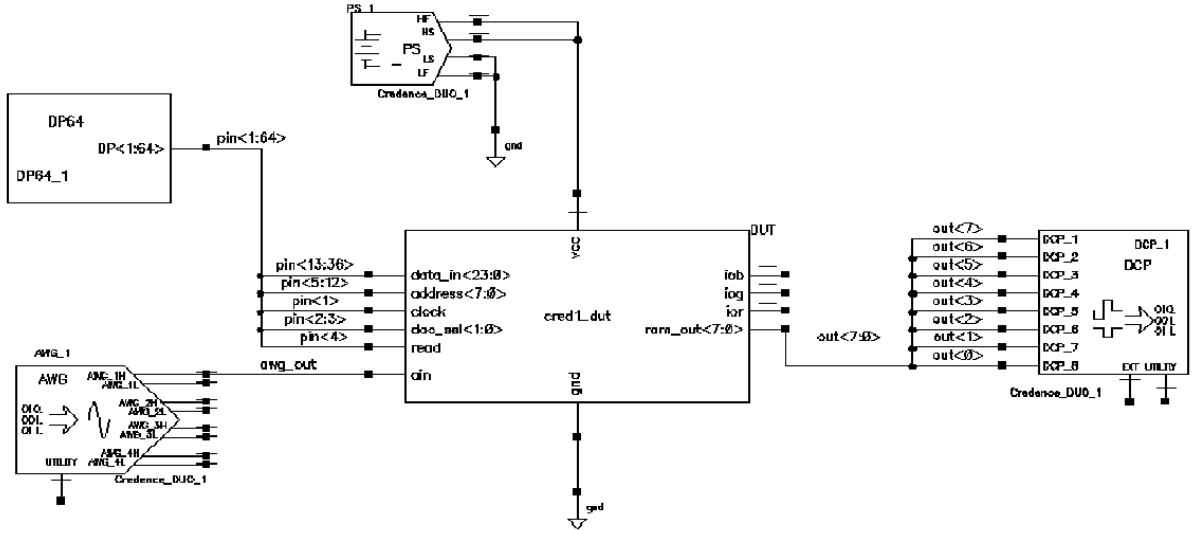



**Abbildung 3.2: Schematic eines analogen-digitalen Tests**

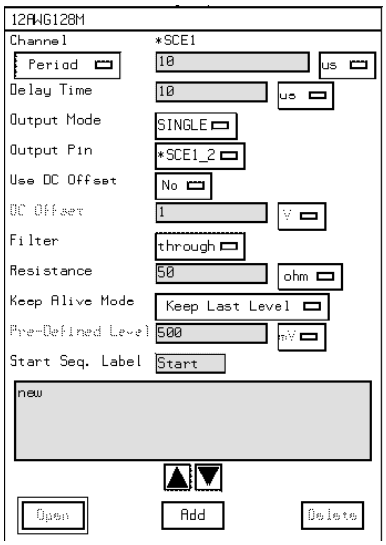



**Abbildung 3.3: Formular eines AWG**

Nachdem ein Testschritt derartig generiert wurde, führt Dantes einen Test-Rule-Check durch, der garantiert, daß spezifische Regeln des Zieltestsystems innerhalb des Testschritts nicht verletzt wurden.

Anschließend kann die Simulation des Testschritts durchgeführt werden. Dabei liefert Dantes die Stimuli der Instrumente zum DUT-Modell, erfaßt dessen Antworten und überprüft, ob diese korrekt sind (Abbildung 3.4). Dantes' „TurboModel Emulation Technology" unterstützt interaktive Testsimulation, bei der die Simulation an jedem beliebigem Zeitpunkt angehalten werden kann [DAN97-1]. In einem derartigen Haltepunkt ist es möglich, alle Instrumentenparameter zu überprüfen (z. B. Spannung, Waveforms DUT-Antworten), zu modifizieren und die Simulation fortzusetzen. Das Herzstück von Dantes bilden die sogenannten TurboModelle des mixed-signal Testsystems, basierend auf Verilog und SpectreHDL. (Im Rahmen der Evaluierung war es allerdings nicht möglich den Source-Code der Modelle zu untersuchen, um so eine Aussage über die Abstraktion in den Modellen zu geben.)

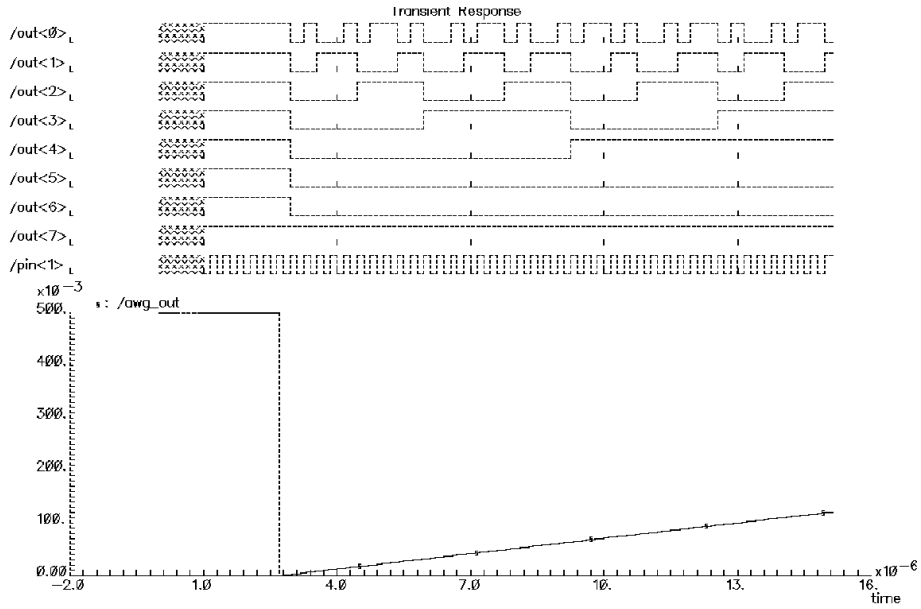



**Abbildung 3.4: Stimulus und DUT-Antwort beim ADC-Test**

Nach Eingabe der Pass-/Fail-Grenzen für jeden Testschritt generiert Dantes kompilierfertigen ATE-spezifischen Testcode, der zusammen mit den analogen und digitalen Daten, die schon in der Simulation eingesetzt wurden, auf dem Zieltestsytem ausgeführt werden kann.

IMS bezeichnet Dantes als selbstdokumentierend, da jedes Schematic den Einsatz und die Verbindung mit dem DUT dokumentiert.

#### 3.1.2.2 Digital Virtual Tester (IMS)

VLSI Design Werkzeuge enthalten einen Automatic Test Pattern Generator (ATPG), der es dem Designer erlaubt, einen Vektorsatz zu generieren, der eine hohe Fehlerabdeckung garantiert. Üblicherweise liegen diese Vektoren nicht in einem Format vor, das von den meisten kommerziellen Testern verarbeitet werden kann. Folglich ist eine Übersetzungssoftware nötig, die diese Vektoren in das testerinterne Format umsetzt (siehe Abschnitt 2.2.4.2).

Obwohl die Konvertierung der Simulationsdaten in ein Patternformat für den Zieltester automatisch funktioniert, verursacht dieser Vorgang häufig Fehler, die erst beim Zusammenspiel mit den ersten Prototypen auffallen. Fehlende Pinfunktion, falsches Timing, Simulationsfehler, Übersetzungsfehler, Initialisierungsfehler und I/O-Fehler sind die häufigsten Fehler, die in den Pattern bei der Übersetzung auftreten und die der Testingenieur suchen und beheben muß. [DVT98-1]

Das Debuggen des Patterns konnte bisher nur Vektor für Vektor mit den ersten Prototypen auf einer realen Testmaschine vorgenommen werden. Folglich vergehen Tage oder Wochen zum vollständigen Debuggen der Testpattern. Obwohl die Zeit zum vollständigen Debuggen mit Hilfe des DVT (Digital Virtual Tester) die gleiche bleibt, wird der gesamte Produktentwicklungszyklus verkürzt, da die Debuggingphase zeitlich nach vorne gezogen wird und somit der kritische Zeitpfad entschärft wird. [DVT98-1]

Arbeitsweise des DVT:
Das Benutzerinterface, das auf Java basiert, steuert ein Simulationsmodell des Zieltesters an, das den Speicher des Testsystems, die dem Timing zugrundeliegende Architektur und digitale Pins repräsentiert. Der Digital Virtual Tester arbeitet mit Pattern, die mit Hilfe von TestDirect (IMS) oder anderen Patternkonvertern erzeugt wurden. Die Modelle des zu testenden Bausteins müssen in Verilog oder VHDL vorliegen. Sobald die Verbindung zwischen Testsystem-Pins und Bauelement-Pins eingegeben wurde, Timing und Pattern Daten geladen wurden, kann die Simulation auf Knopfdruck gestartet werden. Sobald die Simulation abgeschlossen ist, kann sich der Testingenieur die Ergebnisse durch ein beliebiges Verilog Waveform Display Tool oder durch ATE spezifische Werkzeuge anzeigen lassen, um Fehler zu untersuchen und Probleme aufzudecken. [DVT97]

Der Digital Virtual Tester unterstützt die Patternformate folgender Testmaschinen: Advantest T33xx, Agilent 83000, Agilent 9490 and 94000, Schlumberger ITS9000, Teradyne J750, Teradyne J971 [DVT00].

Die Einsatztauglichkeit konnte im Rahmen der Arbeit nur über Berichte, wie zum Beispiel den von Nat Reeves von Level One Communications [DVT98-1] in Erfahrung gebracht werden.

Zur Evaluierung wurde dort ein Baustein mit 70k digitalen Gates, Speicher und analogen Schaltungsgruppen herangezogen.
Die funktionalen Pattern bestehen aus 300k Vektoren, aufgeteilt in 15 Sets und zusätzlich noch 1,2M Scan-Vektoren.

Dabei konnten 4 Fehler-Kategorien lokalisiert werden, die mit Hilfe des DVT nicht entdeckt wurden:

- Testbench Omissions: Vektoren, die nur der Designer in seiner Testbench zur Stimulierung des DUT benötigt. Bei der Übertragung an ein Testsystem sind diese Vektoren zu löschen.
- Timing Errors: Fehler im Timing können aufgrund der idealen Testermodelle oftmals unentdeckt bleiben.
- Contention issues: Dieser Fehler tritt typischerweise auf, sobald an einem bidirektionalen DUT-Pin das DUT treiben will, während sich der Tester seinerseits noch im Treibermodus befindet.
- Nonfunctional Pattern: Fehlerhafte Vektoren müssen von Hand zurückverfolgt werden, um zu klären, ob der Fehler bei der Übersetzung entstanden ist oder ob der Vektor schon in der Testbench fehlerhaft war. Im zweiten Fall ist es die Aufgabe des Designers den Fehler zu lokalisieren.

Die Untersuchung der aufgedeckten Fehler in dieser Evaluierung ergab, daß Virtual Test die Zusammenarbeit zwischen Design und Test fördert und so frühzeitig Design- und Patterngenerierungsfehler erkannt werden können.

Die Erwartungen von IMS gehen dahin, daß bei Einsatz dieses Produkts die Debugging-Phase am realen Tester um die Hälfte reduziert werden kann. [DVT98-1]

#### 3.1.2.3 Virtual Test Emulator (IMS)

Den Virtual Test Emulator gibt es für die Memory Test Systeme Versatest V1300/V1000 von Agilent sowie Kalos von Credence. Er enthält ein digitales Modell der Testmaschine sowie einen Testprogramm-Emulator.

Der Designer entwickelt ein Simulationsmodell und eine Testbench für sein Bauelement. Um diese Vorarbeiten für den Testingenieur nutzbar zu machen, muß das DUT-Modell aus der Testbench herausgelöst werden. Der Testingenieur kann dann in seiner gewohnten Testumgebung

das Testprogramm sowie das Pattern entwickeln. Unter der Verwendung des Virtual Test Emulators als Simulationswerkzeug für den Tester kann er dann sein Testprogramm ausführen und debuggen. Die Bausteinantworten werden von dem Modell geliefert.

Mit Hilfe der Analyse-Funktionen des VHDL-Simulators kann der Testingenieur auch interne Knoten des Bauelements beobachten, um so die Funktion des DUT besser zu verstehen und Fehler besser analysieren zu können. Zusätzlich werden Standard-Ausgaben des Testsystems unterstützt.

#### 3.1.2.4 TestDirect / TurboBridge

TestDirect von IMS ist ein Programm zur Erzeugung von Testpattern, das den Prozeß der Generierung von testsystemspezifischen Pattern aus der Testbench des Designers heraus unterstützt. Damit ist es mehr in die Testprogrammentwicklung einzuordnen.

TestDirect dokumentiert das Verhalten des Prüflings durch die Verwendung von Kurvenformen in Zeitdiagrammen. Daraus wird ein Code generiert, der mit der Simulation des Designs gemeinsam ausgeführt wird. Während des Simulationslaufs werden die Aktivitäten des Designs protokolliert und daraus direkt zyklische Pattern generiert. TestDirect kombiniert diese Pattern mit der Kurvenform für den Zieltester und generiert so testerspezifische Pattern.
Damit greift TestDirect nicht in die Vorgehensweise des Designers ein, vielmehr weitet es die Simulation aus, so daß der Testingenieur direkten Nutzen daraus ziehen kann.

TestDirect kann in Kombination mit DANTES verwendet werden, um das gesamte Pattern in einer Mixed-Signal-Umgebung zu verifizieren. Von einer Bewertung dieses Werkzeugs wird abgesehen, da es mehr zur Unterstützung des virtuellen Tests dient.

#### 3.1.2.5 Testify (Analogy)

Analogy, zwischenzeitlich in Avant! und jetzt in Synopsys integriert, bietet Simulations- und Design-Werkzeuge an, die es dem Kunden ermöglichen, virtuelle Prototypen und Software-Modelle zu entwickeln, die den Design- und Herstellungsprozeß unterstützen. Mit dem Simulator SABER® (siehe Abschnitt 4.1.2) ist Analogy für lange Zeit Marktführer im Bereich Mixed-Signal-Simulation gewesen.

TESTIFY stellt eine Simulationsumgebung für Testingenieure zur Verfügung, die Diagnose- und Testprogramme für Mixed-Signal-Boards entwickeln. Es wurde konzipiert, um die Anforderungen der Testingenieure zu erfüllen, die Testschritte für Luft- und Raumfahrt- sowie militärische

Anwendungen zu programmieren. Dieses Werkzeug hilft ihnen, die Qualität ihrer Testprogramme zu verbessern. Testify liefert hierzu Informationen über das normale Verhalten der zu testenden Baugruppe (UUT - Unit Under Test) und wie gut die Testschritte programmiert sind, indem ein fehlerhaftes Verhalten der UUT simuliert wird. Damit ist es möglich, Diagnose-Programme zu programmieren sowie Sicherheits- und Zuverlässigkeits-Untersuchungen durchzuführen [ANA98-1]. Testify stellt eine Speziallösung mit integrierten Fehlermodellen für den Board Test dar. Dies ist eine Anwendung, die zwar mit dem virtuellen Test verwandt ist, ihn aber in der hier behandelten Form nur am Rande berührt, deshalb wird hier auf eine Bewertung verzichtet.

#### 3.1.2.6 Saber-IC Pro (Analogy)

Saber-IC Pro ist der Mixed-Signal-Simulator für die Virtual Test Umgebung. Er erfüllt die Anforderungen der Hersteller von Großserien ICs und ASICs. Das Paket basiert auf dem Saber analog und mixed-signal Simulator und nutzt die Vorteile von Verilog-XL oder ModelSim zur Simulation der Verilog oder VHDL-Teile des Designs.

Saber-IC Pro ist der Simulator für die Virtual Test Umgebungen, die von Teradyne und LTX angeboten werden. Beide Hersteller haben eine in sich geschlossene Lösung für Virtual Test entwickelt, die es den Testingenieuren erlaubt, funktionale Tests ohne benötigte Hardware zu entwickeln.

Die spezifischen Testerbefehle werden im IMAGE (Teradyne) oder Device Tool (LTX) verarbeitet und an den ATE-Emulator übergeben (Abbildung 3.5) (Definition Testsystem-Emulator siehe Anhnang D). Verilog PLI oder ModelSim FLI bearbeiten die prozessierten Emulator-Kommandos und liefern diese an Saber/Verilog-XL oder Saber/ModelSim. Die Modelle in der Simulation enthalten Stimulus- und Meßinstrumente, DIB und DUT. Die Simulationsergebnisse werden vom Simulator zum Emulator zurückgeliefert. Somit ist die Umgebung in sich geschlossen, denn der Testingenieur kann die Ergebnisse wie an der realen Testmaschine bearbeiten und verwalten. [ANA98-2]

Auf eine Bewertung wird hierbei verzichtet, da dieses Produkt in die Umgebungen von Teradyne und LTX eingebunden wird und dort zur Diskussion herangezogen wird.

#### 3.1.2.7 LTX, Motorola, Analogy

Das Konsortium, bestehend aus den Partnern LTX als Testsystemhersteller, Motorola als IC-Hersteller und ATE-Anwender und schließlich Analogy als Simulator-Anbieter, stellte auf mehreren Konferenzen eigene Vorschläge zu diesem Thema vor.

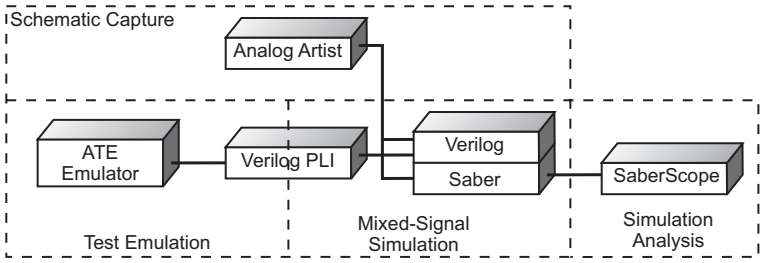



**Abbildung 3.5: Saber-IC Pro Virtual Test Umgebung**

Die angewendete Virtual Test Umgebung wurde von LTX und Analogy entwickelt. Sie basiert auf dem Simulator SABER® (siehe Abschnitt 4.1.2) und Cadence Design Framework.

Die Symbole für DUT, Testerinstrumente und weitere notwendige Bauelemente des Loadboards werden im Cadence Schematic Capture Tool plaziert. Dieses Werkzeug wurde in die LTX-Umgebung integriert und wird dort DeviceTool genannt. Es generiert eine Netzliste, die in den Simulator geladen wird.
Der Anwender sieht nur den Testprogramm-Editor und hat über ihn die Möglichkeit, Testschritte sowie einzelne Befehle auszuführen. Jeder Befehl kann Parameteränderungen im Simulator nach sich ziehen, wie zum Beispiel das Öffnen bzw. Schließen von Relais oder Spannungsänderungen. Jeder Befehl hat eine individuelle Ausführungsgeschwindigkeit, die exakt in simulatorspezifische Zeitschritte umgesetzt wird. Am Ende der Befehlsausführung wird ein Ergebnis vom Simulator in das Testprogramm zurückgelesen.
Die analogen Blöcke des DUT werden in MAST® modelliert, die digitalen Teile werden in Verilog oder auf Gate-Level beschrieben. [FIT97], [FIT98], [REV97], [PAT97]

Die Kopplung zwischen Testprogramm und Simulator erfolgt zwischen Debugger und Cadence Design Framework, in dem auch das DIB graphisch (als Schematic) entwickelt wird. Von dort aus besteht eine Kopplung mit dem Simulator SABER® (Abbildung 3.6). Der Cadence Frameway ist eine Option, die im SABER® geliefert werden kann.

#### 3.1.2.8 VX (Teradyne)

Teradyne ist einer der führenden Anbieter von industriellen Testsystemen. Die Produktpalette umfaßt SOC- und Mixed-Signal Testmaschinen, Memory-Tester und das speziell für Microcontroller ausgelegte sehr kompakte INTEGRA Testsystem, das unter Windwos NT läuft und über Microsoft Excel und Visual Basic programmiert wird.

Die Entwicklungsumgebung für Testprogramme heißt IMAGE und wird auf den SOC und Mixed-Signal Testmaschinen Catalyst, Tiger und der A5-Familie verwendet.

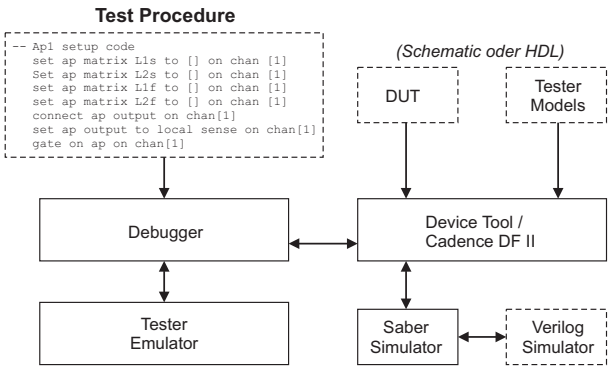



**Abbildung 3.6: LTX-Saber Virtual Test [PAT97]**

VX integriert Teradynes IMAGE zusammen mit einem Simulator zu einer interaktiven geschlossenen Umgebung zur Simulation von Testprogrammen. VX bildet dabei die Tester der A5-Familie sowie die Catalyst nach, wobei eventgetriebene Kommunikation zwischen IMAGE und dem Simulator unterstützt wird. IMAGE steuert die Simulation, indem das Testprogramm das Verhalten jedes Instruments beeinflußt. Das simulierte Testsystem verhält sich wie ein realer Tester, dadurch kann der Testingenieur seine gewohnten Techniken und Funktionen innerhalb von IMAGE nutzen und anwenden.

Dazu kann IMAGE mit verschiedenen Simulatoren gekoppelt werden. Diese Kopplung erfolgt über IMAGE ExChange (Abbildung 3.7). Eine nähere Erklärung erfolgt in Abschnitt 3.2.4.

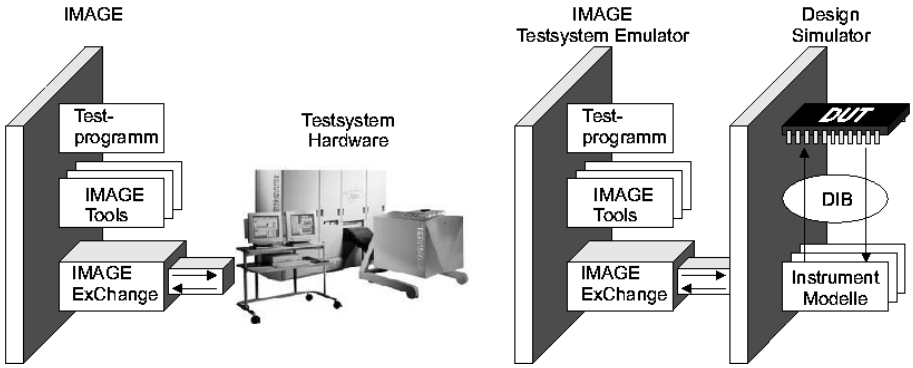



**Abbildung 3.7: Konzept of IMAGE ExChange**

Um rein digitale Simulationen auszuführen werden von Teradyne DigitalVX sowie die DigitalVX-ATE-Instrumenten-Modelle benötigt. Mit diesem Paket kann entweder ModelSim, der Verilog und VHDL verarbeiten kann (von Model Technology), oder der Digital-Simulator Verilog-XL inklusive des Schematic Entry und der Bibliothek DFII Design framework (von Cadence) kombiniert werden.

Für die Mixed-Signal Simulation stehen ebenfalls zwei Varianten zur Auswahl. IMAGE kann

mit Hilfe von VX für Saber und den entsprechenden VX-Saber-ATE-Instrumenten-Modellen mit dem Simulator Saber von Analogy betrieben werden. Zusätzlich wird noch Saber-IC Pro, das den Saber Analog Simulator von Analogy enthält sowie der Verilog-VX Digital-Simulator und das DFII Design framework (Schematic und Bibliothek) von Cadence benötigt.
Um IMAGE mit Spectre zu betreiben, sind wiederum die SpectreVX Simulations-Software sowie die SpectreVX-ATE-Instrumentenmodelle von Teradyne nötig sowie der Spectre Analog-Simulator, der Verilog-XL Digital-Simulator und das DFII Design Framework (Schematic und Bibliothek) von Cadence.

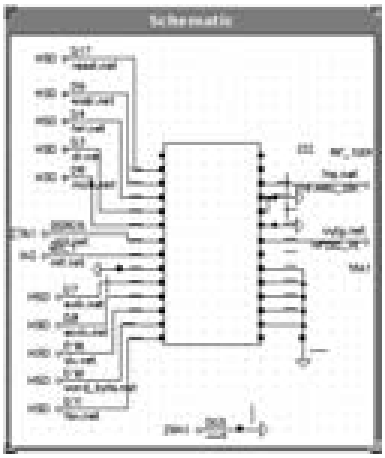


**Abbildung 3.8: Schematic Entry Tool**

VX bindet EDA-Werkzeuge in IMAGE ein und macht sie so dem Testingenieur zugänglich:

- Vereinfachtes Schematic-Entry-Tool unterstützt die Entwicklung von DIBs. (Abbildung 3.8)
- Optimierte ATE Instrumenten Modelle reduzieren die Simulationszeit durch intelligente Partitionierung der Modellteile zwischen den Simulatoren.
- Benutzerfreundliche Umgebung, welche die Standard-Entwicklungs- und Debugging-Umgebung widerspiegelt.
- Test-Ingenieure fühlen sich rasch vertraut in der VX-Simulations-Umgebung. Mit wachsender Erfahrung können sie die Vorteile der EDA-Tools zunehmend nutzen.

### 3.1.3 Produktvergleich / Diskussion

Ein direkter Vergleich mit einheitlichen Vergleichskriterien ist nur eingeschränkt möglich, da die einzelnen Produkte jeweils einen Nischenbereich im virtuellen Test einnehmen (z.B. Speicher-

test). Ebenso ist es schwierig, die vorgestellten Bewertungskriterien anhand von uneinheitlichen Demonstratoren gegenseitig zu gewichten.

#### 3.1.3.1 Bewertung von Dantes

Da Dantes auf dem Cadence Design System basiert und die Testschritte graphisch als Schematic eingegeben werden, erfolgt die Arbeit des Testingenieurs in einer für ihn vollkommen ungewohnten Umgebung. Dem Benutzer stehen ebenfalls keine seiner gewohnten Werkzeuge zum Debuggen zur Verfügung. Die Arbeit erfolgt simulationsbasierend ohne jeden Bezug zum Source-Code, der erst nach erfolgreicher Simulation generiert wird.

In der Evaluierungsversion von Dantes für die VISTA-VISION-Toolbox wies der generierte Source-Code so große Mängel auf, daß es nicht möglich gewesen ist, das Testprogramm auf dem Testsystem zu starten, um die Simulationsergebnisse mit realen Meßwerten zu verifizieren.

Bedingt durch die Vorgehensweise, jeden Testschritt separat zu behandeln, entsteht ein Testprogramm, das nicht zeitoptimiert ist. Sämtliche Einstellungen basieren auf den Grundeinstellungen der Instrumente und nicht auf Instrumenten-Setups vorhergehender Testschritte. Des weiteren kann bei ungünstiger Vorgehensweise ein Gesamt-Schematic und damit ein Loadboard entstehen, das sämtliche Testerressourcen über Relais verschaltet zum DUT führt. Dies kann sehr leicht passieren, denn bei jedem einzelnen Testschritt wird das Schematic von Grund auf neu gezeichnet, das heißt es stehen sämtliche Instrumente immer unverschaltet zur Verfügung. Erst aus diesen einzelnen Schematics wird am Ende ein Gesamtschaltplan entworfen. Treten Mehrfachbelegungen der Ressourcen zwischen zwei Testschritten bei der Beschaltung auf, versucht Dantes diese durch Einsetzen von Relais zu lösen.

Vorteilhaft ist, daß in Dantes eine Bauelemente-Bibliothek enthalten ist, die es erlaubt, Standardkomponenten auf dem Loadboard zu integrieren.

Getestet wurde die Umgebung am Modell eines einfachen RAMDAC. Dieser enthält einen 8-bit AD- und DA-Konverter. Das RAM wird über einen Bus adressiert und umfaßt ein 3 x 8-bit Feld. Im Rahmen der Evaluierung wurden folgende Standardtests programmiert. ADC, DAC, Address Read/Write Test, Setup Test und Hold Test.

#### 3.1.3.2 Bewertung des Digital Virtual Tester

Der Digital VirtualTester ist nur auf die Überprüfung digitaler Pattern und damit auf Verifikation der digitalen Module auf einem IC ausgelegt. Dadurch bietet das Produkt eine hohe Simulationsgeschwindigkeit. Allerdings können analoge Teile mit diesem Werkzeug nicht behandelt werden,

ebenso ist es auch nicht möglich Spannungspegel an den digitalen Pins zu überprüfen.
Der Testingenieur muß sich in eine Simulationsumgebung einarbeiten, die von der Handhabung nicht seiner gewohnten Bedienoberfläche am Tester entspricht. Der DVT testet nur die reine Patternfunktionalität. Sämtliche programmiertechnischen Vorgänge, die an einem Testsystem vorgenommen werden müssen, um ein Pattern ordnungsgemäß zum Laufen zu bringen (Pattern laden, Speicherverwaltung, Pinlisten erstellen, Spannungspegel festlegen, ...), bleiben unberücksichtigt.

#### 3.1.3.3 Bewertung des Virtual Test Emulator

Die vorhandene Umgebung der Testsysteme wird genauso unterstützt wie Werkzeuge der Designer, um so die ideale Voraussetzung für beide Anwender zu bieten und die Kommunikation zwischen ihnen zu erleichtern.

Der Virtual Test Emulator ist für Memory-Testsysteme bestimmt. Das heißt, dieses Produkt unterstützt nur digitale Simulation. Deshalb ist davon auszugehen, daß eine hohe Simulationsgeschwindigkeit erzielt wird. Allerdings werden Effekte unentdeckt bleiben, die durch fehlerhafte Spannungspegel hervorgerufen werden.

#### 3.1.3.4 Bewertung der Vorschläge von LTX, Motorola und Analogy

Eine Kopplung an einen Testsystem-Emulator (siehe Abschnitt 3.3.1) kann bewirken, daß viele emulatorspezifische Daten an den Simulator weitergegeben werden, die dieser nicht notwendigerweise braucht. Deshalb wird die Kopplung ohne Emulator positiv gesehen. Auf diese Weise wird die Kommunikation zwischen Simulator und Testprogramm nicht mit unnötigen Daten belastet.

Nachteilig ist der Umweg in der Kommunikation über den Cadence Frameway, der eine zusätzliche Schnittstelle und damit einen Engpaß darstellt. Es ist zwar von Vorteil, daß für das Loadboard eine graphische Eingabe zur Verfügung steht, doch hätte diese in Saber-Sketch ebenso verwirklicht werden können. Saber-Sketch ist ein Schematic-Entry-Tool, das sich im Lieferumfang des Simulators Saber befindet.

Auffällig waren in den Screen-Shots und den Vorführungen dieses Produkts „Instrumente“, die zwar auf dem Loadboard-Schematic plaziert werden mußten, die jedoch nur zu Simulationszwecken dienen. Diese „Instrumente“ sollten unsichtbar im Hintergrund gehalten werden, da sie einen simulationsunerfahrenen Anwender verwirren können.

#### 3.1.3.5 Bewertung von VX (Teradyne)

Die Verwendung der gewohnten Testprogramm-Entwicklungsumgebung IMAGE mit allen bekannten Debugging-Funktionen erleichtert die Einarbeitung der Testingenieure in den Bereich der Simulation.

Ein einfaches Schematic Entry ermöglicht eine schnelle Eingabe der Verbindungen zwischen DUT und ATE auf dem DIB.

Die Anbindung an verschiedene Simulatoren ist positiv zu bewerten. Allerdings ist, bis auf die Digital-Lösung mit ModelSim, immer der Verilog-XL von Cadence inklusive des DFII Design-Frameworks zu integrieren. Der Zugriff auf reine Digital-Simulatoren bedeutet eine hohe Simulationsgeschwindigkeit. Allerdings ist im Bereich der Mixed-Signal Simulation durch die Kombination zweier Simulatoren im Rahmen einer Co-Simulation (z.B. Saber mit Verilog-XL) ein Performance-Verlust zu erwarten.

#### 3.1.3.6 Zusammenfassung der Bewertung

Abschließend soll für die vorgestellten Produkte eine Gegenüberstellung der Vor- und Nachteile gegeben werden (Tabelle 3.1).

## 3.2 Publizierte Ansätze zur Modellbildung

### 3.2.1 Noise und Jitter im virtuellen Test [HEL96-1] [HEL96-2]

Die Leistungsfähigkeit der virtuellen Testsysteme befriedigt noch nicht alle Wünsche der Anwender [HEL96-1]. Zum Beispiel berücksichtigen die derzeitigen Lösungen nicht den Einfluß von statistischen Effekten wie Noise oder Jitter. Dazu muß die Frage nach der passenden Modellierung gelöst werden. Für rein digitale Bauelemente scheint zunächst eine klassische ereignisgesteuerte Digital-Simulation ausreichend zu sein. Natürlich ist es damit nicht möglich, Kontakt- oder Parametertests zu simulieren, die nach Strom- oder Spannungsmessungen verlangen. Eine rein digitale Simulation kann ebenso wenig Effekte der Signalübertragung auf dem Loadboard berechnen. So ist es denkbar, daß ein digitaler Pin des Testers sowohl als digitale Quelle als auch als Quelle für eine Referenzspannung oder -strom fungiert. Somit wird eine analoge und digitale Modellierung des Pins notwendig. [HEL96-1]

| **Produkt** | **Vorteile** | **Nachteile** |
|---|---|---|
| Dantes | Schematic Entry | Keine Test-GUI und Debugging-Werkzeuge |
| | Eigenständig lauffähige Testschritte durch eigenständigen Aufbau | Kein zeitoptimiertes Testprogramm |
| | Automatisch generierter Sourcecode | Sourcecode nicht lauffähig auf realer Testmaschine |
| | Bauelementebibliothek für Loadboard | Suboptimales Loadboard durch Teststepbezug |
| DVT | Schnelle Simulation | Nur digitale Pattern, keine Spannungspegel |
| | Analyse mit beliebigen Tools | Simulationsausführung über Java-GUI |
| | Analyse von Patterdaten für viele Maschinen | Keine Analyse des Sourcecodes |
| VTE | Verwendung der gewohnten ATE-GUI und von Designwerkzeugen | Nur für Memory Test |
| | Schnelle Simulation | Nur digitale Pattern, keine Spannungspegel |
| LTX | Verwendung des ATE-GUI | |
| | Schematic Entry für DIB | Zusätzliche Integration des Cadence Frameway |
| | Kopplung ohne Emulator | Zusätzliche Instrumente auf dem Loadboard |
| VX | Verwendung des ATE-GUI | Kopplung mit Emulator |
| | Schnelle Simulation im Digitalen | Langsame Simulation im mixed-signal durch Co-Simulation |
| | Schematic Entry für DIB | Einbindung nur über DFII Design Frameway |
| | Ankopplung von verschiedenen Simulatoren möglich | |

**Tabelle 3.1: Bewertung der kommerziellen virtuellen Test Umgebungen**

Aufwendige Synchronisationsmechanismen stellen das Rückgrat moderner Testsysteme dar. Diese Synchronisationsmechanismen stellen die Einhaltung eines Pakets von Zeitbedingungen, die nicht ein einzelnes Instrument betreffen, sondern die Funktion von zwei oder mehreren Instrumenten kombinieren. Deshalb können Testerinstrumente nicht als eigenständige Einheiten modelliert werden, die „nur“ auf einen Simulationstakt hören. Vielmehr ist eine exakte Modellierung dieser Trigger und Synchronisationsmöglichkeiten notwendig. [HEL96-2]

Zur Verdeutlichung der speziellen Modellierungsanforderungen beim virtuellen Test werden die Antwort-Signale der Bauelemente herangezogen, die gerade im Mixed-Signal Bereich eine große Rolle spielen. Hier werden die Baustein-Antworten häufig mit DSP-Routinen bearbeitet, um die Bauelement-Parameter zu extrahieren. Diese Fähigkeiten und Einschränkungen der DSP-Software muß die virtuelle Testumgebung unterstützen. Die Ergebnisse einer DSP-Untersuchung können Spektren, Histogramme oder einzelne Werte (z. B. SNR) sein. Diese Werte müssen in der Simulation nicht nur berechnet werden, sie müssen auch mit den Ergebnissen am realen Testsystem übereinstimmen. Der Ansatz, die Signale des DUT mit der dafür erforderlichen Genauigkeit zu simulieren, erscheint zunächst übertrieben aufwendig, ist aber sinnvoll, wenn man den Untersuchungskriterien der zu messenden Daten betrachtet. [HEL96-1]

Bei der Verwendung von fehlerfreien DUT-Modellen wird immer die optimale Charakteristik des Bauelements gemessen. Gerade bei Tests, die DSP-Routinen verwenden, ist das wichtig. Mit fehlerbehafteten Modellen weiß der Testingenieur nur, daß ein Fehler vorliegt und er muß - wie bisher - zeitaufwendig feststellen, ob der Fehler nicht doch im Bereich des DUT liegt. Somit wird die Analyse und das Debugging mit fehlerfreien Modellen klarer.
Ein modelliertes Fehlverhalten soll die Teile des Testprogramms überprüfen, die exakt diesen Fehler erkennen sollen. Auf diese Weise können Erkenntnisse gewonnen werden, ob das Testprogramm das Fehlverhalten mit ausreichender Genauigkeit erfaßt, was wieder bei der Anwendung von DSP-Testschritten eine entscheidende Rolle spielt. Somit verbessert der virtuelle Test die Qualität der Testprogramme, dies führt zu einer höheren Ausbeute.
Die bisher besprochenen Modelle sind deterministisch. Bei realen Messungen haben jedoch stochastische Effekte wie Noise und Jitter einen nicht zu unterschätzenden Einfluß.
Für das Rauschen in elektrischen Systemen kann in der Regel eine Gaußverteilung angenommen werden. Die Verteilung ist definiert durch den Mittelwert und die Standardabweichung. Wenn Rauschen als Abweichung vom erwarteten Signal definiert wird (der Mittelwert der Rauschspannung ist Null) bleibt die Standardabweichung $sn$ der charakteristische Parameter. Um Rauschen in die Verhaltensmodelle von DUT, DIB oder ATE zu implementieren, müssen Zufallszahlen, die eine Gaußverteilung mit dem Mittelwert Null und der Standardverteilung $sn$ haben, zum deterministischen Signal addiert werden.
Jitter wird verursacht von verschiedenen voneinander abhängigen statistischen Prozessen. Die Wahrscheinlichkeitsverteilung $pj(t)$ für die resultierende Abweichung $t$ vom erwarteten Zeitpunkt

des Ereignisses zeigt ebenso Gaußsche Charakteristik mit der Standardabweichung, dem Jitterparameter *tj* (RMS Jitter). Um Jitter zu simulieren ist es notwendig, die Triggereingänge von DUT und Testerinstrumenten (inklusive der Synchronisationsmechanismen) derartig zu modifizieren, daß eine statistische Zeitunsicherheit $t$ eine Gaußverteilung um Null mit der Standardabweichung *tj* zeigt. Der Parameter *tj* ist spezifisch für jedes Modell. Um zu deterministischen Modellen zurückzukehren, muß er abschaltbar sein. Rauschen wurde mit einem ähnlichen Mechanismus realisiert. An- und Abschalten sowie Veränderung der Noise- und Jitter-Parameter in den einzelnen Modellen ermöglicht dem Benutzer eine Feinabstimmung seiner Meßprozeduren. [HEL96-2]

Die Simulationszeit lag bei wenigen Minuten für einen kompletten Parametertest. [HEL96-2]

Der Standard-Ansatz von Virtual Test benutzt fehlerfreie idealisierte deterministische Modelle. Als Erweiterung dazu bietet die Verwendung von DUT-Modellen mit „bekanntem Fehler“ die Möglichkeit, das Testprogramm auf die korrekte Erfassung der fehlerhaften Parameter hin zu überprüfen. In einem weiteren Schritt können nicht deterministische DUT- und ATE-Modelle reale physikalische Eigenschaften wie Rauschen und Jitter besser nachbilden. Die in der Veröffentlichung vorgestellten Ergebnisse zeigen, daß die Verfahren praktisch anwendbar sind und somit die Leistungsfähigkeit der virtuellen Testsysteme steigern. [HEL96-2]

### 3.2.2 Testsystemsimulation mit hSpice

In einem weiteren Ansatz wurde ein Verifikationstestsystem in SPICE modelliert [BEL99]. Bei dem Testsystem handelt es sich um IntegraTest, das von Microlex Systems und IMS entwickelt wurde. Es ist ein flexibles Testsystem, das sich aus Standard GPIB- und VXI-Komponenten aufbaut (Abbildung 3.9). Damit ist dieses System ein applikationsspezifisches Testsystem, das Kosten minimiert, indem es sich nur aus Unterkomponenten zusammensetzt, die der Kunde benötigt und auf alle für die aktuelle Anwendung überflüssigen Testressourcen verzichtet. Beim Controller kann zwischen einer SUN Workstation mit UNIX und einem PC mit Windows gewählt werden. Unter beiden Systemen stehen verschiedene Software-Tools zur Verfügung, die mit Hilfe von LabView programmiert werden, das unabhängig von der gewählten Plattform ist. [MIC00]

Bei der Modellierung des Systems wurden folgende Komponenten berücksichtigt. HPE1445 Waveform Generator, HP1428 und B&K 3005 Digitizer, HPE1465 Matrix Modul und HPE1472 RF Multiplexer Modul. Counter und Multimeter wurden nicht modelliert, aufgrund der komplexen DSP-Routinen, die in den Meßinstrumenten implementiert sind.
Die Aufgaben für die einzelnen Instrumente werden im Simulationsfile mit Hilfe von Parametern festgelegt (Abbildung 3.10). Dieses Simulationsfile soll in einer späteren Ausbaustufe einfach mit Hilfe von Software generiert werden. Somit muß es möglich sein, die Modelle in Cadence Framework zu integrieren.

**Abbildung 3.9: Aufbau des Testsystems IntegraTEST**

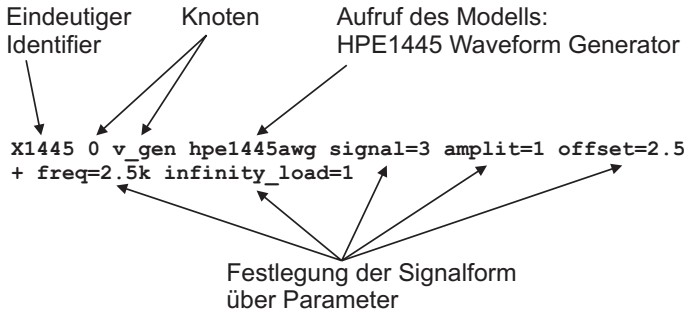



**Abbildung 3.10: Ausschnitt aus einem SPICE Simulationsfile**

Der vorgestellte Ansatz soll offensichtlich nicht das Testprogramm selbst verifizieren, sondern nur das Testkonzept. Die Auswertung der angelegten Signale sowie der Bausteinantworten erfolgt anhand des von SPICE angelegten Logfiles.

Mit den verwendeten, sehr einfachen Modellen konnte in der Simulation nachgewiesen werden, daß die Ergebnisse nahezu immer den Meßwerten in der Realität entsprachen. Aufgetretene Unterschiede konnten durch Ungenauigkeiten und Vereinfachungen in der Modellierung erklärt werden.

### 3.2.3 Wiederverwendbarkeit von DUT-Systemmodellen

Der effektive Einsatz von Virtual Test ist stark abhängig vom Aufwand der notwendig ist, um DUT- und DIB-Modelle zu schreiben sowie von der erforderlichen Rechengeschwindigkeit und Genauigkeit der verwendeten Modelle [EIN98].

In der Veröffentlichung [EIN99] setzt sich das DUT-Modell aus zwei Komponenten zusammen, bei dem DSP- und Kontrolleinheiten-Design sowie die Gesamtsystem-Verifikation wiederverwendet wurden. Im COSSAP-Teil, einem System-Level-Simulator, sind DSP-Algorithmen, Hardware-Filter und analoge Komponenten realisiert. Im Leapfrog-Teil, einem VHDL-Digital-Simulator, sind die Kontroll-Funktionen eingebaut. Diese Kombination führt in Bezug auf die Funktionalität zu sehr detaillierten und im Vergleich zu Modellen auf Schaltkreis-Ebene zu sehr schnellen Modellen.

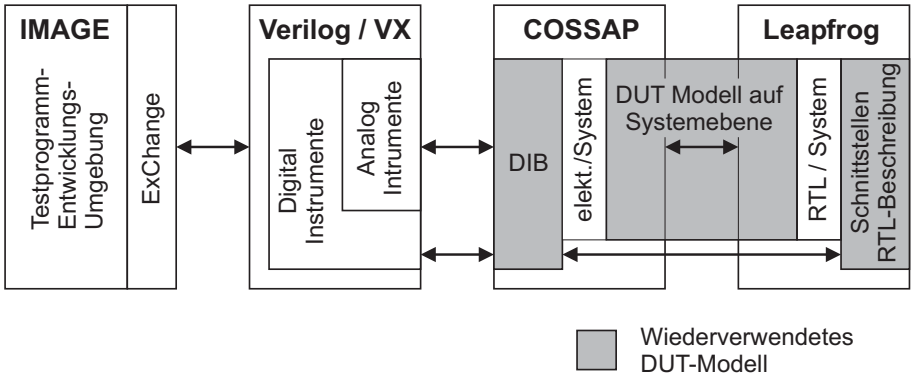



**Abbildung 3.11: Simulationsumgebung für wiederverwendete DUT-Modelle**

Die komplette Umgebung, welche die Wiederverwendung von DUT-Modellen unterstützt, umfaßt die Komponenten IMAGE/ExChange, Verilog (mit der VX-Software), COSSAP und Leapfrog, die über eine IPC (Inter process communication) miteinander gekoppelt sind (Abbildung 3.11) [EIN00].

### 3.2.4 Benutzerdefinierte Virtual Test Umgebung

Wie in Kapitel 1.1.1 beschrieben, stehen dem Designteam verschiedene spezialisierte Anwendungen zur Verfügung, um die Funktionalität auf Schaltkreisebene zu definieren. Je nach Produktpalette des Chip-Herstellers können sich die eingesetzten Design-Werkzeuge und Simulatoren und damit der Entwicklungsfluß ganz erheblich unterscheiden.
Teradyne bietet zwar in seiner virtuellen Testumgebung verschiedene Simulatoren an (siehe Abschnitt 3.1.2.8), doch kann ein Testerhersteller nicht den gesamten Simulatormarkt mit Modellen abdecken. Deshalb ist die Untersuchung der Möglichkeit und des Aufwands einer Anpassung virtueller Testumgebungen auf den benutzerspezifischen Designfluß wichtig, um Virtual Test in den gesamten Entwicklungsfluß zu integrieren. In der Veröffentlichung [RIO99] wird beschrieben, wie die Virtual Testumgebung Image/ExChange an benutzerspezifische Vorgaben angepaßt wird.

Bei Analog Devices ist die Simulations- und Design-Umgebung ADICE seit über acht Jahren im Einsatz. ADICE bietet eine Simulationsumgebung an, mit der es möglich ist, Analog-, Digital-

oder Mixed-Signal-Schaltungen vom Verhaltensmodell bis hin zur Transistorebene zu simulieren. IMAGE/Exchange stellt die Schnittstelle zwischen der Testprogramm-Entwicklungsumgebung und Simulatoren dar, indem dieser Programmteil zeitlich feststehende Ereignisse generiert, die die Zustandsänderungen in der Testmaschine aufgrund des ausgeführten Testprogramms beschreiben. Um diese Daten in eine simulatorspezifische Form zu bringen, ist ein Interface-Programm notwendig (Abbildung 3.12). Dieses Programm muß den von ExChange ausgehenden Datenstrom analysieren und manipulieren können. Dazu kann auf C-Funktionen zurückgegriffen werden, welche die ExChange Interface Bibliothek zur Verfügung stellt. Auf diese Weise werden den Modellen Datenstrukturen mit den Parametern übergeben, die das Instrumenten-Setup aufgrund des Testprogramms festlegen. Das gilt auch für digitale Subsysteme, bei denen jede Änderung im Pattern - ob ein digitaler Wert auf 0 oder 1 liegen soll - von IMAGE via ExChange übermittelt wird.

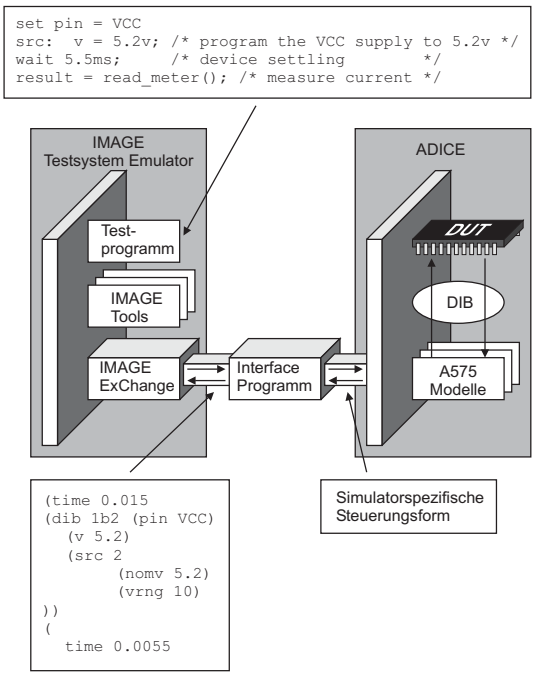



**Abbildung 3.12: IMAGE/ExChange in Kombination mit einem nichtunterstützten Simulator**

Teradyne hat mit dieser Schnittstelle ein System geschaffen, das es ermöglicht, einen eigenen Simulator an IMAGE anzudocken, um so Virtual Test in den hauseigenen Design-Flow zu integrieren.

## 3.3 Kopplungsmechanismen

Im Folgenden sollen zwei Möglichkeiten der Kopplung zwischen Testprogramm und Simulator erklärt werden.

### 3.3.1 Emulator-Docking

**Definition: „Testsystem-Emulator“**
Der Emulator eines Testsystems bildet in Software jeden Zustand des Testers nach. Das ermöglicht zunächst einen Konsistenz-Check, ob alle im Testprogramm verwendeten Instrumente auch als Hardware im Testsystem vorhanden sind. Des weiteren erlaubt es eine erste Überprüfung der Syntax auf Fehler, ohne einen Simulator oder reale Testsystem-Hardware heranzuziehen.

Über den Emulator wird der aktuelle Zustand des Testsystems an den Virtuellen Tester (Simulator) gesendet (Abbildung 3.13). Das heißt, bei jedem abgearbeiteten Befehl wird eine große Datenmenge an den Simulator geschickt, die die Zustandsänderungen der Testmaschine enthält.

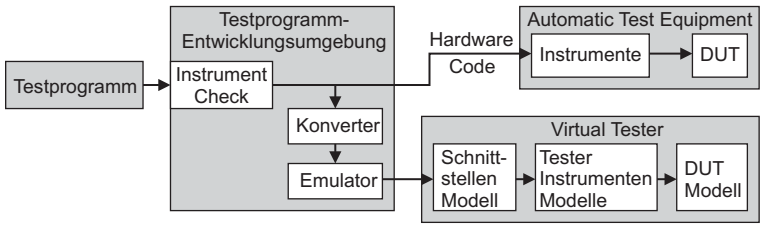



**Abbildung 3.13: Emulator Docking**

### 3.3.2 Testprogramm-Docking

Beim Testprogramm-Docking werden die Befehle des Testprogramms syntaktisch überprüft, bevor sie für den Simulator aufbereitet werden, indem sie in ein Protokollformat verpackt werden (Abbildung 3.14). Auf der Simulatorseite werden die eingehenden Befehle aufbereitet und in einer Datenstruktur abgelegt, die der Simulator verarbeiten kann. Auf diese Weise wird nur ein Minimum an Daten ausgetauscht.

## 3.4 Zusammenfassung der Analyse und resultierende Ziele

Die Untersuchung verfügbarer Virtual-Test-Werkzeuge ergab, daß alle Produkte nur eine kleine Nische im Entwicklungsfluß besetzen. Ein allgemeiner Ansatz fehlt, der auf die gesamte Aufgabe

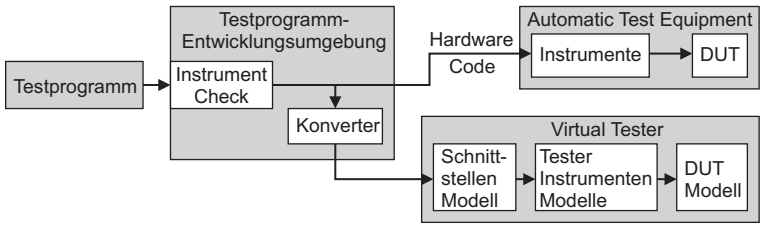



**Abbildung 3.14: Direktes Testprogramm-Docking**

des Testingenieurs abzielt. Beispielsweise verwenden die Produkte zum Teil eigene Benutzeroberflächen, die sich nicht in die Testprogramm-Entwicklungsumgebungen der Testsysteme integrieren lassen. Nicht alle Produkte erlauben es Spannungspegel der Signale zu verifizieren oder lassen den Sourcecode der Testmaschine komplett außer acht.

Auch bei den Kopplungsmechanismen für nebenläufige Prozesse waren Schwachstellen erkennbar. So werden bei dem Verfahren des Emulator-Dockings unnötig große Datenmengen über die Schnittstelle und damit den Engpaß des Systems versendet.

Eine Virtual-Test-Umgebung, die es ermöglicht, auf verschiedenen Abstraktionsebenen zu simulieren, ist derzeit am Markt noch nicht verfügbar. Die vorgestellten Produkte setzen zwar auf verschiedenen Abstraktionsebenen auf und nutzen die Vorteile dieser Ebene, können jedoch nicht zu einer anderen Ebene umschalten.

Die in der Arbeit geschaffenen virtuelle Testumgebung (siehe Kapitel 4) bietet die Möglichkeit der „Prüfprogrammsimulation“ auf verschiedenen Abstraktionsebenen. Sie bietet Schnittstellen für die Optimierung von Testprogrammen [BEY00]. Dazu wurde ein Verfahren zum netzwerkweiten Datenaustausch entwickelt, der nebenläufige Prozesse zeitgenau synchronisiert und alle möglichen Parameterübergaben berücksichtigt, die bei der Steuerung von Testmaschinen eine Rolle spielen. Am Beispiel eines digitalen Pins werden Lösungswege gezeigt, wie in Simulationsmodellen mit großen Datenmengen, die durch Testmuster und Kombinatorik hervorgerufen werden, zu verfahren ist.

Die eingehende Untersuchung momentan für Testsysteme verfügbarer Hard- und Software-Komponenten (siehe Kapitel 2) hat gezeigt, daß im Zuge der historischen Entwicklung und der dabei durchgeführten Verbesserungen aufgrund der Forderungen nach mehr

- Flexibilität
- Ressourcen
- Parametrisierung

- Parallelisierbarkeit

eine komplexere Hardware resultierte.
Dies führte jedoch zu

- erhöhten Fehlerquoten in der Treiberstruktur
- aufwendigerer und erschwerter Low-Level-Programmierung der Hardware
- schwierigerer Modellierung
- längeren Simulationszeiten

Um dem Ziel, schnell und effektiv Testprogramme zu schreiben [ITR03], näherzukommen, wäre eine system-unabhängige High-Level-Sprache notwendig, die momentan aber nicht mit ausreichender Ausdrucksfähigkeit zur Verfügung steht, um die vorhandene Hardware geeignet anzusprechen. Dies ist aus heutiger Sicht nur durch das Entfeinern der Hardware mit gleichzeitiger Unterstützung von Modularität möglich. Dadurch wird die Hardware ebenfalls leichter modellierbar, die Simulationszeiten beim virtuellen Test werden sinken und die Programmierung der Instrumente vereinfacht.

Um die aufgestellten Forderungen an die Testsystem-Ressourcen zu erfüllen, muß das Zusammenspiel verschiedener Komponenten innerhalb einer Architektur verifiziert und verschiedene Architekturkonzepte miteinander verglichen werden. Das in der Arbeit vorgestellte Konzept (siehe Kapitel 5) unterstützt die Entwicklung und Bewertung neuer Testsystemarchitekturen.

# Kapitel 4

# Implementierung eines virtuellen Testsystems

In diesem Kapitel wird das realisierte Konzept einer virtuellen Testumgebung vorgestellt. Zunächst werden die Komponenten beschrieben, die aufgrund von Kundenwünschen oder äußeren Vorgaben im Konzept und der Implementierung zu berücksichtigen sind. Bei der Vorstellung der im Rahmen dieser Arbeit entstandenen virtuellen Testumgebung wird von Anforderungen an virtuelle Testsysteme ausgegangen, die zusammen mit Industriepartnern erarbeitet wurden. Anschließend wird der Lösungsweg zur Umsetzung jeder einzelnen Anforderung skizziert, der im Bereich virtueller Test implementiert wurde. Beispiele mit detaillierter Beschreibung der Implementierung finden sich im Anhang B. Bei der Modellierung von Testsystemen für den virtuellen Test ist zu beachten, daß der Fokus auf den Signalen liegt, die zwischen ATE und DUT anliegen. Vorgänge, die innerhalb der Testmaschine ablaufen, interessieren nicht und können deshalb abstrahiert modelliert werden. Dies soll in den nachfolgenden Abschnitten dargestellt werden.

## 4.1 Vorhandene Komponenten

### 4.1.1 SPACE

Im Rahmen des Projektes VIRTUS war die Vorgabe, eine virtuelle Testumgebung für SZ-Testsysteme aufzubauen. Dazu war in die Simulationsumgebung das Software-Paket SPACE (SZ Programming & ATE Controlling Environment) zu integrieren.


**Abbildung 4.1: SPACE**

Hierunter fällt die ganze Software-Architektur wie in Abschnitt 2.2 beschrieben. Für den Anwender ist vordergründig nur die Oberfläche sichtbar, mit der er das Testsystem bedient (Abbildung 4.1). Von hier aus werden Testprogramme geladen und gestartet, Ergebnisse eingesehen (Abbildung 4.2), Waveforms analysiert, Debugger gestartet und alle Funktionen ausgeführt, die während der Entwicklung und des Debuggings eines Testprogramms notwendig sind und die mit der Steuerung des ATE zu tun haben. Über diese Funktionen und Unterprogrammteile wird der Tester angesprochen und gesteuert. Hinzu kommen Werkzeuge und Programmpakete, die nicht direkt zum Betriebssystem der Testmaschine gehören, die aber zur Arbeit notwendig sind. Dazu gehört beispielsweise das Programmpaket WaveMaker, mit dessen Hilfe Pattern analysiert und editiert werden können (siehe Anhang A.3.1). Diesem Programmpaket liegen ferner Konverter bei, die es ermöglichen, verschiedene Patternformate ineinander überzuführen. Da diese Bedienoberfläche eine derart zentrale Schaltposition einnimmt, soll diese Umgebung ebenso beim virtuellen Test Einsatz finden, damit der Testingenieur in seiner gewohnten Arbeitsumgebung weiterarbeiten kann.

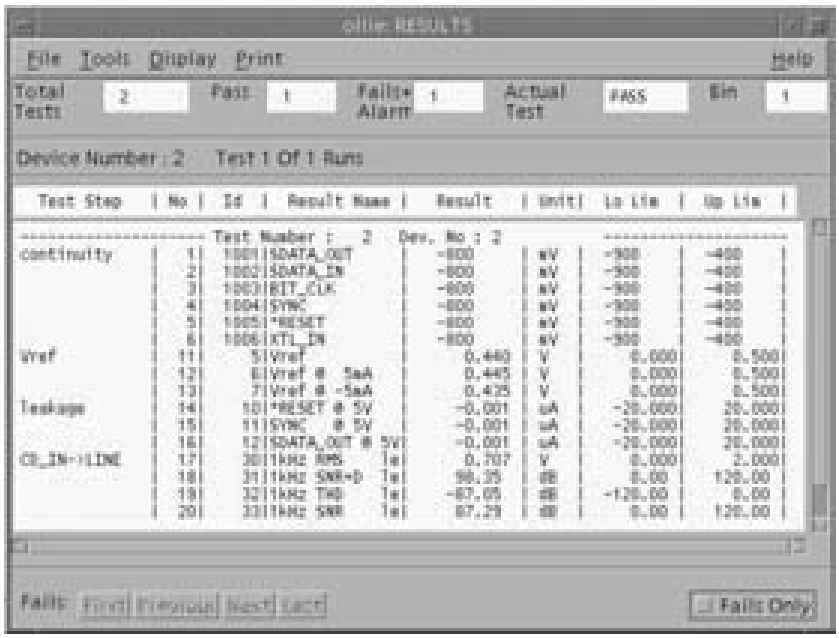



**Abbildung 4.2: Result-Display**

### 4.1.2 Simulator

Ein Simulator für die virtuelle Testumgebung wurde in der Arbeit „Zur Simulation von Prüfprogrammen für Integrierte Schaltungen“ [GOL98-1] ausgewählt. Der Entscheidung lag eine Untersuchung hinsichtlich der Modellierungssprache, der Schnittstellen sowie der Simulationsperformance zu Grunde. Die Wahl fiel auf den Simulator SABER® der Firma Synopsys (früher Avant!, davor Analogy) mit der Hardwarebeschreibungssprache MAST®.

SABER® ist ein Mixed-Signal-, Mixed-Level-Simulator, der unter anderem im Automobilbereich Einsatz findet. Seine Stärken liegen in der Systemsimulation. Komplette Anlagen und Systeme können von der ersten groben Planung bis in Details modelliert und simuliert werden. Dabei stehen nicht nur Modelle und Beschreibungsmöglichkeiten aus dem elektrischen Bereich zur Verfügung, auch mechanische, thermisch und hydraulische Systeme können in die Gesamtsimulation integriert werden. Die Verfügbarkeit der verschiedenen Modellierungsebenen während des Systementwurfs nennt man Multi-Level-Modellierung (Abbildung 4.3).

Eine systematische Einteilung dieser Ebenen bei der Simulation wurde in [KLU03] untersucht. Hierbei wurden zur Klassifizierung der Modellierungsansätze domänenübergreifende Modellklassen definiert. Dabei wurde zum einen der Bezug zu etablierten Modellierungskonzepten hergestellt, zum anderen wurden Unterschiede im Zeitmodell sowie die Mächtigkeit der Beschreibungskonzepte formal deklariert. Es ergaben sich Modellklassen, die den vorgestellten Leveln im

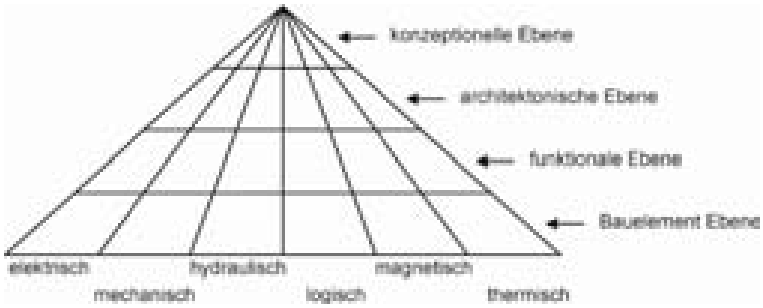



**Abbildung 4.3: Multi-Level-Modellierung**

Einsatz ähnlich sind und die sich an den sich ändernden Anforderungen während der Entwicklung komplexer Systeme orientieren.

Ebenso kann SABER® die verschiedenen Domänen „Analog“ und „Digital“ miteinander verknüpfen und in einer Simulationsumgebung mit einem Sprachumfang abdecken. Der sogenannte „Calaveras-Algorithmus“ steuert hierbei den Datenaustausch zwischen beiden Welten [SAB04]. Analoge und digitale Welt sind aber nicht nur im elektrischen Sinne zu verstehen. Vielmehr werden im Analogen zeit- und wertekontinuierlich beschreibbare Vorgänge eingeordnet. Im Gegensatz dazu fallen Zustandsmodelle in den Bereich der diskreten Welt. (Abbildung 4.4)

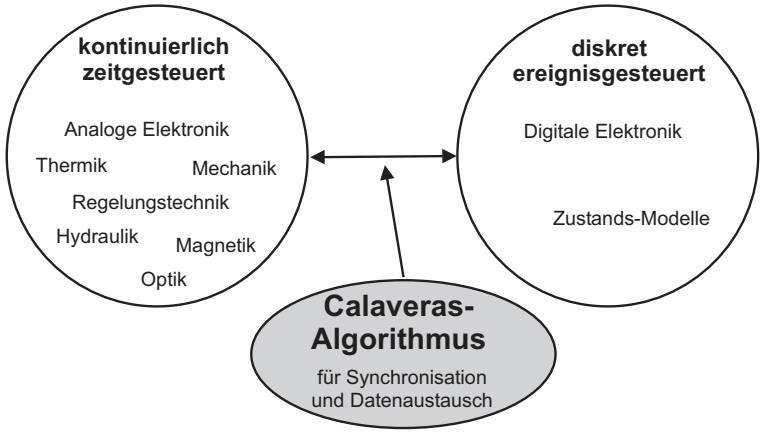



**Abbildung 4.4: Mixed-Signal Simulation**

### 4.1.3 Interface SPACE-Simulator

In der Arbeit [GOL98-1] wurde folgendes Basis-Konzept entwickelt, das in seiner Realisierung durch hinzufügen einer Bibliothek an das Testprogramm angebunden wurde. Darin befinden sich auch die Komponenten zum Aufbau einer Kommunikationsverbindung (IPC).

Auf der Simulatorseite wird ein eigenes Modell benötigt, das die Kommunikation aufbaut, die ankommenden Daten auswertet, aufbereitet und damit die Testsystemmodelle steuert. (Abbildung 4.5) Den schematischen Ablauf im Inneren der beiden Kommunikationsstrukturen beschreibt Abbildung 4.6. Diese Basiskomponenten zur Kommunikation standen somit implementiert zur Verfügung.

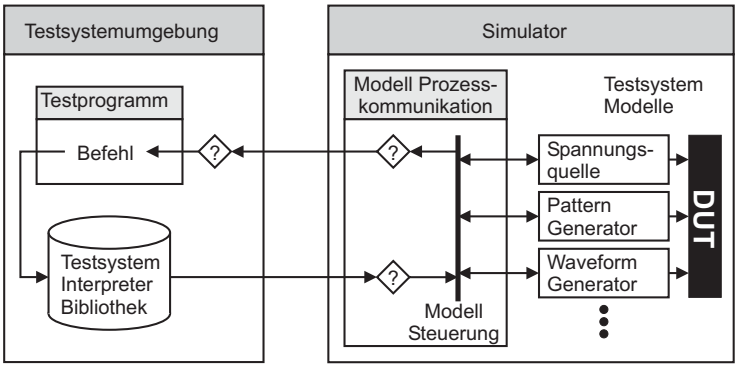



**Abbildung 4.5: Konzept eines Modell basierten Testprogramm-Simulator-Interfaces**

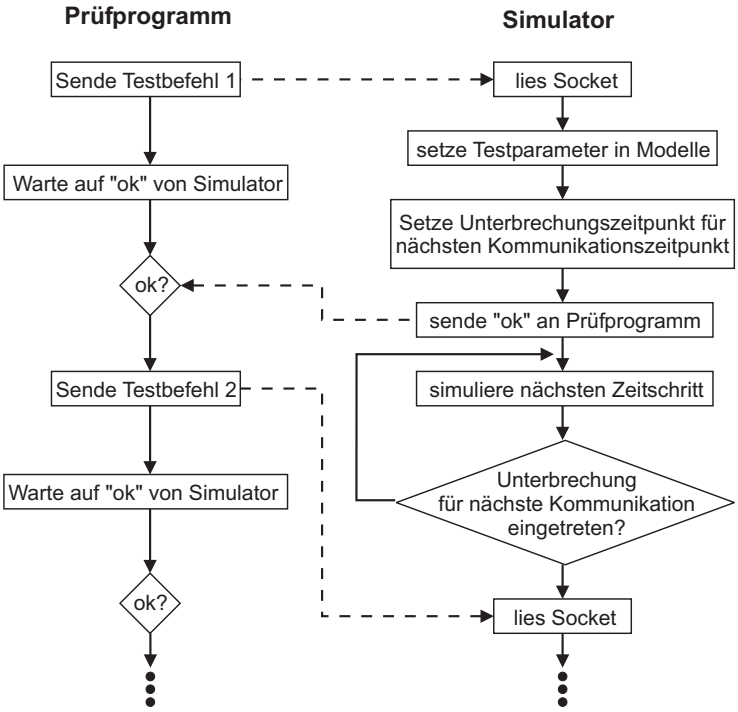



**Abbildung 4.6: Funktionsweise des Algorithmus zum Datenaustausch zwischen Prüfprogramm und Simulator**

## 4.2 Anforderung, Konzept und Realisierung des virtuellen Tests

Kapitel 3 beschreibt verschiedene Produkte aus dem Bereich „Virtual Test“. Die Akzeptanz dieser Werkzeuge hängt von der Erfüllung der Anforderungen ab, welche die Testingenieure an diese Produkte haben.

Im Rahmen vom virtuellen Test ist es notwendig, den Modellen eine Struktur, Hierarchie und ein Konzept zugrunde zu legen, das es ermöglicht, die Anforderungen der Praxis zu erfüllen und gleichzeitig eine Übersichtlichkeit in der Modellstruktur zu wahren.

Um eine in Benutzung und Ausführung effektive Simulationsumgebung zu schaffen, sind nicht nur die Wünsche der Testingenieure zu berücksichtigen, sondern auch Modellstrukturen zu planen, die eine zeiteffiziente Simulation eines deratig komplexen Systems ermöglichen. Solche Probleme zeigen sich schon bei der Simulation eines einzelnen Bausteins, der aufgrund seiner Komplexität in Baugruppen zerlegt wird. Die Schwierigkeiten potenzieren sich bei der Simulation eines kompletten Testsystems, das aus tausenden einzelnen Bausteinen besteht.

Je nach den Problemstellungen, die während der Entwicklung eines Testprogramms auftreten, will der Benutzer sehr unterschiedliche Informationen ermitteln. Dies kann die Genauigkeit der Signaldarstellung betreffen oder auch einen komplett anderen Blickwinkel auf eine Problemstellung bedeuten. Nachfolgend sind Beispiele für mögliche Untersuchungskriterien aufgezählt, die in der täglichen Arbeit des Testingenieurs auftreten:

- Konsistenz der Instrumenten-Parameter
- Zusammenspiel einzelner Testschritte (Setup, Setdown der Instrumente)
- Einfache Messung analoger Signale (z.B. Ruhestrom)
- DSP-Analyse analoger Signale
- Signalpfade im Testkopf korrekt zum DUT verschaltet
- Digitale Signale auf elektrischer Ebene (z.B. Ausgangstreiber)
- Hardwarenahe Effekte (z.B. Einschwingvorgänge verursacht auf dem DIB)

Diese sehr unterschiedlichen Anforderungen müssen in der Simulationsumgebung berücksichtigt werden

Die aus den Überlegungen der Abschnitte 2.1, 2.2 und des Kapitels 3 resultierenden Anforderungen, Konzepte und Lösungen sollen im Folgenden dargestellt werden.

### 4.2.1 Bedienungsumgebung

Die mäßige Akzeptanz existierender Werkzeuge in der Praxis hat gezeigt, daß eine der wichtigsten Vorgaben, die es zu erfüllen gilt, die Bedienungsumgebung ist. Sie sollte so beschaffen sein, daß sich der Testingenieur schnell und einfach zurechtfindet. Die Testingenieure stehen Simulationswerkzeugen mit einer gewissen Skepsis gegenüber. Da sie von der Ausbildung her keine Simulationsexperten sind, ist es notwendig, die Simulationsumgebung in die bekannte ATE-Software einzubetten. Dies hat so zu erfolgen, daß von der Bedienungs- und Arbeitsweise kein Unterschied zur realen Umgebung vorhanden ist, denn damit entfällt eine Einarbeitungszeit in die Simulationsumgebung.

Um das Erscheinungsbild sowie die gewünschte Arbeitsweise zu unterstützen, muß nicht nur die Software-Oberfläche und die dahinterliegende Software-Struktur des ATE-Betriebssystems verwendet werden, es ist vielmehr notwendig, auch alle Debugfunktionen einer realen Testmaschine zu unterstützen. Dabei muß es möglich sein, interaktiv in den Simulationsablauf einzugreifen, Haltepunkte zu setzen, Ergebnisse auf Wunsch abzufragen und dergleichen mehr.

Diese Forderungen an die Bedienung definieren die äußere Erscheinung und haben Einfluß auf die nachfolgenden Punkte.

### 4.2.2 Konfiguration der Simulation

#### 4.2.2.1 Anforderung

Die Steuerung der Simulation soll aus dem Testprogramm erfolgen, wobei die Benutzeroberfläche genauso wie an der realen Testmaschine erscheinen soll. Allerdings muß eine Simulation aufgesetzt werden, und dazu sind zusätzliche Einstellungen nötig, die nicht aus dem Testprogramm selbst ersichtlich sind. Es sind geeignete Mechanismen und Bedienungen vorzusehen, um u.a. folgende Daten an den Simulator zu übergeben:

- Auswahl der Simulationsart / Analyse
  (Transient, DC Operating Point, Kleinsignal, DC Transfer, Monte Carlo ...)
- Startzeit
- Simulationsdauer
- Schrittweite (falls nicht automatisch)
- Auswahl der dargestellten Signale
- Auswahl Simulations-Algorithmus

#### 4.2.2.2 Lösung: Simulationskonfigurator

Einige dieser Einstellungen sind aufgrund des virtuellen Tests logisch vorgegeben (Transienten Analyse mit Berechnung des DC Arbeitspunktes) und sollten automatisch an den Simulator weitergegeben werden, ohne daß sich der Benutzer darum kümmern muß. Andere Einstellungen (Schrittweite der Simulationszeitpunkte, Auswahl interner Signale, die zu berechnen und darzustellen sind ...) sind durch den Testingenieur vorzunehmen. Hierzu waren geeignete Formulare in die Benutzerumgebung des ATE-Betriebssystems zu integrieren. Eine Eingabe der erforderlichen Parameter über die Masken des Simulators würde unnötige Verwirrung und Unsicherheit stiften.

Während eines Testprogrammlaufs werden in der Regel nicht alle Instrumente eines Testsystems benötigt. Es beschleunigt die Simulation, wenn nur die vom Testprogramm verwendeten Baugruppen in das Gesamtmodell der Testmaschine eingebunden werden (Abbildung 4.7). Hierzu war eine Benutzerführung zu schaffen, die eine einfache Auswahl der benötigten Instrumente gestattet. In einem zweiten Schritt muß die Konsistenz der ausgewählten Komponenten mit dem Testprogramm überprüft werden.

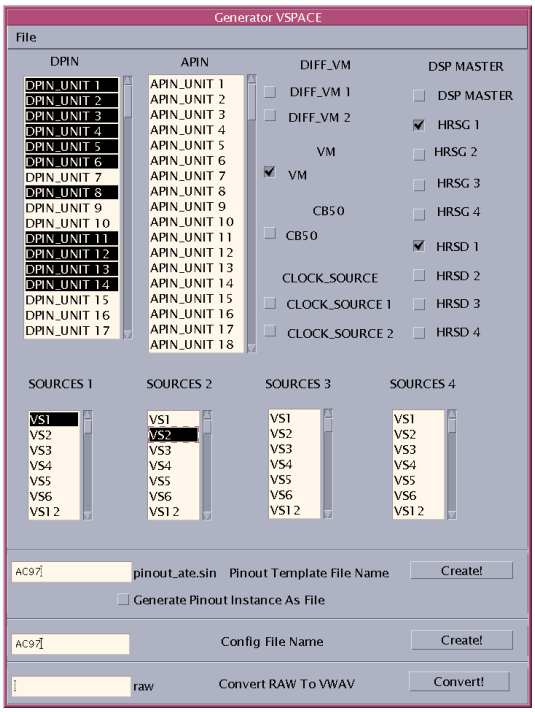



**Abbildung 4.7: Konfigurationswerkzeug zur Auswahl der Instrumente**

Mit Hilfe dieser Einstellungen (Abbildung 4.7) kann ein Modell generiert werden, welches die einzelnen Verbindungen der Instrumente untereinander enthält (Abbildung 4.8). Dieses Modell kann dann als Testermodell gehandhabt werden. Es ist zu überlegen, ob künftig aus dem

Testprogramm ein Modell der Testmaschine mit der korrekten Verschaltung der benötigten Instrumentenmodelle generiert werden kann.

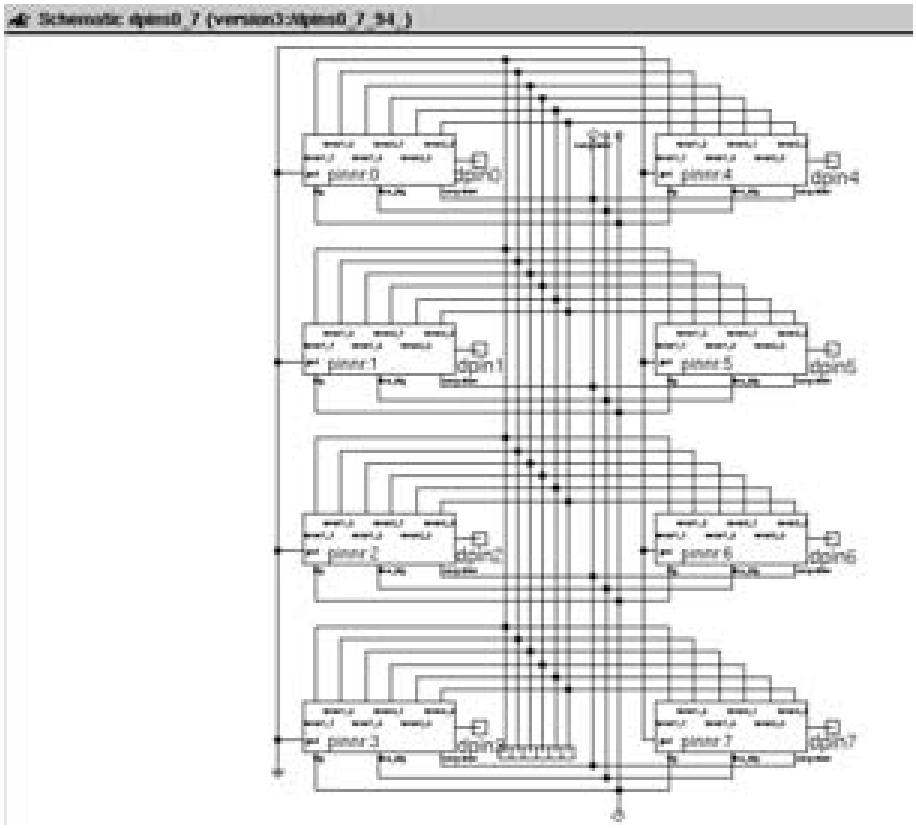



**Abbildung 4.8: Verbindungen zwischen Pinelektroniken innerhalb eines Pin-Modules**

Vor dem Start der Simulation müssen die Verbindungen zwischen den einzelnen Instrumenten und dem DUT festgelegt werden. Dabei sind zusätzliche Bauelemente zu berücksichtigen, die auf dem DIB untergebracht werden. Hierzu muß es ein einfaches Schematic-Entry geben (Abbildung 4.9). Simulatoren stellen solche Eingabefunktionen in der Regel zur Verfügung. Diese sind noch so zu konfigurieren, daß sie die Darstellung einer Verbindung eines einzelnen Bauelements mit einem großen System in übersichtlicher Weise erlauben.

### 4.2.3 Simulationssteuerung

Hinsichtlich der Simulationssteuerung ergeben sich beim virtuellen Test im Vergleich zur Simulation eines Bausteins im Design-Fluß grundsätzliche Unterschiede, die es zu beachten gilt.

Ein einzelner Baustein innerhalb einer Testbench wird durch Standard-Quellen stimuliert. Diese Aufgabe übernimmt das Testsystem, das aber seinerseits einen Stimulus benötigt, nämlich das

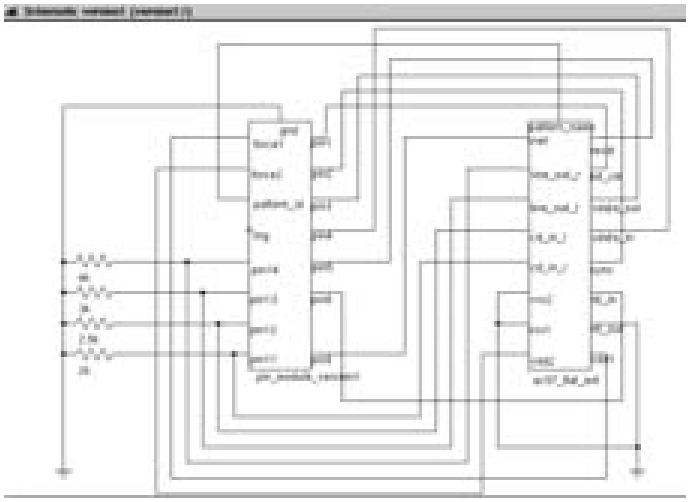


**Abbildung 4.9: Konfigurationswerkzeug zur Verbindung von ATE und DUT**

Testprogramm, das wiederum in die Entwicklungsumgebung (SPACE) eingebettet ist. Somit muß bei der Einbindung eines Testermodells in den Simulator eine Kontrolleinheit geschaffen werden, die eine Kommunikation (IPC = Interprocess Comunication) zwischen dem Simulator und dem Testprogramm erlaubt (Abbildung 4.5). Über diese Verbindung werden große Datenmengen ausgetauscht. Das sind alle Setup-Informationen der Instrumente sowie deren Meßergebnisse bzw. Rückmeldewerte. Da eine Schnittstelle zum Datenaustausch immer einen Engpaß darstellt, muß das Interface, an der der Simulator an das Testprogramm angedockt wird, sorgfältig ausgewählt, dimensioniert und spezifiziert werden.

#### 4.2.3.1 Anforderungen

Es wurden bekannte Verfahren zur Kommunikation zwischen Testprogramm und Simulator untersucht (Abschnitt 3.3) und ihre Eignung für die Erfüllung der Anforderungen an die optimale Verbindung festgestellt.

Drei bekannte Verfahren zum Datenaustausch zwischen Prüfprogramm und Simulationsumgebung sind:

- Kopplung zwischen Tester-Emulator und Simulator
  (Teradyne IMAGE ExChange, Abschnitt 3.1.2.8)
- Generierung der Modellparameter aus einem Testsetup (Schematic)
  (Dantes, Abschnitt 3.1.2.1)
- Umsetzung des Prüfprogrammes in eine Hochsprachenbeschreibung (z.B. VHDL, Verilog)

Alle drei Verfahren wurden für nicht geeignet bewertet. Der entscheidende Nachteil beim ersten Verfahren ist der hohe Kommunikationsaufwand, der zu einer Reduzierung der Leistungsfähigkeit der Simulationsumgebung führt (Abschnitt 3.3.1). Die beiden anderen Verfahren sind nicht in die Standardtestumgebungen integrierbar.

Zur Überwindung dieser Unzulänglichkeiten wurden folgende Leitlinien für das verwendete Softwarekonzept aufgestellt:

- Aktiver Datenaustausch zwischen Prüfprogramm und Simulator
- Prüfprogramm und Simulator sind eigenständige Prozesse
- Datenaustausch via Standard Unix Software-Sockets
- Keine Einbeziehung eines Tester-Emulators (Abschnitt 3.3.2)
- Synchronisationsmechanismus zwischen Prüfprogramm und Simulator (Abschnitt 4.2.3.2)
- Schnittstelle für Debugging-Werkzeuge muß gegeben sein (Abbildung 2.14)
- Unabhängigkeit von Simulator und Testsystem

Das vorgestellte Kommunikationsprinzip (Abschnitt 4.1.3) mußte im Laufe der Arbeit erweitert werden, um alle Aufgaben, die während eines Testprogrammlaufs auftreten, verarbeiten zu können. So müssen Ergebnisse zurückgelesen werden. Hierbei gibt es wiederum Besonderheiten, wie zum Beispiel Multiple-Read. Das heißt, mit einem Befehl wird ein ganzer Satz an Instrumenten abgefragt, die ihre Ergebnisse dann in einem Datenstrom an das Testprogramm liefern. Oder eine Meßanfrage wurde abgesetzt, das Ergebnis kann aber erst zu einem späteren Zeitpunkt geliefert werden. Hierzu müssen Speicheradressen verwaltet werden, die ein späteres Rückschreiben des Ergebnisses in die Zielvariable ermöglichen. Um solche Fälle erfolgreich zu bearbeiten, mußte ein Protokoll festgelegt werden, das diese Anforderungen erfüllt.

Im Hinblick auf die rechenintensive Simulation war die Erweiterung der Kommunikation auf den Netzwerkmodus sinnvoll, um die Simulationsmaschine von allen zusätzlichen Aufgaben zu entlasten. Mit dieser Funktionalität ist es möglich, das ATE-Betriebssystem auf einem anderen Rechner als dem Simulationsrechner auszuführen.

Bei der Kommunikation über das Bussystem zum realen Tester werden sämtliche Befehle des Testprogramms in den Hardwaretreibern in einen Binärcode zerlegt, mit dem die Instrumente arbeiten können. Eine Verbindung zum Simulator an dieser Stelle würde einen riesigen Datenstrom verursachen, der die Simulationsgeschwindigkeit heruntersetzt. Da der Datenaustausch

zwischen zwei UNIX-Prozessen ein zeitintensiver Vorgang ist, der optimiert sein muß, um dem virtuellen Testsystem eine größtmögliche Leitungsfähigkeit zur Verfügung zu stellen, wird die Schnittstelle, auf der die Testsystemsoftware mit dem Simulator verbunden wird so gewählt, daß sie unterhalb des Instrument Control System (ICS) und oberhalb der Hardwaretreiber liegt (Abbildung 4.10). Das ICS überprüft die Syntax, Parameter-Bereichs-Überschreitungen, Hardware-Konflikte, optimiert redundante Befehlsfolgen. Damit ist eine zweite Programmierung dieser Kontrollfunktionen innerhalb des Simulators unnötig. Um eine Kommunikation an dieser Stelle vorzunehmen wurde das ICS zum ICSV (Instrument Control System Virtual) erweitert, die Befehle gelangen nun über das ICSV zum Simulator.

Im realen Betrieb lädt das Testerbetriebssystem (SPACE) die Daten und liefert diese über das Bussystem an die reale Testmaschine. In der virtuellen Umgebung wäre diese Vorgehensweise auch denkbar, doch müssen die Datenmengen über die IPC vom Testerbetriebssystem zum Simulator übertragen werden. Jede Kommunikationsschnittstelle stellt eine Engstelle dar, welche die verbundenen Prozesse in ihrer Ausführungsgeschwindigkeit bremst. Somit war hier ein Konzept zu entwickeln, welches die Patterndaten effizient und schnell in den Simulator lädt und die Daten dort so hinterlegt, daß der Simulator für seine Berechnungen zügig auf diese Daten zugreifen kann (Abbildung 4.10).

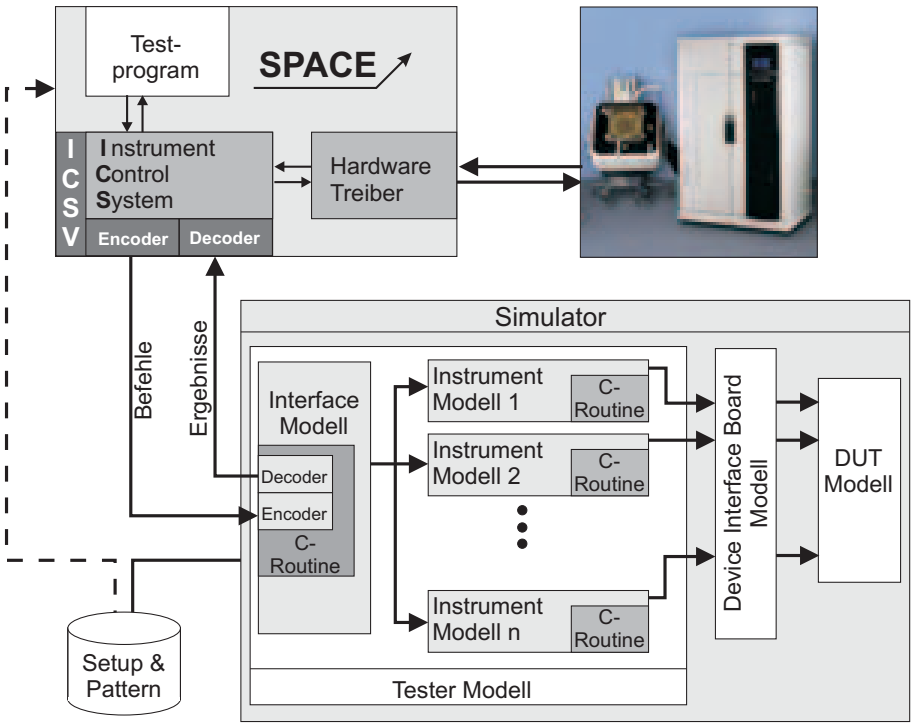



**Abbildung 4.10: Konzept des virtuellen Testsystems**

#### 4.2.3.2 Lösung: Erweiterung der Kommunikation

Das in Kapitel 4.1.3 zitierte Basiskonzept der Kommunikation mußte für den „virtuellen Test“ in den nachfolgend behandelten Punkten grundlegend erweitert werden.

**Kodierung / Dekodierung**

Zur Steuerung des Simulators ist ein Interface-Modell notwendig, das in jedem Simulator aufgebaut werden kann, der Verhaltensmodelle unterstützt und eine C-Schnittstelle besitzt. Dieses Interface-Modell, das die C-Schnittstelle des Simulators nutzt, ist notwendig um die Verbindung zwischen Simulator und SPACE herzustellen.

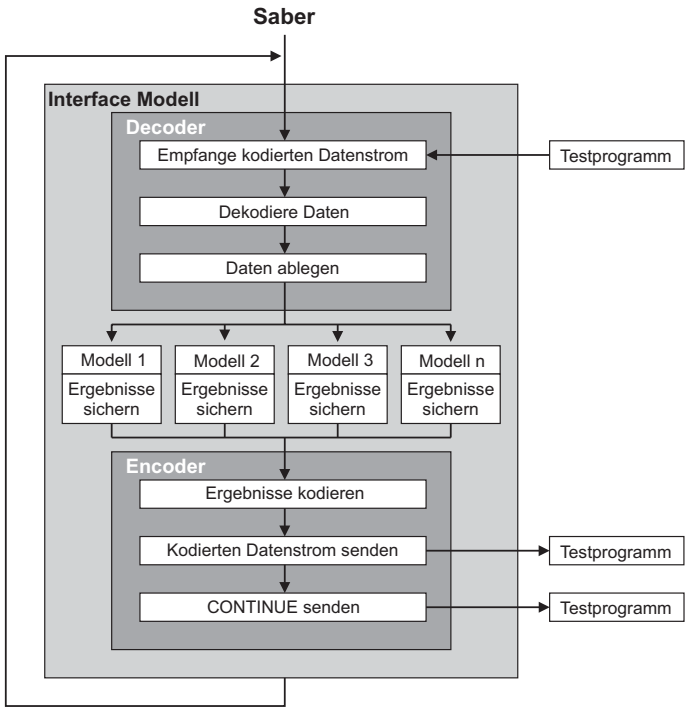



**Abbildung 4.11: Dekodierung des Datenstroms im Saber**

Das ICSV übernimmt auf Grundlage des Testprogramms die Ansteuerung der Instrumente. Im Simulationsfall werden die Daten der Instrumentenansteuerung kodiert und blockweise zum Saber verschickt. Der Datenstrom aus ankommenden Befehlen und Parametern wird in der Funktion dpinGetTp.c, die vom Interface-Modell aus aufgerufen wird, ausgewertet. Die Auswertung umfaßt zunächst die Dekodierung (Abbildung 4.11) der eingehenden Control-Blöcke aus dem Testprogramm. Die dekodierten Daten werden in einer allgemein zugänglichen Datenstruktur (Kapitel 4.2.4.2) abgelegt. Das Interface-Modell schreibt auf eine modell-interne Triggerleitung

(trig) die Instrumenten-ID des Modells, das mit dem dekodierten Control-Block programmiert wird (Abbildung 4.10, Abbildung 4.20). Die Modelle arbeiten den in der Datenstruktur abgespeicherten Control-Block ab. Dieses Vorgehen wird am Beispiel des DPIN in Abschnitt 4.2.5.2 eingehend erläutert. Ergebnisse dieser Berechnungen werden gesichert, anschließend kodiert und an das Testprogramm zurückgesendet.

Die Dekoder auf jeder Seite sind für die richtige Plazierung der Daten zuständig. So ist der auf der Testprogramm-Seite beispielsweise für Zeitkorrekturen, für die Ausgabe von Saber-Fehlermeldungen und dem Zurückschreiben der Simulationsergebnisse zuständig.

### Zeitliche Synchronisation

Bei der Erklärung der zeitlichen Synchronisation nebenläufiger Prozesse stößt man auf die Begriffe „Echtzeit“, „Simulationszeit“ und „Rechenzeit“ . Die Definitionen bzw. die Benutzung dieser Schlagworte ist oftmals widersprüchlich, so daß vor ihrer Verwendung eine Definition gegeben werden soll.

**Definition: „Simulationszeit“** [LAN95]
Die Größe, welche im Modell den Bezug zur Zeit im realen System herstellt, bezeichnet man als Simulationszeit.

**Definition: „Simulationszeitraum“**
Zeitspanne über die ein System analysiert wird.

**Definition: „Rechenzeit“** [GRM01]
Benötigte Zeit für die Analyse eines Systems über einen gegebenen Simulationszeitraum.

**Definition: „Echtzeit“** [WIK04]
Echtzeit bedeutet, daß das Ergebnis einer Berechnung innerhalb eines gewissen Zeitraumes garantiert vorliegt, d.h. bevor eine bestimmte Zeitschranke erreicht ist.
Ein Echtzeit-System (engl. real-time system) muß also ein Berechnungsergebnis nicht nur mit dem richtigen Wert, sondern auch noch rechtzeitig liefern. Andernfalls hat das System versagt. Eine genaue Definition der Zeitschranke hängt von der Anwendung ab und kann daher nicht allgemein angegeben werden. Ein Echtzeit-System reagiert also auf alle Ereignisse rechtzeitig und verarbeitet die Daten „schritthaltend“ mit dem technischen Prozess.
Um die Echtzeit-Fähigkeit eines Echtzeit-Systems theoretisch nachweisen zu können, müssen die Häufigkeit der externen Ereignisse, die Laufzeit der einzelnen Programmteile und die Zeitschranken bekannt sein.

Rechner zur Steuerung von technischen Einrichtungen oder Prozessen wie Maschinen, verfahrenstechnische Anlagen, Autos, Flugzeugen usw. sind immer Echtzeit-Systeme.

Der Begriff Echtzeit ist damit für Reaktionszeiten von Systemen vorbelegt und kann im Zusammenhang mit Simulationen falsch verstanden werden. Deshalb soll Echtzeit im Folgenden nicht verwendet werden. Dafür wird der Begriff der „realen Zeit“ eingeführt.

**Definition: „Reale Zeit“**
Die Reale Zeit ist die Zeit, die ein Vorgang in der Realität benötigt.

Diese Definition ergänzt die Definiton der „Simulationszeit“ von Lanchès [LAN95] schlüssig, denn die „reale Zeit“ wird somit auf die „Simulationszeit“ abgebildet.

Bevor auf die zeitliche Synchronisation nebenläufiger Prozesse im Bereich des virtuellen Tests eingegangen wird, soll noch ein Begriff definiert werden, der im Bereich Test anzusiedeln ist und in diesem Zusammenhang ebenfalls zu Mißverständnissen führen kann.

**Definition: „Laufzeit“**
Unter Laufzeit versteht man die Zeit, die ein Testprogramm zu seiner Ausführung benötigt.

Die Simulation eines Testsystems wird aufgrund der Komplexität des Systems nie in „realer Zeit“ erfolgen können. Bei der Entwicklung und Bearbeitung von Testprogrammen kann die Zeitgenauigkeit jedoch von großer Bedeutung sein, besonders wenn es sich um ein System handelt, welches im Hinblick auf „Laufzeit“ stark optimiert ist. Im Fall von „Virtual Test“ würde eine Nichteinhaltung der realen Zeitverhältnisse bedeuten, daß Testprogramme, die in der Simulationsumgebung entwickelt werden, nicht auf einem realen Testsystem funktionieren, weil z.B. Wartezeiten nicht eingehalten werden. Der umgekehrte Fall ist ebenso vorstellbar. Zudem kann man ohne Berücksichtigung der realen Zeitverhältnisse keine Testprogrammoptimierung auf dem Simulator durchführen.

Um diese Eigenschaft auch in Verbindung mit dem Simulator zu erreichen, muß auf die „reale Zeit“ verzichtet werden und statt dessen auf eine systembezogene „relative Zeit“ zurückgegriffen werden.

**Definition: „Relative Zeit“**
Die relative Zeit läuft weder schneller noch langsamer ab. Sie wird jedoch an verschiedenen Punkten gestoppt, an anderen Stellen wieder gestartet oder „nachgestellt“. Somit kann man eine Zeitbasis für nebenläufige Prozesse schaffen, welche „Real-Zeit-Eigenschaft“ für beide Systeme realisiert.

Die „relative Zeit“ muß in diesem Zusammenhang eingeführt werden, da der SPACE in „realer Zeit“ arbeitet, während sich die Simulation auf die „Simulationszeit“ bezieht, die aber nur in

einer bestimmten „Rechenzeit“ ausgeführt werden kann. Da die verschiedenen Zeiten auseinanderlaufen, muß die „reale Zeit“ im SPACE nachgestellt werden und man erhält die „relative Zeit“. Abbildung 4.12 veranschaulicht die Verwendung der eingeführten Begriffe und den Austausch der Zeitmarken, der nachfolgend programmtechnisch beschrieben wird und in Abbildung 4.13 visualisiert ist.

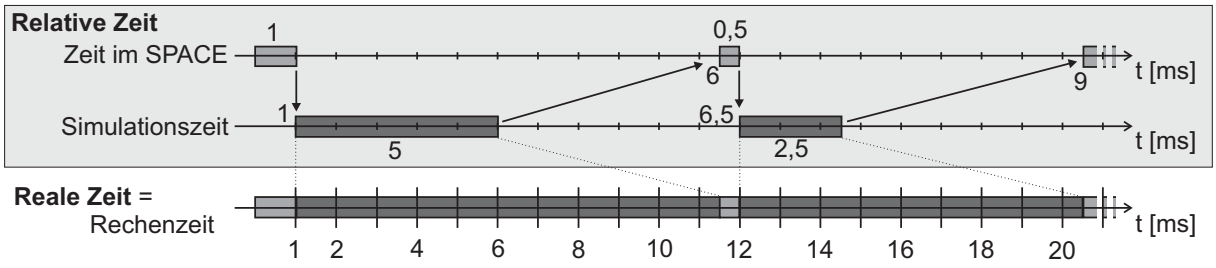



**Abbildung 4.12: Zeitliche Synchronisation von Space und Simulator**

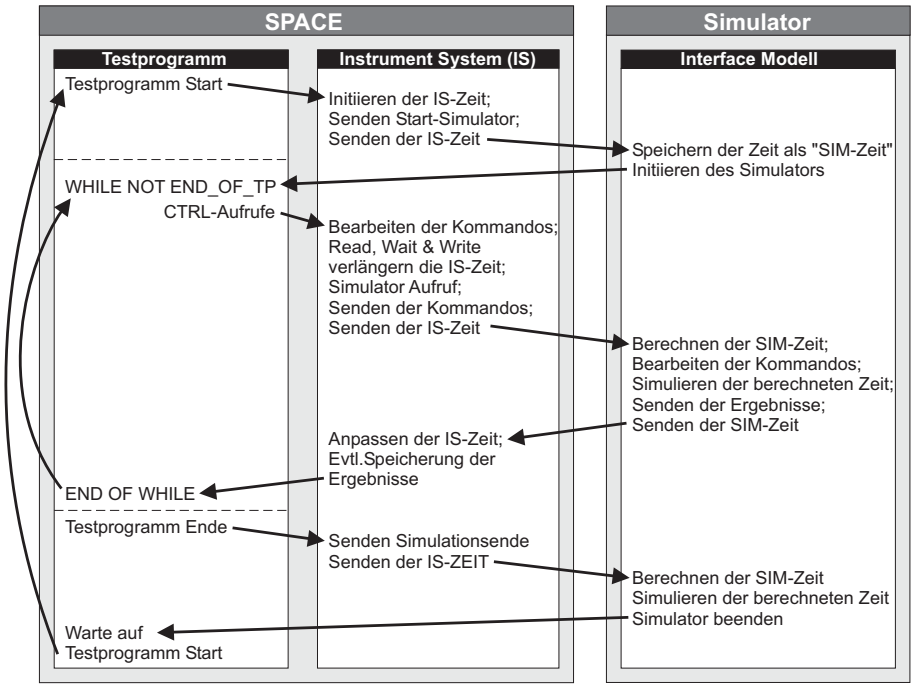



**Abbildung 4.13: Funktionsweise der Synchronisation von Space und Simulator**

Nach dem Start des Testprogramms wird die „reale Zeit“ gestartet und an den Simulator gesendet. Dort wird die Zeit abgespeichert und die Simulation initialisiert. Im Simulator werden die einzelnen Instrumente, wie auch im realen Testsystem, initialisiert. Der Simulator ermittelt die dafür verwendete Zeit, schickt sie zurück an das Instrumenten System (IS) und hält die Simulation an. Die Uhr in der Testprogramm-Umgebung wird auf die vom Simulator übermittelte Zeit nachjustiert, bevor der nächste Satz an Kommandos bearbeitet wird. Das IS ermittelt die Zeit, die es zum Abarbeiten der Befehle benötigt hat und sendet diese mit dem zu simulierenden

Befehlscode an den Simulator. Im Simulator wird ein Zeitschritt erzeugt, der den Simulator wieder auf die Zeit bringt, die durch die Abarbeitung des Testprogramms im IS vergangen ist. Auf diese Weise synchronisieren sich beide Systeme stets auf die aktuelle „relative Zeit“, die sich natürlich von der realen Zeit aufgrund der langsameren Simulation entfernt. Nach außen - für den Anwender - erscheint der Koppelmechanismus wie in der Realität, nur langsamer.

### Aufbau eines Datenstroms

Die Kommunikation, die in [GOL98-1] entwickelt und in Kapitel 4.1.3 vorgestellt wurde, mußte hinsichtlich des Datenstroms komplett umgestellt werden. Ein Rückschreiben der Ergebnisse in die Variablen des Testprogramms war mit der alten Kommunikation nicht möglich, denn es wurde dem System nicht mitgeteilt, wohin der Datentyp des Ergebnisses zurückgeschrieben werden muß.
Alle Werte, die von der Hardware bzw. Simulation zurückgelesen werden sollen, müssen vom Simulator in die entsprechenden Variablen geschrieben werden. Dazu muß das Instrumenten-System den Speicher für die zurückzulesenden Werte reservieren. Das Rückschreiben dieser Werte geschieht mit Hilfe des Datenstroms, der vom Simulator kommt. Dazu muß dem Simulator zuvor die reservierte Speicheradresse über den Datenstrom mitgeteilt werden. Diese Adresse wird zusammen mit dem „Meßwert“ wieder an das Testprogramm zurück geschickt. Auf diese Weise kann das Instrumenten-System das Ergebnis in der passenden Variable ablegen.

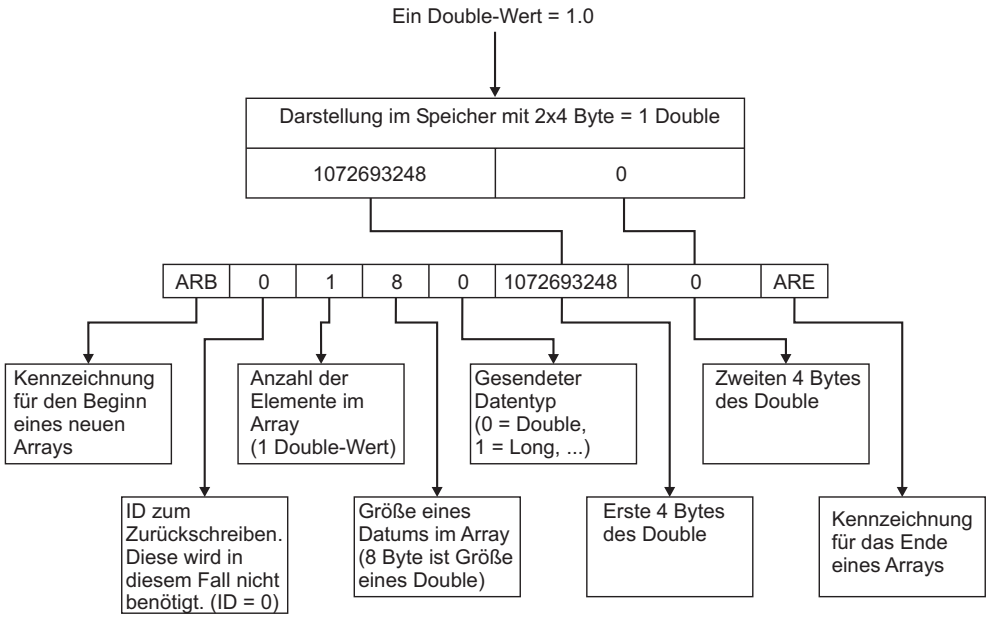



**Abbildung 4.14: Aufbau des Datenstroms**

Bisher konnten keine komplexen Datentypen dargestellt werden. Die Verschlüsselung der Daten-

typen war in der bisherigen Version nicht einheitlich und mußte für jeden neuen Datentyp von Hand aufgesetzt werden. Diese Datentypen sind aber notwendig, wenn man an das Rückschreiben der Ergebnisse, an das Senden von Arrays oder an die Fehlerbehandlung denkt.

Um diese Anforderungen zu erfüllen muß ein Datenstream aufgesetzt werden, der unabhängig vom Datentyp ist. Abbildung 4.14 zeigt den Aufbau des neuen Datenstroms. Daten werden im Datenstrom in Arrays gekapselt, die wiederum ineinander geschachtelt werden können. Auf diese Weise kann man komplexe Datentypen, die immer aus mehrfach verschachtelten Arrays und einfachen Datentypen bestehen, in beliebiger Komplexität übertragen. An den Anfang des Arrays werden Zusatzinformationen wie Anzahl der Elemente des Arrays, Größe eines Datums, Art des Grund-Datentyps hinterlegt. Indem komplexe Strukturen auf Grund-Datentypen zurückgeführt werden wird erreicht, daß die Daten intern ein einheitliches Format haben.

Damit ergeben sich folgende Vorteile mit der neuen Kommunikation:

- Marshalling [SCH02]
  „Flachklopfen“ komplexer Strukturen zum Zwecke der Datenübertragung. Dadurch wird der Mechanismus maschinen- und systemunabhängig.
- Kein komplexes Interpretieren des Datenstroms auf der Empfängerseite. Komplexe Datentypen sind im kodierten Datenstrom einfach aufgebaut und benötigen keinen speziellen Dekodier-Algorithmus.
- Automatisches Anlegen des benötigten Speicherbereichs für die ankommenden Daten. Die Daten können während der Interpretation im Speicher angelegt werden. Auch mehrstufig verzeigerte Arrays können ohne Kenntnis des Datentyps verwaltet werden.
- Rückschreiben der Ergebnisse erfolgt mit Hilfe der ID, die zum Testprogramm zurückgesendet wird und damit das Füllen des Speichers auf dieser Seite garantiert.

Nachteile der Kommunikation:

- Der Datenstrom ist für den Menschen kaum mehr zu lesen, denn durch die Umwandlung in „Long-Werte“ verlieren vor allem „Double-Werte“ jeglichen Bezug zur eigentlichen Zahl.
- Der Datenstrom wird durch die Kodierung um den Faktor 5 länger
- Durch das Kodieren wird die Kommunikation etwa um den Faktor 3 langsamer

Ein Zeitvergleich (Tabelle 4.1) zwischen der alten und der neuen Kommunikation zeigt eine Verlangsamung um etwa den Faktor 1.2. Dieser Faktor, der nur einen Trend angeben soll, zeigt

den Mehraufwand, der durch die Unterstützung komplexer Datentypen notwendig wurde. Dabei muß die Zeit der Funktionen „TPSIM_ctrlFunc“ und „TPSIM_fetchCmd“ betrachtet werden, denn in diesen sind die Kodierfunktionen untergebracht. Dazu wurden die Ausführungszeiten für einen typischen Befehlsblock (Abbildung 4.15) erfaßt.

| Funktion | Alte Kommunikation Zeit [s] | Neue Kommunikation Zeit [s] |
|---|---|---|
| TPSIM_readFunc | 0.01608 | 0.01608 |
| TPSIM_ctrlFunc | 0.01063 | 0.01342 |
| TPSIM_fetchCmd | 0.02626 | 0.03089 |

**Tabelle 4.1: Zeitmessung der Kommunikation**

```
CTRL DPIN
   DO_FOR_SIGNAL_LIST pinList
REMOVE_FROM         PDCL
CONNECT_TO          LPMU
SELECT_MODE         FORCE_VOLTAGE
SET_VOLTAGE         5 V
SET_EXP_CURRENT     20 uA
SET_GATE            ON
FLUSH
END;
```

**Abbildung 4.15: Befehlsblock zur Zeitmessung der Kommunikation**

### Netzwerkbetrieb und Stabilität

Die Kommunikation soll netzwerkübergreifend sein, damit das Testprogramm mit der Testsystemsoftware und die Simulation nicht die gleiche Maschine belasten und damit die Ausführungsgeschwindigkeit drosseln. Für die Simulation ist es damit möglich, sich die geeignetste Maschine im Netzwerk auszusuchen.

Die in der Arbeit von Goldbach [GOL98-1] vorgeschlagene Socketverbindung ist auch für eine netzwerkweite Kommunikation verwendbar. Allerdings treten Probleme auf, wenn der Client seinen Socket nicht wieder freigibt. Dies passiert beispielsweise, wenn ein Testprogramm im Debugger geladen ist und bei seiner Ausführung abstürzt. In diesem Fall wird oftmals der Socket nicht mehr freigegeben, da der Vaterprozeß (Debugger) noch existiert. Dieser noch existierende Socket kann aber nicht mehr verwendet werden. Im schlimmsten Fall muß die Maschine vollständig gebootet werden. Diese Gefahr besteht in ähnlicher Weise ebenfalls, wenn in den Simulationsmodellen etwas schiefgeht und dadurch der Simulator unerwartet beendet wird. Es ist also problematisch, den Kommunikationskanal ständig offen zu halten, da die Prozesse bei einem Programmabsturz keine Möglichkeit haben, diesen ordnungsgemäß zu schließen. Die Möglichkeit, die Verbindung nur für die Zeit der Übertragung zu öffnen ist ähnlich problematisch, da der Server nicht feststellen kann, ob der Client aufgrund eines Absturzes keine Anforderung

mehr stellt. Der Server sperrt damit den Port für jeden anderen Prozeß, da er vom Client keine Bestätigung über das Ende der Kommunikation erhalten hat. Ein Timeout kann nicht eingeführt werden, da sich bei der Simulation Antwortzeiten im Stundenbereich ergeben können.

Eine andere Möglichkeit ist der Einbau eines Überwachungsprozesses. Dieser meldet den Client sofort beim Server ab, falls der überwachte Prozeß nicht mehr existiert. Diese Möglichkeit bietet eine vielversprechende Lösung und ist in der Informatik unter dem Begriff „Liveness“ bei nebenläufigen Prozessen bekannt [KLE04].

Analoges muß natürlich auch für den Fall gelten, daß der Simulator abstürzt. Auch für diesen Fall stellt die Variante mit dem Überwachungsprozeß eine erfolgversprechende Implementierung dar, da der Überwachungsprozeß unabhängig von den absturzgefährdeten Prozessen agiert. Damit ist es möglich, den korrespondierenden Prozeß im Fehlerfall sauber zu beenden und die Socketverbindung vollständig zurückzusetzen.

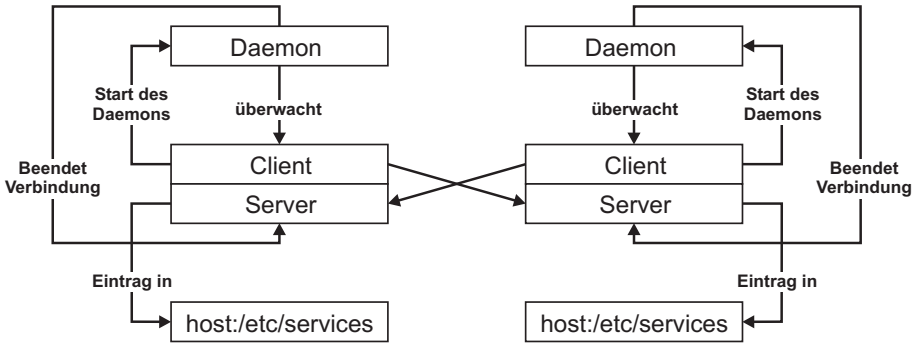



**Abbildung 4.16: Funktionsweise des Daemons**

Um die gewünschte Stabilität zu erreichen, müssen die Sockets in ihren Eigenschaften streng getrennt sein. Das heißt, jeder Socket kann nur Server oder nur Client sein, wobei jeder Server-Socket nur Daten empfängt und jeder Client Socket nur Daten sendet. Damit besitzt jeder der beiden Prozesse einen Server- und einen Client-Socket, um damit Daten senden und empfangen zu können. Der reguläre Zustand der beiden Prozesse besteht darin, daß Client und Server miteinander kommunizieren. Geschieht dies über ein Netzwerk, wird die Nummer des jeweilgen Servers in die Datei „/etc/services“ des jeweiligen Hosts eingetragen.
Damit im Falle eines Absturzes jeder Client ordnungsgemäß beim Server abgemeldet wird, startet jeder Client seinen eigenen Daemon (Abbildung 4.16). Der Daemon erhält als Parameter die Prozeß-ID des Vaterprozesses und den Hostnamen, auf dem der Prozeß ausgeführt wird. Der Daemon selbst startet einen Trap-Handler, für den Fall, daß der Vaterprozeß stirbt, damit nach Beendigung des Vaterprozesses der Daemon, der ein Kindprozeß ist, nicht mit in den Tod gerissen wird.
Der Daemon wacht also über den Zustand des Programms, das ihn gestartet hat. Beendet

sich das Programm auf unvorhergesehene Weise, schickt der Daemon dem jeweiligen Server eine Bestätigung über das Ende der Anforderungen seines Clients. Kann sich der Vaterprozeß selbst ordnungsgemäß beenden, so muß er seinen Kindprozeß terminieren, bevor er sich selbst beendet und den Server über das Ende der Kommunikation informiert.

### 4.2.4 Datenverwaltung

#### 4.2.4.1 Anforderungen

Während eines Bauelement-Tests fallen große Datenmengen an, die es zu verwalten gilt. Diese Datenmengen entstehen dadurch, daß jeder Zustand der gesamten Testmaschine zu jedem Zeitpunkt in abfragbaren Daten vorliegen muß. Das enthält Setup-Daten aller Instrumente, die Ergebnisse und Meßwerte einzelner Messungen, Testmuster und Pinlisten, die Daten abgetasteter analoger Signale und dergleichen mehr. Somit ist eine Verwaltung aufzubauen, die mit diesen Daten umgehen sowie diese bei Bedarf an das Testprogramm zurückliefern kann und dort an die passende Speicheradresse schreibt. Somit erscheint es sinnvoll, sich an der Datenverarbeitung und -behandlung des ATE-Betriebssystems zu orientieren. Auf diese Weise kann die Vollständigkeit der implementierten Daten überprüft werden.

**Testprogramm-Daten**

Das Testprogramm liegt komplett nur im Steuerrechner vor und wird dort abgearbeitet. Die Befehle werden blockweise an das Testsystem geschickt. Die Abarbeitung der Befehle aus dem Testprogramm und damit der Austausch der Befehle und Ergebnisse muß sich vom Ablauf an der realen Kommunikation orientieren, so daß zu jedem Zeitpunkt das Testprogramm aus dem Debugger heraus angehalten und variiert werden kann. Zudem müssen die Befehle, die aus Zusatzwerkzeugen, wie zum Beispiel dem Panel abgesetzt werden, vom virtuellen Testsystem verarbeitet werden.

**Patterndaten**

Das Pattern liegt im Gegensatz zum Testprogramm vollständig im realen - bzw. virtuellen Testsystem vor. Die Datenstruktur muß so gestaltet sein, daß Debug-Kommandos in den Befehlsstrom eines Patternlaufs jederzeit eingreifen können. Das heißt, es ist notwendig, erst während der Simulation den nächsten Simulationspunkt aus den Patterndaten zu berechnen. Beim Debuggen eines Patterns kann beispielsweise der zeitliche Ablauf der digitalen Signalfolgen vom Testingenieur jederzeit beeinflußt und geändert werden.

Die Patterndaten können in einem der drei vorgestellten Formate vorliegen (siehe Abschnitt 2.2.4.2 und Anhang A.3). Es muß eine Entscheidung getroffen werden, welches Format geeignet ist. Dabei ist zu berücksichtigen, inwiefern die Daten aufbereitet werden müssen, um möglichst keine Angriffsstellen für Konvertierungsfehler zu bieten.

Innerhalb der Pattern wird der Zustand jedes digitalen Pins zu jedem Zeitpunkt beschrieben. Damit ist es möglich, daß in dieser Abfolge redundante Vorgänge aufgeführt sind. Das sind Abfolgen, bei denen sich das elektrische Signal an einem Pin nicht ändert. Die Modelle müssen nun ihrerseits eine gewisse Intelligenz besitzen, um in diesen Datenmengen redundante Vorgänge zu erkennen und in die Berechnung des Timings während der Simulation einfließen zu lassen. Das Erkennen und Herausfiltern dieser redundanten Vorgänge ist notwendig, um akzeptable Simulationszeiten zu erzielen. Jedes Anstoßen der Simulation, um die Zustände zu einem bestimmten Zeitpunkt zu berechnen, kostet Zeit. Deswegen sollten Berechnungen von Zuständen, bei denen sich nichts ändert, eingespart werden.

#### 4.2.4.2 Lösung: Dynamischer Aufbau des Datenfelds

Die vom Testprogramm ankommenden Befehle und Parameter werden in einem Datenfeld abgespeichert. Die Struktur ist den Unterbaugruppen bzw. den syntaktischen Zusammenhängen nachgebildet.

Das Datenfeld bildet einen Teil des ISV (Instrument System Virtual), deshalb soll im Folgenden kurz dessen Struktur beschrieben werden. Jedes Instrument muß als autonome Struktur gehandhabt werden. Das bedeutet, daß jedes Instrument jederzeit austauschbar sein muß. Das führt soweit, daß die Instrumente erst zur Laufzeit bekannt werden und erst damit Bestandteil des ISV werden. Diese Autonomität wird allerdings durch die oftmals informationstechnisch untrennbare Vernetzung der Instrumente eingeschränkt. Dieser Einschränkung muß vom ISV und vom jeweiligen Instrument Rechnung getragen werden. Der Vorteil der autonomen ISV-Strukturen liegt in der Möglichkeit, die Instrumente parallel zu entwickeln.

Das ISV teilt sich in ICSV (Instrument Control System Virtual), Definitionen der virtuellen Instrumente, die virtuellen Instrumente selbst und die kundenspezifischen Instrumente auf. Das ICSV seinerseits untergliedert sich wie folgt:

- Memory Management und Stream-Analyse
  Der Speicher für die virtuellen Instrumente wird von diesem Programmteil angelegt und verwaltet. Der eingehende Datenstrom wird analysiert und aufbereitet und zur Speicherung im Datenfeld für die einzelnen Instrumente vorbereitet.

- Kommunikation
  Diese legt die Schnittstelle zwischen dem Simulator und der Testsystem-Software fest. Es steuert den Aufbau und die Verwaltung der Verbindung zwischen beiden Programmteilen.

- Steuermodule
  Die allgemeine Schnittstelle zu allen Instrumenten. Jeder Zugriff läuft über das ICSV. Es verwaltet die Gemeinsamkeiten der Instrumente und steuert die Speicherverwaltung und Stream-Analyse.

- Generische Teile
  Dieser Teil dient zur Analyse der vorhandenen Instrumente und deren Eigenschaften und stellt deren Verfügbarkeit im ISV sicher. Er stellt Bezüge zu den reellen Instrumenten her und verwaltet deren Umsetzung.

Während der Initialisierungsphase wird mit Hilfe dieser Programmodule die Datenstruktur angelegt, in der die vom Testprogramm ankommenden Informationen abgelegt werden, welche die Instrumentenmodelle während der Simulation benötigen. Dazu müssen in die allgemeine Datenstruktur instrumentenspezifische Datenfelder eingehängt werden. Dies geschieht während der Initialisierungsphase mit genau den Instrumenten, die während der Simulation benötigt werden und dadurch in der Netzliste des Simulators aufgeführt sind.

Abbildung 4.17 gibt einen Einblick wie der dynamische Aufbau der Datenstruktur erfolgt. Das Pinmodul stellt die oberste Hierarchieebene des virtuellen Testsystems dar. Darin sind alle Instrumente aufgelistet und ihre Verbindungen untereinander definiert, wie zum Beispiel der DPIN. Da das Pinmodul als oberste Ebene immer vorhanden ist und sicher aufgerufen wird, ist hierin der Aufruf der Initialisierungsroutine für das Datenfeld untergebracht. Diese Funktion („datainit“) gehört bereits zum ICSV, da mit ihr der allgemeine, nicht instrumentenspezifische Teil des Datenfeldes initialisiert wird.
Der weitere Schritt folgt, indem die instrumentenspezifischen Datenfelder eingebunden werden. Dazu müssen zunächst die Initialisierungsroutinen dieser Felder gefunden werden, die innerhalb der C-Routinen der ATE-Instrumente programmiert sind. Die Funktion „VSIM_getInstInitFunc“ findet diese aus dem Array „instFuncNames“ heraus. Damit ist dieses Array die zentrale Position, an der sämtliche instrumentenspezifischen Initialisierungsroutinen bekannt gemacht werden müssen. Der so zurückgelieferte Funktionsname stellt den Pointer auf die Initialisierungsroutine dar. Im Beispiel wird „VSIM_dpinInit“ durchlaufen und bindet mit Hilfe der Funktionen „writeInitDataStruct“ und „VSIM_ICSV_getInstrumentPtr“, das DPIN-Datenfeld in die gesamte Datenstruktur ein, bevor die darin befindlichen Variablen initialisiert werden.

Die Implementierung des instrumentenspezifischen Datenfelds ist am Beispiel der Digitalen Pinelektronik (DPIN) im Anhang B.1 eingehend erläutert

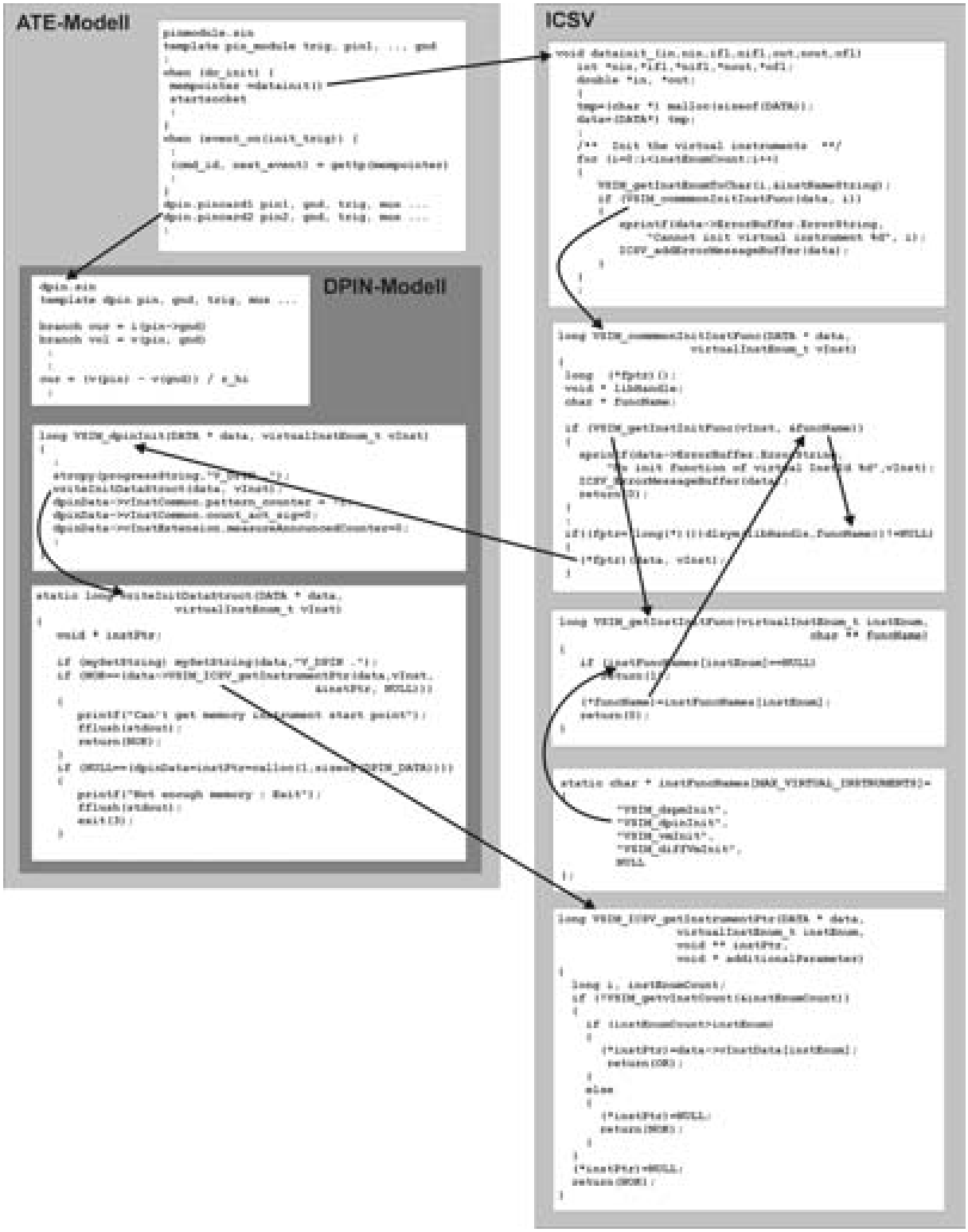



**Abbildung 4.17: Initialisierung des globalen Datenfeldes**

### 4.2.5 Modellierungstechnik und Geschwindigkeit

#### 4.2.5.1 Anforderungen

Ein ATE ist ein hochkomplexes und -flexibles System. Nahezu jedes Signal kann intern zu jedem Instrument verschaltet werden (Abbildung 2.3). Ein Testsystem besteht aus einer Vielzahl an Instrumenten, die sich aus Hunderten von Bauelementen, angefangen vom einfachen Transistor bis hin zum DSP, zusammensetzen. Somit ist es nur möglich, ein Testsystem auf Verhaltensebene zu simulieren. Vor allem wenn man in Betracht zieht, daß eine detaillierte Simulation eines einzelnen Bausteins schon Stunden oder Tage betragen kann.

**Modulare Modelle**

Modulare Modelle bieten den Vorteil der Wiederverwendbarkeit von einzelnen Komponenten, wie zum Beispiel Routinen zur Messung von Signalen, die in vielen Testsystem-Ressourcen mit unterschiedlicher Genauigkeit Verwendung finden. Zudem ermöglicht eine modulare und damit strukturierte Modellierung ein leichteres Verständnis der Modelle.

Beim Grad der Modularität ist zu beachten, daß die Granularität im Widerspruch zur Simulationsgeschwindigkeit steht. Je modularer Simulationsmodelle aufgebaut sind, desto langsamer wird die Simulation. Durch eine Vielzahl von Untermodellen werden zusätzliche Knoten in die Modellstruktur eingefügt, welche die Untermodelle ihrerseits wieder verbinden. Dadurch werden zu berechnende Matrizen größer, was die Berechnungszeit erheblich erhöht.

Hierbei ist also Geschwindigkeit gegen Übersichtlichkeit abzuwägen.

Bei der Aufteilung der Funktionalität in modulare Modelle mit Augenmerk auf die Geschwindigkeit ist eine Ausgliederung der zeitraubenden Algorithmen (die Hauptintelligenz der Modelle) in externe Routinen mit einer geeigneten Programmiersprache zu fordern. Die eigentlichen Simulationsmodelle können mit den Parametern gefüttert werden, die in diesen ausgelagerten Routinen berechnet wurden. Programmiersprachen sind simulatorunabhängig. Das heißt, die Testermodelle können relativ einfach auf einen vom Kunden gewünschten Simulator portiert werden.

Hinsichtlich der Modularität ist also ein Optimum zu finden, das sich aus folgenden Bedingungen ergibt:

- Je größer die Modularität in MAST, desto langsamer ist die Simulationsgeschwindigkeit

- Je höher der MAST-Anteil im Verhältnis zum Anteil externer Routinen, desto langsamer ist die Simulationsgeschwindigkeit

Hierzu kann allerdings keine Faustformel angewendet werden. Es hängt vielmehr sehr vom Erfahrungsschatz des Modellierers ab, welche Funktionalität in der Hardwarebeschreibungssprache und welche in externen Routinen einzubauen ist und welcher Grad der Modularität darauf anzuwenden ist.

Des weiteren hängt der Grad der Modularität eng zusammen mit der hierarchischen Struktur der Modelle.

### Schichtenmodell

Im Rahmen des virtuellen Tests war eine hierarchische Modellierung notwendig, um eine geordnete einheitliche Struktur während der Modellierung modulübergreifend zu erreichen. Auf diese Weise werden gleichzeitig die Schnittstellen der Module, die auf die nächste Hierarchieebene zugreifen, eindeutig festgelegt. Abbildung 4.18 gibt einen Überblick über die Schichten der Modellierung im virtuellen Test. Diese Schichten lassen sich auf das 3-Schichten-Modell (Datenbasis, Control-Schicht, sichtbare Schicht) [BLU98] abbilden. Die „Datenbasis“ bildet dabei das globale Datenfeld. Auf diese kann nur die „Control-Schicht“, die aus externen C-Routinen, Instrumenten- und Interface-Modellen besteht, zugreifen. Die „sichtbare Schicht“ besteht aus dem Testsystemmodell, das alle Untermodelle in sich integriert und die Schnittstelle zum DUT bildet.

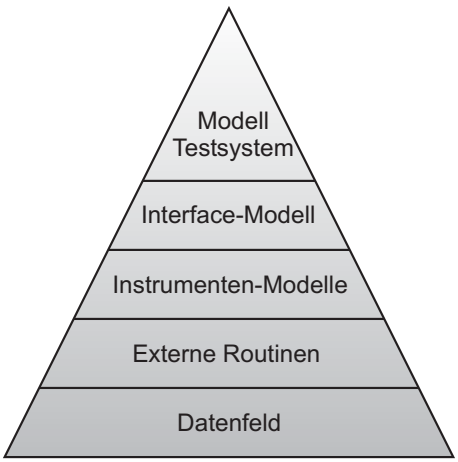



**Abbildung 4.18: Schichtenmodell**

#### 4.2.5.2 Lösung: Einsatz von C-Routinen

Um die Simulationsgeschwindigkeit zu steigern, werden zeitraubende Algorithmen (die Hauptintelligenz der Modelle) zur schnellen Abarbeitung in externe Routinen ausgelagert. Diese Routinen sind in einer geeigneten Programmiersprache zu schreiben. Darin werden die Parameter berechnet, mit denen die Simulations-Modelle gefüttert werden. Der Einsatz externer Routinen bietet einen weiteren Vorteil. Während eines Tests fallen sehr viele Daten an, die verarbeitet werden müssen. Eine Hardwarebeschreibungssprache unterstützt keine Speicheroperationen, wie sie zur Verwaltung dieser Datenmengen (Instrumenten-Einstellungen, Pattern, Meßwerte, Abtastwerte...) notwendig wären. Dies kann nur mit Hilfe dynamischer Speicherverwaltung aufgebaut werden, wie sie in einer Programmiersprache zur Verfügung steht.

Für den virtuellen Test wurde Saber als Simulator ausgewählt [GOL98-1]. Eine Steuerung des Simulators kann entweder durch C oder FORTRAN erfolgen. Die Wahl fällt auf die C-Schnittstelle, weil SPACE und das Testprogramm jeweils in C++ bzw. C geschrieben sind.

Die Schnittstelle von C zu MAST, der Modellierungssprache in Saber, ist dabei denkbar einfach ausgeführt. Ein Array im C-Teil enthält die eingehenden Variablen. Ein zweites Array die Variablen, die an das MAST-Modell zurückgegeben werden (Abbildung 4.19).

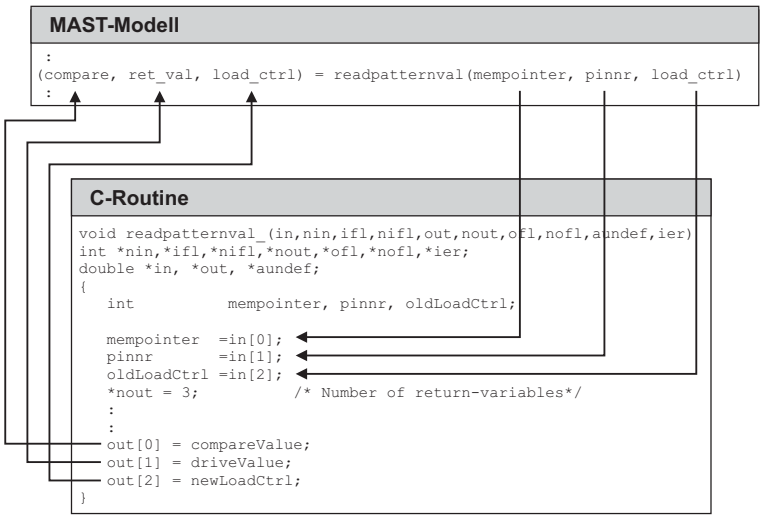



**Abbildung 4.19: Schnittstelle zwischen Saber und C**

Mit Hilfe solch eingebetteter C-Funktionen in die MAST-Modelle läßt sich unter anderem sehr einfach eine Schnittstelle zur Software des Testsystems und damit zum Testprogramm aufbauen. Dies kann durch eine Socketverbindung geschehen (Abschnitt 4.1.3), welche, um allen Anforderungen zu genügen, allerdings noch erweitert werden muß (Abschnitt 4.2.3.1).

Der Aufbau der Kommunikation ist aber nur ein Grund, warum C-Routinen bei der Modellierung

eingesetzt werden sollen. Ganz pragmatische und logische Gründe erklären den sinnvollen Einsatz dieser Modellierungsvariante.

Eine Modellierungssprache beschreibt das Verhalten eines Systems auf elektrischer - oder abstrakter Verhaltensebene (Abbildung 4.3). Diese Beschreibungen berufen sich auf physikalische Gesetzmäßigkeiten, die - in mathematischen Beschreibungen definiert - dem Simulator zur Berechnung zur Verfügung gestellt werden. Die Berechnung wird zu jedem Zeitpunkt ausgeführt, in dem sich der Zustand des Systems ändert. Auf diese Weise erfolgt die Verifikation einzelner Bauelemente und Schaltungen in überschaubarem Zeitaufwand.

In einem ATE gibt es unter anderem Controller, Speicher und DSP-Bausteine, die komplexe Funktionen zeiteffizient bearbeiten. Auf diesen Bauelementen laufen zum Teil eigene Programme und Sequenzen ab. Daten werden gespeichert, ausgewertet und verwaltet, sei es zur Steuerung der Instrumente, zur Stimulierung des DUT oder zur Erfassung der Ergebnisse. Diese Vorgänge wären in einer Hardwarebeschreibungssprache durchaus nachzubilden, allerdings unter erheblichem Modellierungs-Aufwand. Solche Modelle wären in der Simulation viel zu langsam.

Solche Funktionen können über die Integration von C-Routinen in die Modelle eingebracht werden. Die Teile in Hardwarebeschreibungssprache bilden dabei ein Rahmenmodell, welches die C-Routinen mit aufbereiteten Daten füttert. Es ist also eine Partitionierung zwischen Softwarebeschreibung und Hardwarebeschreibung vorzunehmen, ähnlich wie es bei der Problematik von „Hardware-Software-Codesign“ vorliegt.

C als Programmiersprache ist für die angesprochenen Problematiken das geeignete Hilfsmittel. Es können große Speicherbereiche und Datenmengen, wie sie bei der Arbeit mit Patterndaten auftreten, verwaltet werden. Es können Programmcodes einfach implementiert werden. Dabei sieht man sich nicht den Problemen gegenüber, wie Programmcode und damit verbundene Probleme in einer Hardwarebeschreibungssprache zu realisieren sind. Die Ausführungsgeschwindigkeit des C-Codes ist sehr schnell. Außerdem kann sehr einfach eine Schnittstelle zum Testprogramm hergestellt werden.

### Steuerung des DPIN-Modells

Nachdem geklärt ist, wie der Datenaustausch zwischen Testprogramm und Simulator abläuft, wo die Daten aufbereitet werden und warum der Einsatz von C-Routinen sinnvoll ist wird jetzt beschrieben, wie damit die Modelle innerhalb des Simulators gesteuert werden.

Die Modelle timingen.sin und dpin.sin (Abblildung 4.20), die beide den DPIN modellieren, hören gleichzeitig die Instrumenten-ID-Leitung ab und warten auf den DPIN-Aufruf (ID=309):

```
when(event_on(trig) & (trig==309)) {
```

Ist diese Abfrage wahr, erfolgt innerhalb beider Modelle die Auswertung des Datenfeldes, das wie in Abschnitt 4.2.4.2 beschrieben angelegt wurde.

Das Modell timinggen.sin stellt anhand des Datenfeldes fest, ob in der aktuellen Befehlsfolge ein Pattern abgearbeitet werden muß oder nicht. Ist dies der Fall, muß aus dem Pattern-Datenfeld der nächste Zeitpunkt gesucht werden, bei dem sich an einem der Pins ein Wert ändert. Die zugehörigen Pegel (Treiber oder Compare) werden aus dem Pattern-Datenfeld in das Datenfeld der PDCL übertragen. Der Zeitpunkt des nächsten zu simulierenden Events wird an das Modell übergeben.

Das Modell dpin.sin stellt in der Routine doUpdateDpin.c anhand der abgespeicherten Signalliste fest, ob die entsprechende Instanz des Modells im folgenden Programm-Block Befehle auszuführen hat. Ist dies der Fall, so werden in der Funktion updateDpin.c die im Datenfeld gespeicherten Werte ausgelesen und der Betriebsmodus des DPINs bestimmt. Es wird in diesem Fall zwischen den Modi „Drive“ und „Measure“ unterschieden. (Die Modi „Drive“ und „Compare“ aus dem Pattern werden in Zusammenarbeit mit dem timinggen.sin festgelegt.) Weiterhin muß zwischen Force Current Measure Voltage (FCMV) und Force Voltage Measure Current (FVMC) unterschieden werden. Die Daten für die Pegel und Modis werden an das DPIN-Modell übergeben. Erfolgt am DPIN eine Messung mit Hilfe der PMU wird die Funktion doMeasureDpin.c aufgerufen. Ihr wird der akutelle Meßwert (Strom oder Spannung) aus dem Saber-Modell mitgeteilt und im Datenfeld abgespeichert. Ein Flag wird gesetzt, welches dem Interface-Modell anzeigt, daß ein oder mehrere Meßwerte vorliegen, die dem SPACE übermittelt werden müssen.

Vom Modell timinggen.sin geht auf einer Pattern-Trigger-Leitung ein Event ein, sobald sich an einem der Pins ein Wert ändert. Daraufhin wird mit Hilfe der Routine readPatternVal.c der Modus (Drive oder Compare) für den entsprechenden Pin ermittelt und bei Bedarf (im Drive-Modus) die Pegel aus dem Datenfeld ausgelesen und an dpin.sin übergeben.

Im Compare-Fall werden die anliegenden Spannungspegel an die Funktion patternCompare.c übergeben. Darin werden die Meßwerte mit den Sollwerten verglichen.

Der Drive-Fall entspricht dem Force-Voltage-Modus. In der elektrischen Sektion des DPIN-Modells werden jetzt, je nach anliegendem Modus, die Ströme und Spannungen berechnet, die dem DUT zugeführt werden.

Ein Beispiel für die Auslagerung von physikalischem Verhalten in die abstrahierte modulare Beschreibung mit Hilfe von C-Routinen bietet die Aktive Last. Die Beschreibung dieser Implementierung findet sich im Anhang B.2.

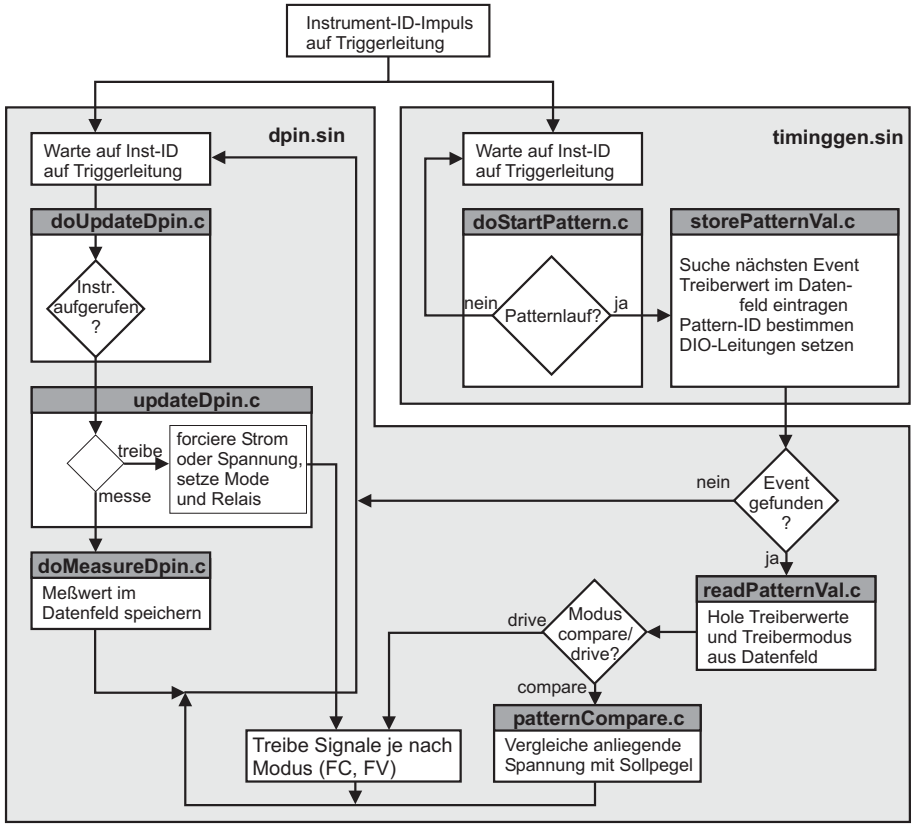



**Abbildung 4.20: Konzept des DPIN-Modells**

### 4.2.6 Ergebnisanalyse

Während der Testprogrammentwicklung analysiert der Testingenieur verschiedene Details innerhalb der Testschritte. Dazu sind verschiedene Genauigkeiten und unterschiedliche Ergebnisse erforderlich bzw. ausreichend. In diesem Abschnitt wird behandelt, welche Vorgehensweise bei der Testprogramm-Entwicklung welche Simulationsergebnisse benötigt.

#### 4.2.6.1 Anforderungen

**Einfache Setup-Überprüfung**

Bei der Simulation kompletter Testprogrammläufe will man den Übergang zwischen einzelnen Testschritten sowie das Zusammenspiel des letzten Testschrittes mit dem ersten beobachten. Hierbei spielen vor allem die Setups eine wesentliche Rolle, die ohne Konflikte ineinander greifen müssen. Es ist zu kontrollieren, ob bei den Setups der einzelnen Testschritte, die oftmals aufeinander aufbauen, alle Instrumente zugeschaltet bzw. angeschaltet wurden und somit die nachfolgende Messung nicht negativ beeinflussen. Um derartige Ergebnisse zu erlangen, müssen keine detaillierten Informationen während der Simulation generiert werden. Somit kann man abstrakte einfache Modelle verwenden, die eine hohe Simulationsgeschwindigkeit garantieren.

**Detailanalyse**

Bei der Analyse und dem Debugging einzelner Testschritte ist es häufig notwendig detaillierte Informationen aus der Simulation zu erhalten. Im Mittelpunkt bei derartigen Untersuchungen steht oftmals das genaue Verhalten eines Instruments unter kritischen Bedingungen, die Reaktion des Bausteins in zeitlicher und signaltechnischer Hinsicht oder das Verhalten der DIB-Verdrahtung mit dem Einfluß der zusätzlichen darauf befindlichen Bauelemente. Um Ergebnisse für diese Analysen mit Hilfe der Simulation zu erzielen, sind genaue Modelle der beteiligten Baugruppen erforderlich, die natürlich eine Erhöhung der Simulationszeit nach sich ziehen.

**Einschwingvorgänge**

Das Einschwingverhalten der Instrumente und der Ausgangstreiber des DUT sind hierbei zu analysieren. Hierzu sind sehr detaillierte Modelle notwendig, welche die Simulationszeit erheblich anheben. Deshalb sind hierfür geeignete Verfahren zur Zeiteinsparung zu finden. Mit der derzeitigen Rechenleistung ist die Gesamtsimulation aller dieser Vorgänge zur Zeit nicht praktikabel.

**Leitungssimulation**

Die Berechnung von Vorgängen wie Reflektionen, Störung, Dämpfung und Übersprechen einer Leitung sind aufwändig und nehmen in einer Simualtion sehr viel Zeit in Anspruch. Hierbei wird man wohl einzelne Kanäle auf dem DIB aus der Gesamtsimulation heraustrennen müssen und in eigenständiger Simulation betrachten.

#### 4.2.6.2 Lösung: Abstraktionsebenen

Wie man aufgrund der unterschiedlichen Anforderungen an die Ergebnisse sieht (Abschnitt 4.2.6.1), ist es unumgänglich, verschiedene Abstraktionsebenen oder Genauigkeitsstufen einzuführen, denn eine Simulationsebene kann nicht gleichzeitig schnell und sehr genau sein.

Die Arbeiten von Helmreich [HEL96-1] [HEL96-2] (siehe auch Abschnitt 3.2) haben gezeigt, daß eine detailreiche Simulation, bei der z.B. auf Effekte wie Noise und Jitter eingegangen wird, nicht in einer Gesamtsimulation mit allen Instrumenten zu bewältigen ist. Hierbei sind einzelne Komponenten aus dem Gesamtmodell herauszulösen, um sie in einer detailreichen Modellierung exakt zu simulieren. Genauigkeitsstufen oder Abstraktionsebenen erlauben es somit, die jeweils optimale Performance für die gewünschte Simulation zur Verfügung zu stellen.

Während der Entwicklung eines Testprogramms untersucht der Testingenieur in den einzelnen Testschritten verschiedene Details. Deshalb müssen die Genauigkeitsstufen bzw. Abstraktionsebenen ganz unterschiedliche Gesichtspunkte erfüllen, um beispielsweise die Untersuchung eines bestimmten Instrumentenverhaltens unter kritischen Bedingungen in einer Detailsimulation zu unterstützen. Eine derart detaillierte Simulation hat einen erhöhten Zeitbedarf, den ein Testingenieur jedoch akzeptieren wird, erhält er dafür eine Antwort auf die untersuchte Problematik. Im Gegensatz dazu steht ein Simulationslauf des gesamten Testprogramms oder mehrerer Testprogramm-Wiederholungen, bei der das Augenmerk auf dem Zusammenspiel zwischen den Testschritten und dem Setup der Instrumente liegt. Diese Untersuchung ist besonders wichtig, bevor ein Testprogramm für den Produktionsbetrieb freigegeben wird. Dafür ist allerdings eine erheblich ungenauere, dafür wesentlich schnellere Simulation notwendig.

Abbildung 4.21 gibt einen Überblick über die benötigten Abstraktions-Ebenen. Zu Beginn einer Testprogramm-Entwicklung muß ein Syntax-Check durchgeführt werden. Für eine derartige Überprüfung ist kein Simulator erforderlich.

Die nächste Stufe beinhaltet einen Emulator. Der Emulator bietet die Möglichkeit, das Testprogramm ohne jegliche Hardware ablaufen zu lassen. Sämtliche Komponenten der Software-Seite -

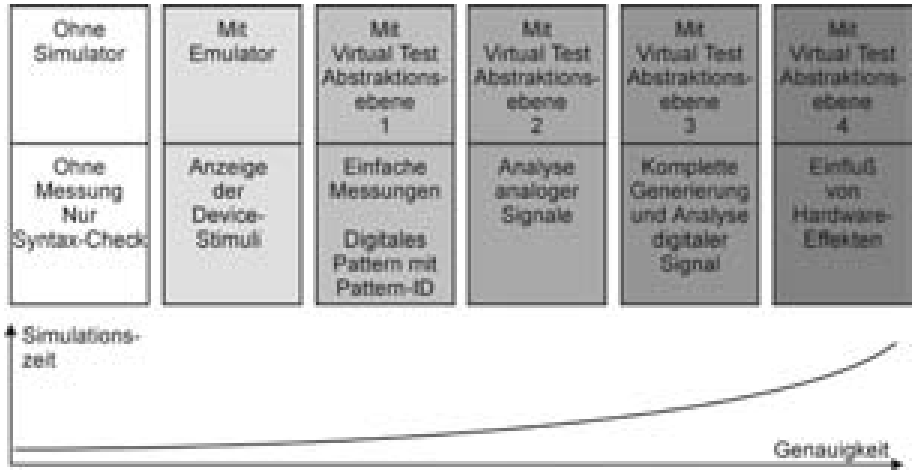



**Abbildung 4.21: Abstraktionsebenen**

also das gesamte Betriebssystem der Testmaschine - ist in diesen Ablauf eingebunden. Nur werden die Daten nicht an die Hardware weitergeleitet. Damit ist es dem Testingenieur möglich, die programmierten Werte und die Richtigkeit der Instrumenten-Parameter zu kontrollieren. Beide Ebenen liefern sehr schnell Ergebnisse, da keine Simulation benötigt wird.

Auf der **Abstraktions-Ebene 1** wird zum ersten mal der Simulator eingebunden. Auf dieser Ebene kann der Testingenieur einfache Untersuchungen durchführen und digitale Pattern mit einer Pattern-ID ausführen. Hierbei wird dem DUT eine ID übermittelt, die für jedes Pattern einzigartig ist. Eine vereinfachte Version des DUT-Modells erkennt diese ID und liefert das dazu passende Signal an seinen Ausgang. Dieses Vorgehen ist natürlich an keinem realen Tester möglich.

**Abstraktions-Ebene 2** erlaubt es analoge Signale zu analysieren. Die Stärke dieser Ebene liegt in der Generierung und Analyse analoger Signale.

Auf **Abstraktions-Ebene 3** können digitale Pattern komplett simuliert werden. Das erlaubt die Überwachung der Signalerzeugung und -erfassung digitaler Signale auf elektrischer Ebene.

**Abstraktions-Ebene 4** zielt auf die Analyse von hardwarenahen Effekten. Auf diesem sehr detaillierten Level ist es natürlich nicht möglich, das komplette Testprogramm und alle Instrumente mit ihren Verbindungen zu simulieren. Nur die zu untersuchenden Instrumente und Signal-Pfade werden extrahiert und auf Schaltungsebene mit nichtlinearen Differentialgleichungen simuliert.

Im Anhang B.3 wird am Beispiel der Behandlung von Pattern die Implementierung der Abstraktionsebenen erklärt.

## 4.3 Zusammenfassung der Implementierung

Im Rahmen dieser Arbeit wurden wesentliche Komponenten zum Aufbau einer virtuellen Testumgebung entwickelt. Dazu wurden zunächst die Anforderungen an einzelne Module der Umgebung definiert. Die für die entsprechenden Anforderungen entwickelten Konzepte wurden vorgestellt und exemplarisch an verschiedenen Beispielen implementiert. Tabelle 4.2 gibt einen Überblick über die Anforderungen und deren Realisierung.

| **Anforderung** | **Realisierung** |
|---|---|
| Bedienungsumgebung | SPACE |
| Simulationskonfigurator | Schematic Entry |
| Simulationssteuerung | zeitgenaue Synchronisation nebenläufiger Prozesse |
| Datenverwaltung | Bspl. Patternverwaltung<br>Status der Instrumente |
| Modulare Modellierung | Bspl. Module des DPIN (timinggen.sin, dpin.sin) |
| C-Routinen | Bspl. Aktive Last |
| Abstraktionsebenen | Bspl. Pattern |

**Tabelle 4.2: Stand der Implementierungen beim virtuellen Test**

# Kapitel 5

# Modellierungskonzept für die Entwicklung einer neuen Testsystemarchitektur

## 5.1 Anforderungen

Bei der Entwicklung neuer Komponenten für ein Testsystem bzw. bei der Entwicklung einer neuen Testsystem-Architektur steht nicht so sehr das elektrische Verhalten an der Schnittstelle vom ATE zum DIB und DUT im Vordergrund (siehe Kapitel 4), das Augenmerk richtet sich vielmehr auf interne Vorgänge und Engpässe innerhalb der Architektur. Aus diesem Grund werden hierbei zum Teil andere Simulationskonzepte zum Einsatz kommen. Es soll im Folgenden kurz aufgelistet werden, welche Gesichtspunkte bei der Entwicklung einer neuen Testsystemarchitektur in die Bewertung eingehen:

- Hardware
  - Engpässe in der Kommunikation
    - Leerzeiten und Belastungsprofile
    - Deadlocks
    - Überlastungen
    - Fehlerhafte Konfigurationen
  - Temperaturproblematiken
  - Timingunverträglichkeiten

  - Genauigkeitsanforderungen
  - Simulation der Kalibriertechnik
  - Selbsttest (on demand)
  - EMV
  - Robustheit
  - Verhaltenssicherheit
  - Notabschaltung
  - Genügend hohes Potenzial an Parallelisierbarkeit
  - Kostenfunktionen (automatische Generierung von Statistiken)

- Software
  - Plug&Play-Fähigkeit
  - Unterstützung von parallelen Abläufen
  - Shadowing
  - Modularität von Treibern
  - Resends
  - Überläufe

## 5.2 Konzept zur Realisierung der Modellierung neuer Testsystemarchitekturen

### 5.2.1 Hierarchie, Klassen und Versionen bei der Modellierung

Bei der Planung neuer modularer Testsysteme steht die hierarchische Struktur im Vordergrund. In einer Datenbank sind Teilmodelle spezieller Testerressourcen, sogenannte Modellkomponenten, abgelegt. Die Modellierung komplexer Testsysteme muß deshalb folgende Konzepte unterstützen [GENT03]:

- Hierarchische Modellierung: Vorhandene Komponenten werden zu einer neuen Struktur mit beliebig vielen Hierarchiestufen verknüpft. Auf diese Weise entsteht eine neue Komponente, die in der Modelldatenbank abgelegt wird und selbst wieder in andere Komponenten eingebaut werden kann.

- Klassenkonzept: Eine in der Modelldatenbank hinterlegte Komponente steht für eine gesamte Modellklasse zur Verfügung. Sie kann folglich in mehreren Ausprägungen in höhere Komponenten eingebaut sein.
- Versionenkonzept: Für jede Modellkomponente können mehrere Versionen angelegt sein, beispielsweise eine Version zur Grob- und eine zur Feinmodellierung. Die Versionen können im Gesamtmodell sehr leicht ausgetauscht werden.

Bei der Entwicklung eines neuen Testsystems ist es notwendig Modifikationen an den Baugruppen vorzunehmen. Auf diese Weise entstehen neue, an die Aufgaben angepaßte Modelle.

### 5.2.2 Datenbank

Eine Datenbank ist - wie aufgrund des vorangegangenen Abschnitts ersichtlich - zur Verwaltung der Modellkomponenten sowie der Versionen einzelner bzw. kombinierter Modelle notwendig.

Das Gesamtmodell soll sich aus Teilmodellen zusammensetzen, welche die physikalischen Testsystemmodule widerspiegeln. Eine Datenbank übernimmt die Verwaltung dieser Modelle. Welche Eigenschaften, Funktionalitäten, Parameter, Arten der Verknüpfungen diese Modelle dazu benötigen, wird im Folgenden festgelegt [GENT03] (Abbildung 5.1):

- Name [string]
- ID [integer]
- Typ [string]
- Inputs/Outputs (analog, digital, in, out, bidirektional)
- Abstraktionsebene [integer]
- Link auf Vaterressource (befindet sich eine Abstraktionsebene höher)
- Infos über Domänenmix (nötige Umrechnung zwischen zwei Ressourcen)

### 5.2.3 Ressourcen-Editor

Die Datenbank ist mit Modellen zu füttern. Diese Modelle müssen noch strenger einen Standard einhalten, als es beim virtuellen Test war. Dort war es aufgrund der historisch gewachsenen

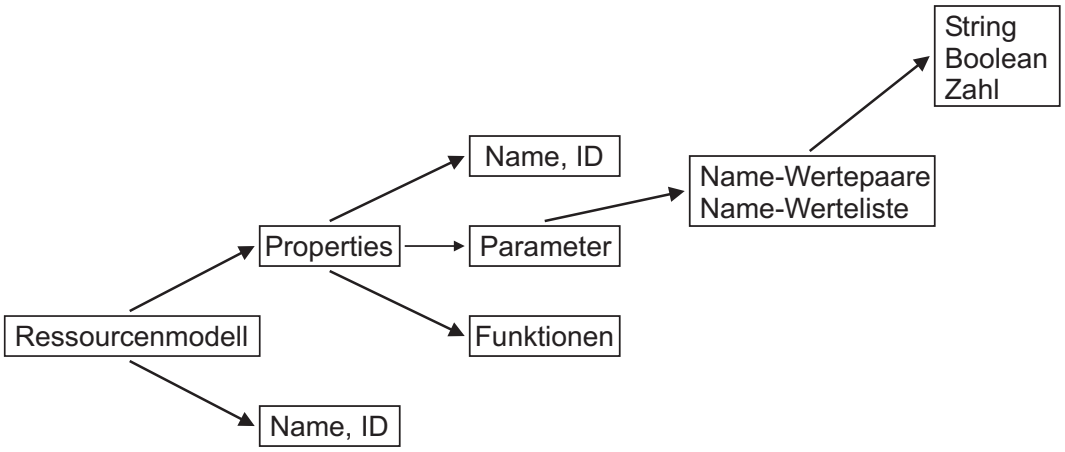



**Abbildung 5.1: Struktur innerhalb der Datenbank**

Strukturen und schlecht modellierbaren Architekturen oftmals nicht möglich, einen übergreifenden Standard anzuwenden. Bei der Entwicklung einer neuen Testsystem-Architektur besteht aber von vornherein die Möglichkeit, die kommenden Instrumente und Systeme auf optimale Modellierbarkeit hin zu entwickeln.

Um einen Standard für die Generierung der Ressourcen, die als Modelle für die Simulation dienen, einzuhalten und die Komplexität beherrschbar zu machen, wird ein Ressourcen-Editor vorgeschlagen. Dieser gibt mittels Maskeneingabe einen Standard für die Modelle vor. Die Vorgehensweise über eine derartige Maskeneingabe ermöglicht es dem Simulator vor der Simulation zu prüfen, ob die verlangten Properties, Parameter und Einstellungen als Daten hinterlegt und gültig sind. Beim Hinzufügen von Eigenschaften zu Modellen werden diese auf Konsistenz innerhalb einer Abstraktions-Ebene überprüft. Es muß ebenso möglich sein, innerhalb einer Abstraktions-Ebene Ressourcen zu schaffen und zu speichern, sofern eine „Vaterressource“ zu einem höheren Abstraktionsgrad existiert. Auf diese Weise werden die Ressourcen in eine Baumstruktur eingegliedert, über die Instrumente ausgewählt und verändert werden können (Abbildung 5.2).

Damit ergeben sich folgende Gesichtspunkte beim Entwurf an einen Ressourcen-Editor:

- Standardisierte Modellierung über Templates
- Modelle, die auf verschiedenen Abstraktionsebenen vorliegen
- Berücksichtigung der Eigenschaften aus höheren Ebenen
- Eingliederung der Modelle in die Datenbank
- Auflistung der Ressourcen zur Übersichtlichkeit in einer Baumstruktur

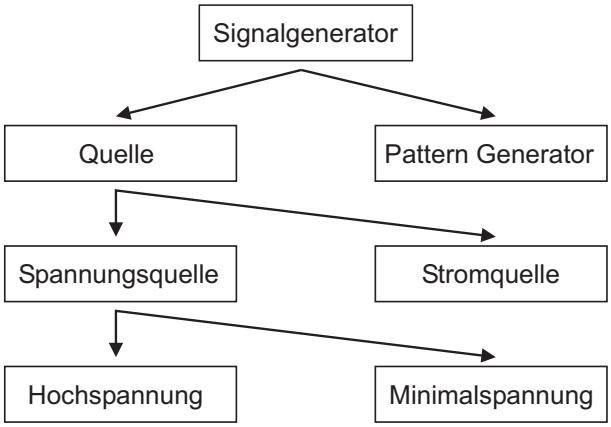



**Abbildung 5.2: Baumstruktur innerhalb des Ressourcen-Editors**

### 5.2.4 Beobachter

Es gibt unterschiedliche Ergebnisse, die man bei der Entwicklung einer neuen Testsystem-Architektur sehen und analysieren will. Diese Ergebnisse unterscheiden sich grundsätzlich von denen des virtuellen Tests, da hierbei interne Vorgänge analysiert werden, die beim virtuellen Test keine größere Rolle gespielt haben. Deshalb ist es notwendig sogenannte „Beobachter“ einzuführen (Abbildung 5.3), die interne - eigentlich verborgene - Vorgänge protokollieren und dem Entwicklungsteam zur Verfügung stellen. Im Software-Engineering heißt dieser Vorgang „Instrumentierung “. Diese Beobachter sind in die Modelle eingebunden. Sie stellen aber eine fest definierte Struktur dar. Die Ergebnisse werden in einer Protokolldatei abgelegt, welche nach der Simulation ausgewertet werden kann. Dazu sind verschiedene Funktionen vorzusehen, wie graphische Darstellung, Formeleditor, Kosten, Gewichtung usw.. Damit sind folgende Funktionen in die Beobachter zu implementieren:

- Spezifiziertes standardisiertes Format
- Anzahl der Aufrufe eines Moduls
- Zeitbelegung
- Kollisionen
- Requests
- Redundanzen
- Überläufe
- Data lost

- Resend

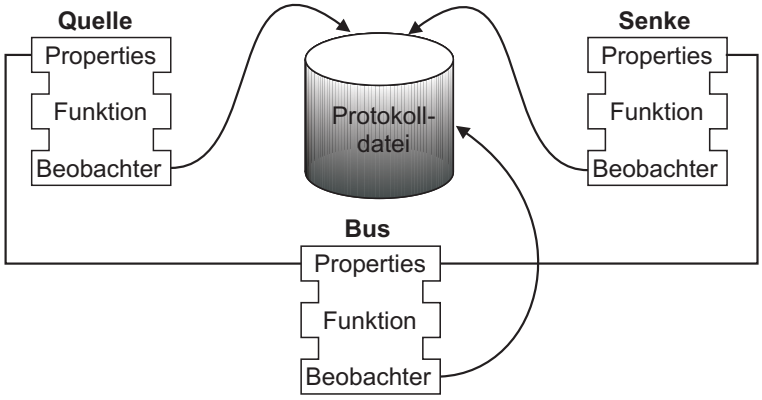



**Abbildung 5.3: Funktionsweise und Integration der Beobachter in den Modellen**

### 5.2.5 Schematic-Entry-Tool

Zur Eingabe der Konfiguration bietet sich an, eine graphische Oberfläche zu wählen. Das Schematic-Entry stellt das zentrale Element der Entwicklungsumgebung dar (Abbildung 5.4). Aus diesem Werkzeug kann alles gesteuert werden. Seine Hauptaufgabe liegt allerdings im platzieren und verbinden einzelner Ressourcen. Dazu werden ATE-Module aus einer Ressourcen-Liste ausgewählt, auf der Arbeitsfläche plaziert und miteinander verbunden. Nachdem eine mögliche Testsystem-Architektur derartig entworfen wurde, kann ein Schematic-Check erfolgen. Er überprüft die Konsistenz der Architektur, ob Anschlüsse fehlerhaft belegt oder ob Anschlüsse, die zur Funktion notwendig sind, nicht belegt sind.

An das Schematic-Entry-Tool werden folgende Anforderungen gestellt:

- Zentrale Steuerung aller wichtigen Programmteile
- Darstellung sämtlicher Testerressourcen in einer Liste (Baumstruktur)
- Schematic-Entry für die Ressource und deren Verknüpfung
- Schematic-Check
- Leichter Austausch der Einzelressourcen
- Unterstützung der physikalischen Architektur (z.B. Mainframe und Testkopf als oberste Komponenten)
- Option, ob Ressourcen eigene Controller besitzen (zentrale / dezentrale Steuerung)

### 5.2.6 Betriebssystem

Beim virtuellen Test war das Betriebssystem der Testmaschine vorgegeben und mußte verwendet werden, um eine gewohnte Arbeitsumgebung für den Testingenieur zu schaffen. Bei der Entwicklung einer neuen Testsystemarchitektur wird parallel zur Hardware eine neue Software entwickelt. Somit ist es auch hierbei möglich, eine neue Architektur aufzubauen und sich von evolutionären Altlasten bei der Entwicklung des Betriebssystems zu trennen. Das neue Betriebssystem soll die Anforderungen an moderne Testsysteme unterstützen. Beispielsweise soll künftig eine exakte Trennung von Setup, Messung und Setdown innerhalb eines Testschrittes erfolgen, um somit die Parallelisierung von Vorgängen zu unterstützen. Es sollten, im Hinblick auf ein modulares Testsystem, nur Treiber geladen werden, die vom Testprogramm benötigt werden. Diese Prozeduren sind zu automatisieren, so daß hier von einer „plug&play-Tauglichkeit“ des Systems gesprochen werden kann. Diese Fähigkeit muß ebenfalls die Kalibrierung und den Selbsttest umfassen.

Vom virtuellen Test kann man somit Ideen zur Kommunikation zwischen zwei Prozessen übernehmen, um durch vorhandenes Know-How schnell zum Erfolg zu kommen.

### 5.2.7 Struktur der Entwicklungsumgebung

Aufgrund dieser Einzelanforderungen und -konzepte ergibt sich folgende Struktur der Entwicklungsumgebung.

Das Schematic-Entry-Tool stellt den zentralen Teil der Software-Umgebung dar. Mit dieser Oberfläche kann die komplette Simulationsumgebung gesteuert werden (Abbildung 5.4).
Der Ressourcen-Editor unterstützt den Benutzer bei der Entwicklung neuer Testsystem-Ressourcen. Dieser Entwicklungsprozeß wird standardisiert, dadurch ist es möglich sämtliche Ressourcen in einer Datenbank zu verwalten und somit wiederum dem Schematic-Entry zur Verfügung zu stellen.
Im Schematic-Entry wird die Architektur des neuen ATE entwickelt und graphisch festgelegt. Nach erfolgreicher Eingabe muß das entworfene System simuliert werden. Dazu steuert dieses zentrale Element einen Simulator an, der die eigentliche Simulation durchführt. (Es ist möglich, daß mehrere Simulatoren zum Einsatz kommen, abhängig von der Abstraktionsebene und dem gewünschten Simulationsaspekt).
Das künftige Betriebssystem, in dem die Testprogramme eingebettet sind, kann ebenso vom Schematic-Entry gestartet werden.
Nach erfolgter Simulation liefern die Beobachter Daten über Engpässe oder sonstige Unzulänglichkeiten in der geplanten Architektur an das Schematic-Entry zurück, welches die aufgetretenen Probleme in der gezeichneten Architektur visualisiert.

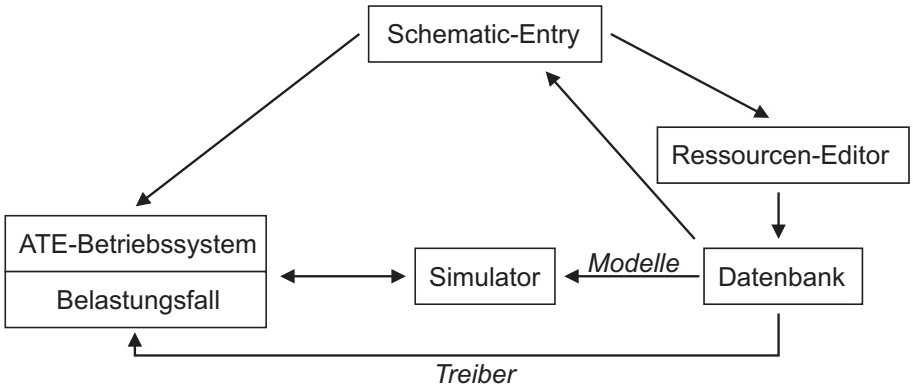



**Abbildung 5.4: Konzept einer Entwicklungsumgebung für neue Testsystemarchitekturen**

## 5.3 Zusammenfassung

Aufgrund der detaillierten Analyse der Hard- und Software existierender Testsysteme (Kapitel 2), der Analyse des Stands der Technik bei virtuellen Testsystemen (Kapitel 3) sowie der Erfahrung bei der Entwicklung von Testsystemmodellen (Kapitel 4) kam man zu dem Schluß, daß künftige Testsysteme auf einer neuen modularen Architektur basieren müssen. Diese bietet die Vorteile einer leichteren Modellierbarkeit, schnellerer Simulationszeiten und eine Vereinfachung der Programmierung.

Um sich von vorhandenen Altlasten komplett zu lösen wurde die Vision erarbeitet, eine derartige Architektur im virtuellen Raum zu entwickeln und zu bewerten. Dazu wurde ein Modellierungskonzept für die Entwicklung neuer Testsystemarchitekturen aufgestellt, das sich an den Gesichtspunkten zur Bewertung der Architekturvorschläge orientiert. Die notwendigen Module wurden konzeptionell vorgestellt und ihr Zusammenspiel in der Entwicklungsumgebung aufgezeigt.

# Kapitel 6

# Verifikation der Modelle

Die Verifikation der Instrumentenmodelle erfolgt unter anderem hinsichtlich der Funktionalität und Programmierbarkeit aus dem Testprogramm heraus, der Richtigkeit der Simulationsergebnisse sowie Aspekten der Simulationsgeschwindigkeit.

Die Überprüfung der Modelle wurde in verschiedenen Phasen durchgeführt.

- Verifikation während der Modellentwicklung anhand funktionierender Testprogramme und deren Meßwerte.
- Überprüfung der Meßwerte im direkten Vergleich zwischen Simulation und Realität.
- Systemtest im Rahmen einer Evaluierungswoche bei den Industriepartnern.

## 6.1 Vorhandene Testprogramme

Während der Entwicklung der Simulationsmodelle wurde fortwährend eine Überprüfung der implementierten Funktionalität durchgeführt. Diese basierte auf bestehenden Testprogrammen. Die Grenzwerte, die in diesen Testprogrammen enthalten sind, wurden bei Tests auf realen Testmaschinen verifiziert. Können diese in der Simulation ebenfalls erreicht werden, wurde die Funktionalität, die von einem Testprogramm-Befehl aufgerufen wurde, korrekt implementiert. Dieses Vorgehen mußte verständlicherweise mit mehreren Testprogrammen durchgeführt werden, um die Funktionalität allgemeingültig zu verifizieren. Hierzu standen verschiedene Programme der Partnerfirmen zur Verfügung. Die hierbei verwendeten Demonstratoren deckten unterschiedliche Bereiche der Applikationspalette ab. Zur Verfügung standen Testprogramme für:

- Soundkartenbaustein
- Auswerteeinheit für Knock-Sensoren
- Auswerteeinheit für Drehraten-Sensoren
- Leistungstreiber für $\mu$C-Interface
- Bausteine aus verschiedenen Schulungsprogrammen
- Selbsttestprogramme

## 6.2 Ausgewählte Testschritte des Audio Codecs

Im Folgenden können nur Beispiele und Screenshots angegeben werden, die mit dem Testprogramm für den Soundkartenbaustein generiert wurden. Für weitere Ergebnisse wird auf die Berichte der Industriepartner hingewiesen (Abschnitt 6.4).

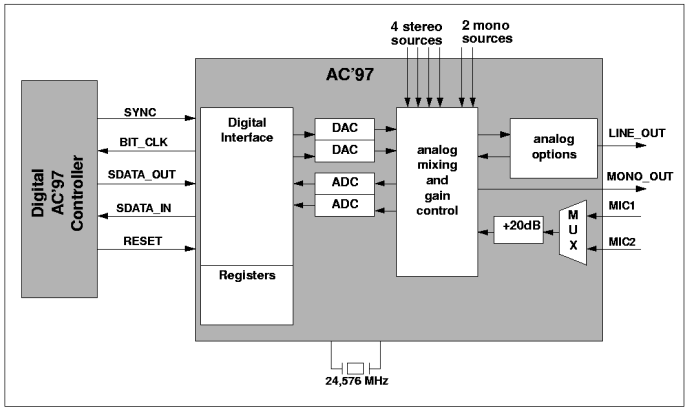



**Abbildung 6.1:** Blockschaltbild des AC97

Bei dem Soundkartenbaustein handelt es sich um den Audio Codec '97 (AC97). Der Baustein ermöglicht es, analoge Audiosignale verschiedenster Quellen an PC-Systeme anzuschließen. Gesteuert wird der IC von einem AC97 Digital Controller. Abbildung 6.1 zeigt die Funktionsblöcke des AC97 und die Verbindung zu seinem Master. Die beiden Digital-Analog-Konverter (DAC) mit 48 Kss unterstützen den Stereo-PCM-Kanal, der ein gemischtes Signal mehrerer digitaler Quellen enthalten kann, die im AC97 Digital Controller abgemischt wurden. Der Ausgang der DACs kann mit verschiedenen anderen analogen Quellen gemischt und dann an den Line-Out

geliefert werden. Für Telefone liefert der Mono-Ausgang entweder das Mikrophon-Signal oder ein Mono-Gemisch verschiedenster Quellen an ein Telefonsystem. Die Analog-Digital-Konverter (ADC) unterstützen zwei Kanäle mit 48Kss. Der Stereo-PCM-Eingang kann sämtliche Quellen aufnehmen. Damit unterstützt der AC97 folgende Quellen: digital PCM Ein- und Ausgang, CD, DVD, Mono-Mikrophon, Stereo-Line-In, Video und AUX. [INT02], [SIG98]

### 6.2.1 Kontakttest

Der erste Test ist in aller Regel ein Kontakttest. In diesem Beispiel wird an die Pins des Prüflings ein Strom von 100 $\mu$A angelegt. Wird die typische Diodenspannung gemessen ist sichergestellt, daß der Prüfling korrekt kontaktiert ist. In der Programmierung hat man die Möglichkeit, sich die Meßergebnisse in einem Ergebnisfenster darstellen zu lassen (Abbildung 6.2). In diesem Beispiel wurde Pin 5 nicht korrekt verbunden und besteht somit den Test nicht. Dieses Verhalten kann im Simulator mit Hilfe einer Scope-Funktion verifiziert und näher untersucht werden (Abbildung 6.3). Mit diesem Test wird die Baugruppe LPMU des DPIN überprüft.

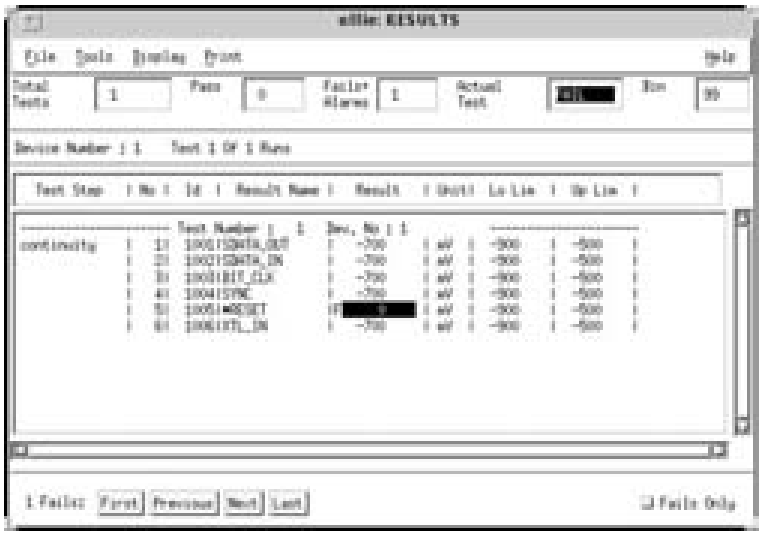


**Abbildung 6.2:** Ergebnisfenster: Kontakttest

### 6.2.2 Test der Ausgangstreiber

Um die Treiber der Ausgangsstufe zu testen, wird der Prüfling zunächst mit Hilfe eines digitalen Pattern in den Reset-Zustand versetzt. Danach sollte am Pin 8 des Prüflings konstant der Low-Pegel in Höhe von 0 V anliegen. Um dies zu überprüfen, wird an diesem Pin ein Strom von -5 mA, 0 mA und 5 mA eingeprägt und der Ausgangspegel überprüft. Bleibt er unterhalb von 0.5

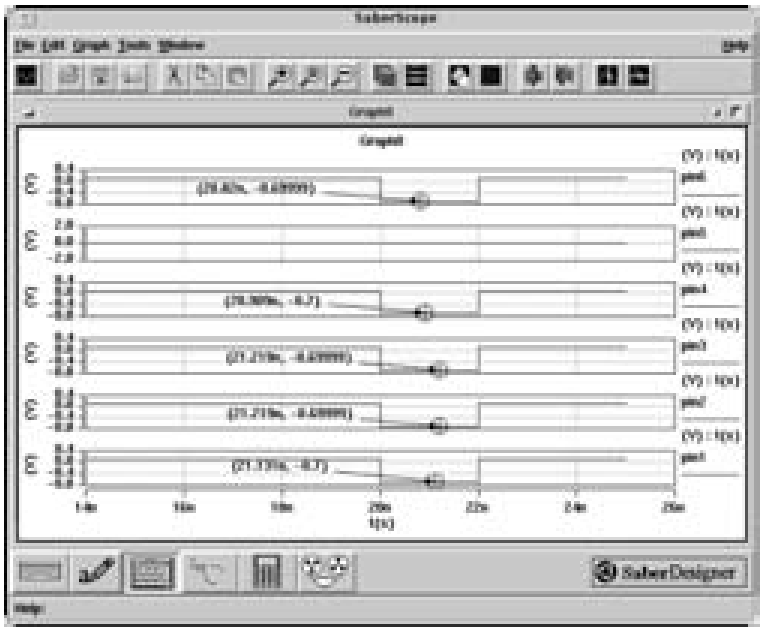


**Abbildung 6.3:** Ergebnisfenster: Kontakttest

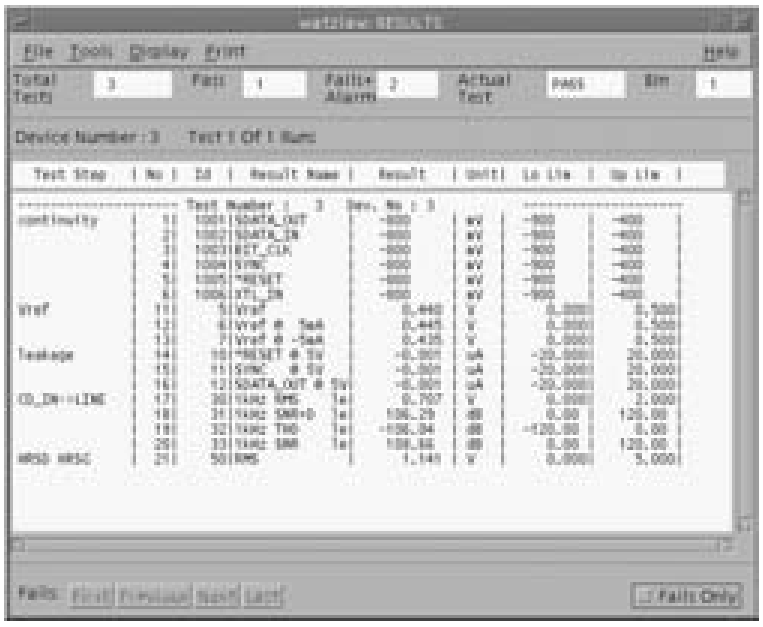


**Abbildung 6.4:** Result-Display für mehrere AC97-Testschritte

V hat der Chip diesen Test bestanden. (Abbildungen 6.4, 6.5) Bei diesem Test wird ebenfalls die LPMU überprüft, außerdem wird die Pattern-Behandlung mittels PDCL verifiziert.

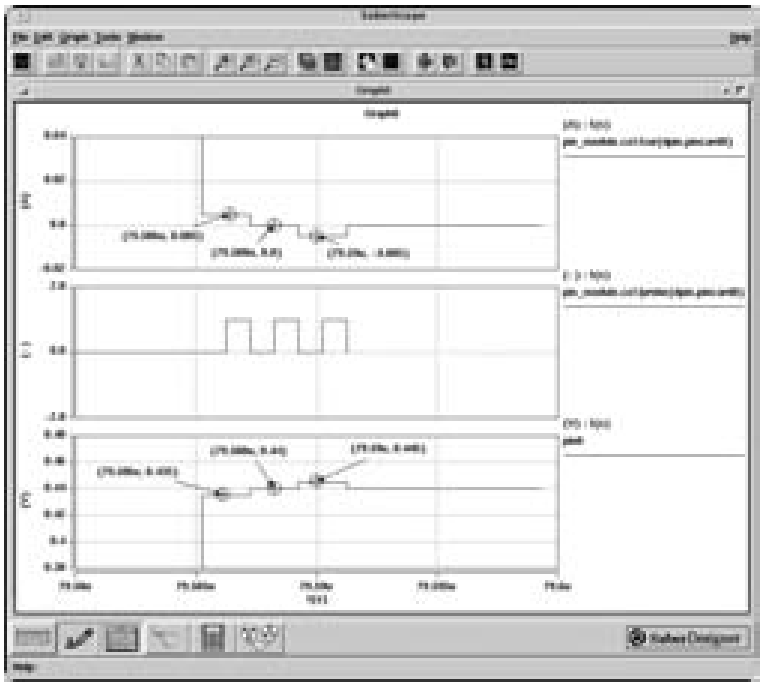


**Abbildung 6.5:** Scope: Test der Ausgangstreiber

### 6.2.3 Leckstromtest

Mit Hilfe der PMU wird an die digitalen Pins eine Spannung angelegt und der Leckstrom gemessen. Dieser Test verifiziert also ebenso die LPMU, allerdings diesmal in einem anderen Modus. (Abbildung 6.6)

### 6.2.4 CD-to-Line-Test

Beim diesem Test wird der analoge Übertragungsweg innerhalb des ICs von den Stereo-Eingängen über den internen Mischer und Verstärker hin zum Line-Ausgang überprüft. Im ATE-Modell verifiziert dieser Test die Funktionalität eines Generators (HRSG), eines Digitizers (HRSD) und der DSP-Einheit sowie das Zusammenspiel und die Synchronisierung der einzelnen Baugruppen. Dazu wird ein analoges Signal mit Hilfe des HRSD erzeugt und die analoge Antwort des Bausteins mit dem HRSD abgetastet und im Speicher abgelegt. DSP-Routinen verarbeiten diese Abtastwerte, so daß aussagekräftige Kenngrößen (RMS, SNR und THD) zum Vergleich herangezogen werden (Abbildung 6.4).

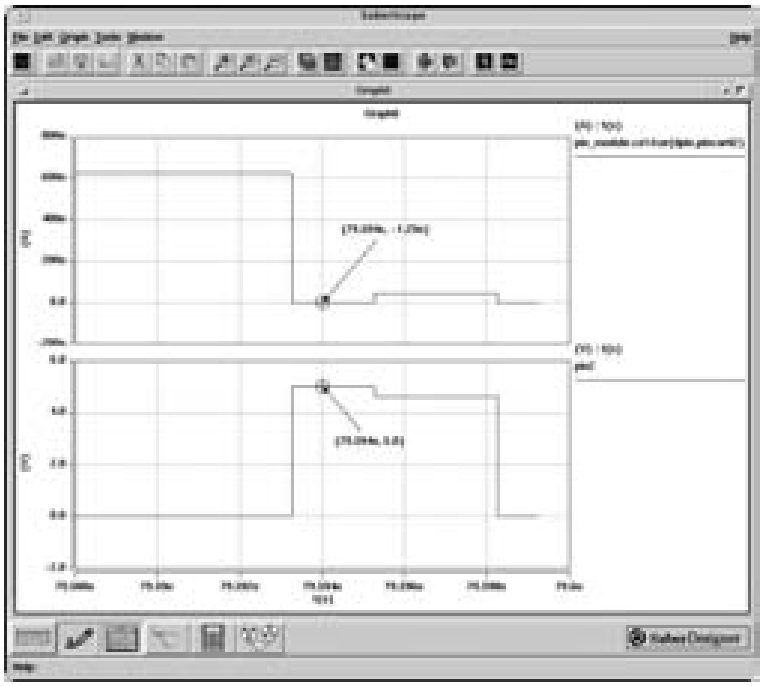


**Abbildung 6.6:** Scope: Leckstromtest

### 6.2.5 DAC-to-Line-Test

Der „DAC2Line-Test“ wurde bereits im Abschnitt B.3 erwähnt. Hierbei werden die Digital-Analog-Konverter getestet (Abbildung 6.7). In den Modellen des Testsystems kann damit die Pattern-Abarbeitung auf verschiedenen Abstraktionsstufen (Abbildungen B.26, B.28), das Abtasten analoger Signale, die Speicherverwaltung und das Job-Handling der DSP-Routinen und das synchronisierte Zusammenspiel der einzelnen Komponenten verifiziert werden.

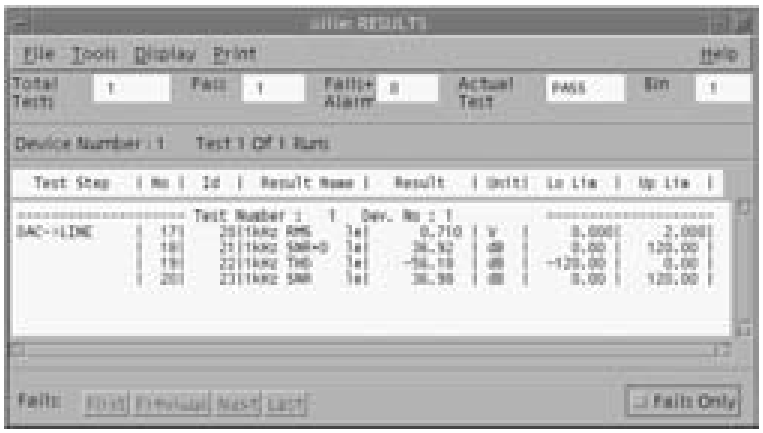



**Abbildung 6.7:** Result-Display: DAC2Line-Test

### 6.2.6 ICC-Test

Der ICC-Test ermittelt die Stromaufnahme des Prüflings in verschiedenen Betriebsmodi, nämlich dem Power-Down-Mode und dem normalen Betriebsmodus. Der AC97 wird mit Hilfe eines Pattern in den Reset-Zustand versetzt. Die Stromaufnahme wird gemessen. Anschließend wird der Chip durch ein digitales Pattern in den Power-Down-Modus gefahren und ebenfalls die Stromaufnahme ermittelt (Abbildungen 6.8, 6.9). Neben dem Patternhandling des DPIN bzw. der PDCL wird mit diesem Test die Strommessung mit Hilfe der Quellen verifiziert.

**Abbildung 6.8:** Result-Display: ICC-Test

## 6.3 Einschwingvorgänge Simulation und Realität

Ein direkter Vergleich zwischen Simulation und Realität konnte am Lehrstuhl für Rechnergestützten Schaltungsentwurf nur an Modellen des MPIN durchgeführt werden. Dieser digitale Pin ist sozusagen der Vorgänger des DPIN und von der Architektur unübersichtlicher und schwieriger zu modellieren als der DPIN (siehe Abschnitt 2.1.4). Diese Pinelektronik sollte nicht im Projekt implementiert werden, konnte aber durch die Verwandschaft zum DPIN parallel am Institut in Betrieb genommen werden. Die am LRS zur Verfügung stehende Testmaschine ist ebenfalls mit MPINs bestückt, so daß ein direkter Vergleich zwischen Simulation und Realität vorgenommen werden konnte.

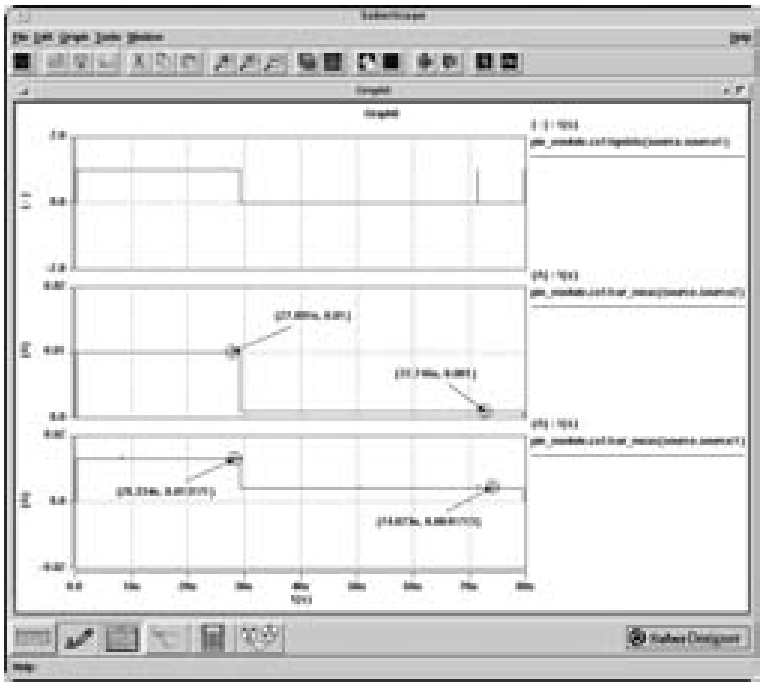


**Abbildung 6.9:** Result-Display: ICC-Test

### 6.3.1 Meßergebnisse

In diesem Abschnitt wird auf das Ausgangsverhalten der Testmaschine im statischen und im dynamischen Fall eingegangen. Dabei wird auch das Verhalten bei unterschiedlicher Belastung erläutert. Die Genauigkeit des Testsystems M3650/10 ist in der Spezifikation festgelegt, dabei sind folgende Spannungen und Ströme von Treiber und Sensor der MPIN spezifiziert (Tabelle 6.1).

| | Treiber | | Sensor | |
|---|---|---|---|---|
| Spannungen | VIL | VIH | VOL | VOH |
| Bereich | -15 V <= VIH | VIL < 15 V | -15 V bis 15 V | -15 V bis 15 V |
| Auflösung | 7,5 mV | 7,5 mV | 7,5 mV | 7,5 mV |
| Genauigkeit | 0,2% von FSR | 0,2% von FSR | 0,25% + 20 mV | 0,25% + 20 mV |
| max. Strom | 20 mA stat.<br>50 mA dyn. | 20 mA stat.<br>50 mA dyn. | - | - |
| Fall / Rise time | < 5 ns | < 5 ns | - | |
| Minimale Pulsbreite | 10.0 ns | - | - | |

**Tabelle 6.1:** Spezifikation für MPIN Treiber und Sensor

Die Spannungen und Ströme der Laststrombrücke des MPINs sind folgendermaßen festgelegt

(Tabelle 6.2):

| | Laststrombrücke | | |
|---|---|---|---|
| Spannungen und Ströme | VCOMM | IOL | IOH |
| Bereich | -12,5 V bis 12,5 V | 0,0 mA bis 50 mA | -50 mA bis 0,0 mA |
| Auflösung | 7,5 mV | 12,5 $\mu$A | -12,5 $\mu$A |

**Tabelle 6.2:** Spezifikation der MPIN Laststrombrücke

Im statischen Fall konnten die Spezifikationswerte eindeutig nachgewiesen werden. Von größerem Interesse war das Verhalten der Pins im dynamischen Fall, das heißt sobald eine digitale Signalfolge generiert wird. Um das Verhalten bei verschiedenen Frequenzen beurteilen zu können, wurde die Frequenz von wenigen kHz bis zu 24 MHz variiert, wobei der MPIN nur bis 20 MHz spezifiziert ist. Die Meßreihen wurden mit verschiedenen Konfigurationen aufgenommen (mit und ohne DIB / DUT). Bei den Aufnahmen ist immer exakt eine Periode dargestellt. Dadurch ergibt sich, daß bei Frequenzen bis zu einem MHz die Form des Rechtecksignals ideal aussieht. Das liegt jedoch an der zeitlichen Auflösung, denn hier sind exakt die gleichen Phänomene wie bei höheren Frequenzen vorhanden (Abbildungen 6.10 und 6.11).

Bei der Auswertung der Messergebnisse ohne DIB und DUT fällt die Verzerrung der Rechteckform bei höheren Frequenzen auf (Abbildungen 6.10). Diese Deformationen rühren von der Flankensteilheit des Signals her. Bei niedrigeren Frequenzen sind diese Deformationen im Anstieg ebenso vorhanden, aber aufgrund der Periodenlänge und der damit verbundenen zeitlichen Auflösung nicht auf Anhieb sichtbar (Abbildung 6.12).

Die gleiche Meßreihe wurde mit einem DIB und einem DUT (74LS245 - ein Octal-Bus-Transceiver) aufgenommen (Abbildungen 6.11).

Bei steilen Flanken kann man den Einfluß der Verbindungsleitung nicht vernachlässigen. Als Faustregel kann gelten, daß ein einfacher Verbindungsdraht nicht mehr ausreicht, wenn die Laufzeit auf dem Verbindungsdraht in die Größenordnung der Anstiegszeit der Schaltung kommt. Daraus ergibt sich für solche Verbindungen eine maximale Länge von ca. 10 cm je Nanosekunde Anstiegszeit. Wird sie überschritten, treten schwerwiegende Impulsverformungen, Reflexionen und mehr oder weniger gedämpfte Schwingungen auf. Diese Fehler kann man durch Leitungen mit definiertem Wellenwiderstand vermeiden (Koxialleitung, Streifenleitung), die man mit ihrem Wellenwiderstand abschließt. [TIE93]

Da die Verbindungen auf dem DIB für den Octal-Bus-Transceiver nur mit Wrap-Draht vorgenommen wurden, sind die Einflüsse auf das Signal relativ einfach erklärbar.

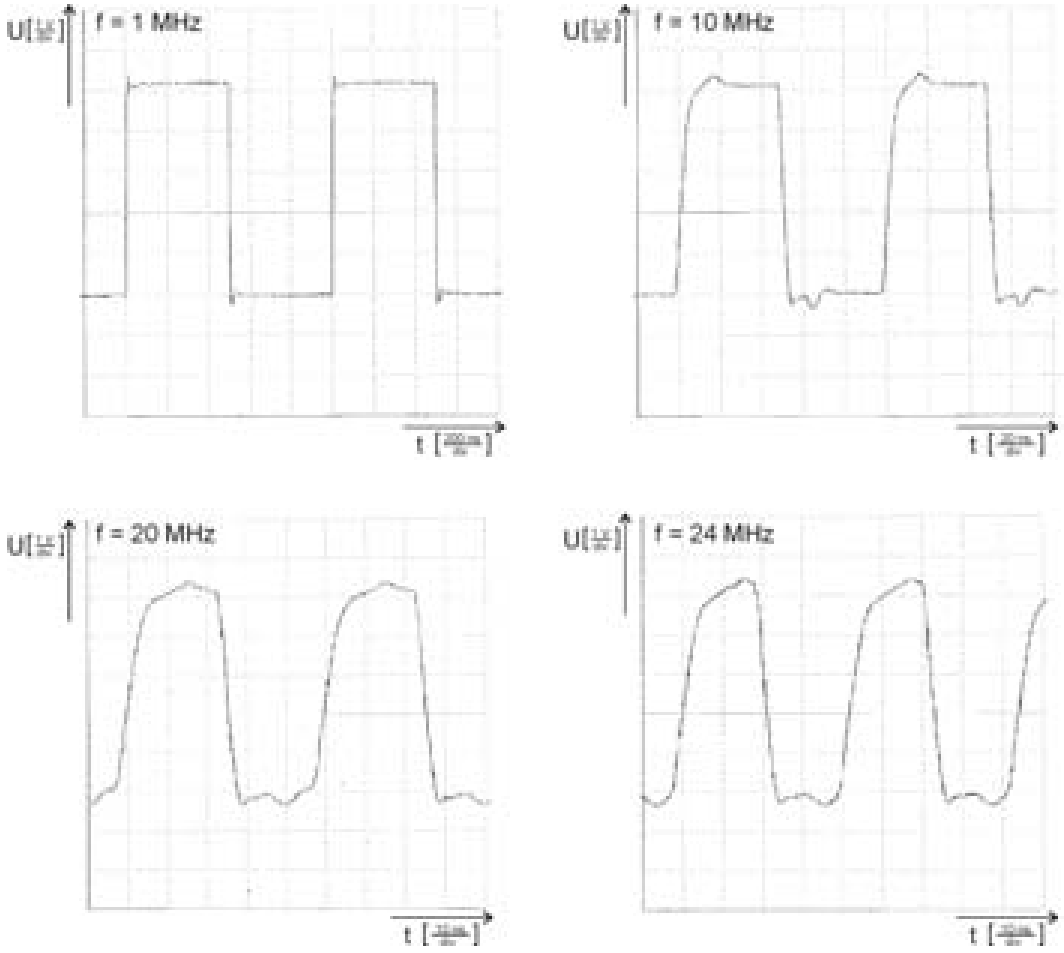



**Abbildung 6.10:** Meßergebnisse ohne DIB und DUT

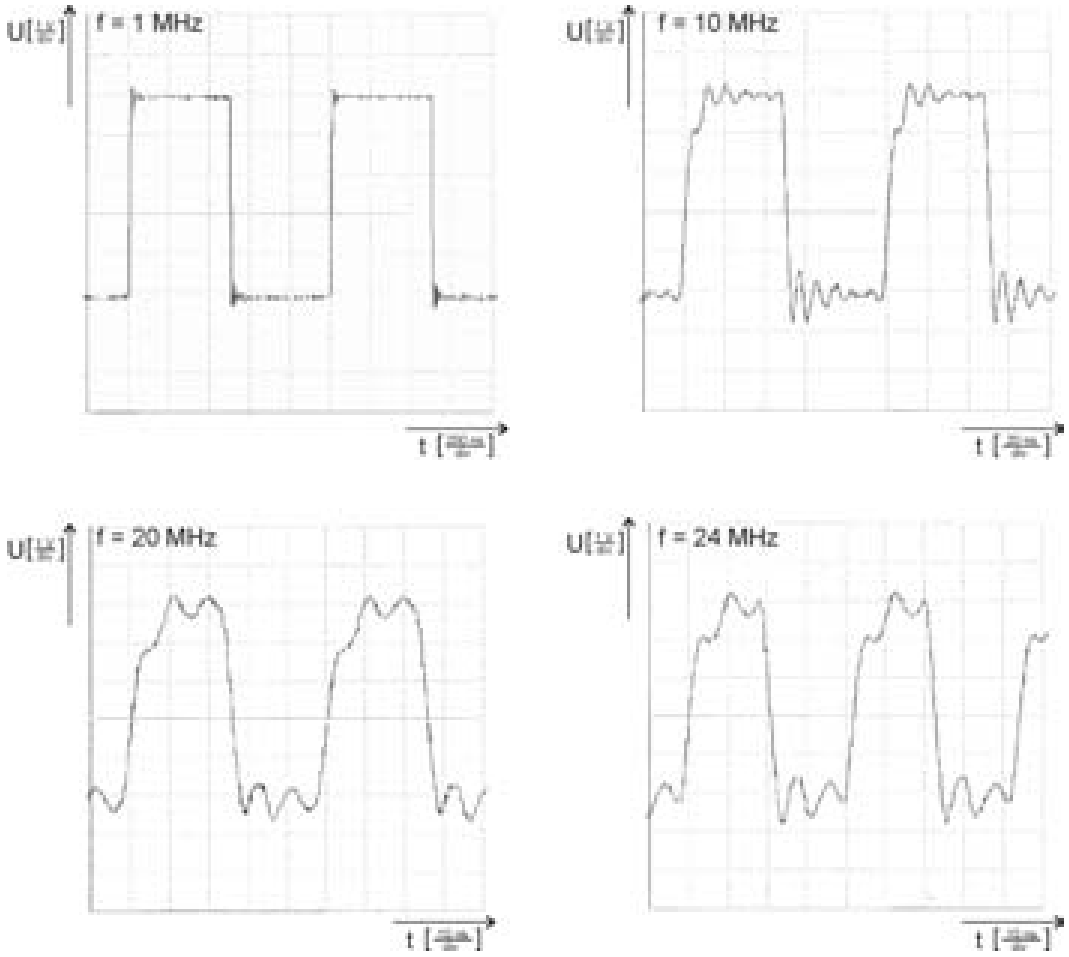



**Abbildung 6.11:** Meßergebnisse mit DIB und DUT

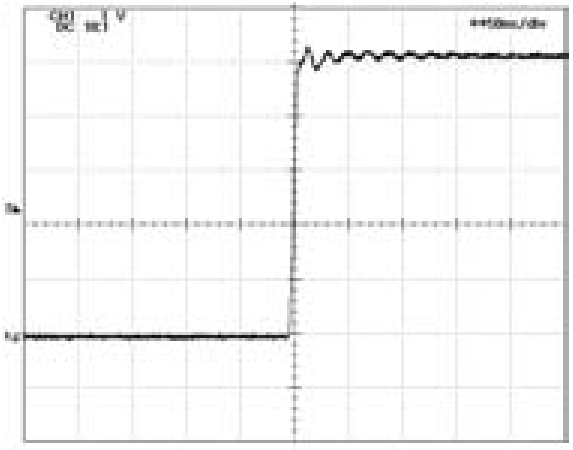


**Abbildung 6.12:** Überschwinger im Bereich der Flanken

### 6.3.2 Simulationsergebnisse

Im Abschnitt 4.2.6 wurde das Problem behandelt, daß während der Entwicklungsphase eines Testprogramms verschiedene Genauigkeitsstufen benötigt werden. Aufgrund des Abschnitts 6.3.1 ist offensichtlich, daß Einschwingvorgänge schnell zu Problemen bei der Programmierung eines Testprogrammes führen können, weil Wartezeiten bis zum stabilen Anliegen eines Signals nicht eingehalten wurden.

Der Vorgang, der hier als Einschwingvorgang bezeichnet wird, resultiert aus Effekten, die bei Leitungen auftreten können:

- Laufzeit
- Dispersion
- Reflexion
- elektromagnetische Kopplung
- Abstrahlung

Um die Einflüsse der physikalischen Effekte zu bestimmen, ist eine mathematische Beschreibung dieser Effekte notwendig. Der allgemeinste Fall einer mathematischen Beschreibung besteht in den Maxwellchen Gleichungen. Doch eine Lösung dieser Gleichungen führt selbst mit Hilfe von Vereinfachungen auf ein außerordentlich kompliziertes Randwertproblem, so daß kaum Hoffnung besteht, zu einer analytischen Lösung zu gelangen.
So bleibt im allgemeinen nur die Möglichkeit einer numerischen Lösung. Im Zeitbereich müssen hierbei allerdings vier Dimensionen berücksichtigt werden (Drei Dimensionen des Raumes und die Zeit). Dadurch wird auch hierbei der Rechenaufwand so groß, daß eine derartige Simulation nicht vertretbar ist.
Zu weniger aufwändigen Simulationen gelangt man durch Reduzierung der Dimensionen. Im Extremfall erhält man bei einer Simulation im Zeitbereich eine eindimensionale, nämlich rein zeitliche Beschreibung. Man spricht dann von einem sogenannten Netzwerkmodell. Der Vorteil liegt in der einfachen Simulation. Der Nachteil ist im Informationsverlust aufgrund der Dimensionsreduzierung zu sehen. Dieses Problem ist durch die mathematische Beschreibung mit Hilfe gewöhnlicher Differentialgleichungen erkennbar. Die vierdimensionale Beschreibung basiert im Gegensatz dazu auf partiellen Differentialgleichungen. Die Beschreibung erfolgt im vereinfachten Fall somit quasistationär. Eine Verbesserung dieses Verfahrens kann erreicht werden, indem die die Leitung beschreibenden Leitungselemente kaskadiert werden. Dieses Vorgehen läßt jedoch wiederum die Simulationszeit sehr stark ansteigen und leicht numerische Instabilitäten bei der

Simulation auftreten.
Als einfachstes Leitungselement wird somit das bekannte Netzwerk gemäß Abbildung 6.13 für eine Leitungselement der Länge $\Delta z$ . [GRA91]

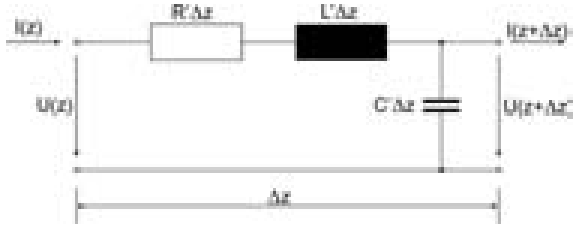


**Abbildung 6.13:** Netzwerk für ein Leitungselement

Bei einer Betrachtung der Einschwingvorgänge spielt die Zahl der Simulationspunkte eine wesentliche Rolle. In den Modellen des ATE werden die Zeitpunkte, an denen die Simulation ausgeführt werden soll, exakt angegeben, um eine möglichst hohe Simulationsgeschwindigkeit zu erreichen. Beim Einfügen der Einschwingvorgänge müssen zusätzliche Zeitpunkte eingefügt werden. Ein Beispiel mit einem Leitungselement soll zeigen, welche Fehlinterpretationen der Simulationsergebnisse möglich sind, wenn die Zeitintervalle zwischen den Simulationspunkten falsch gewählt werden.

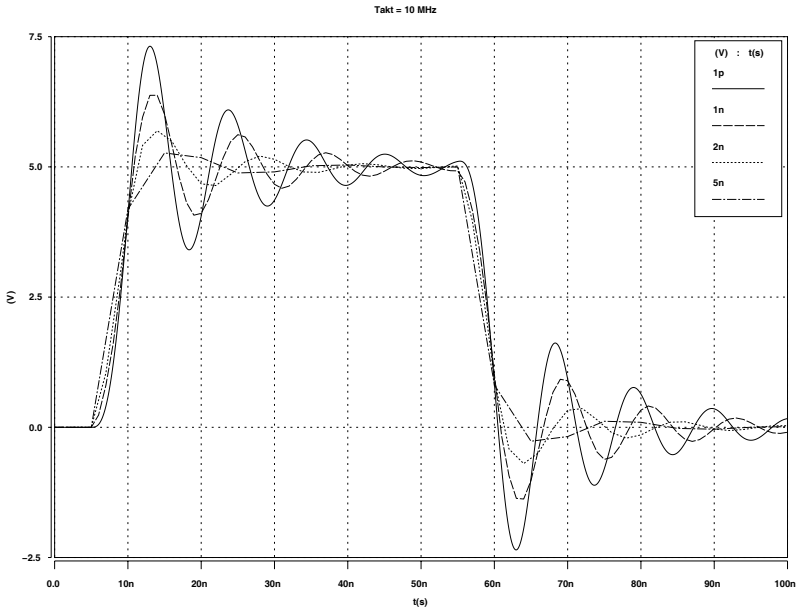



**Abbildung 6.14:** Simulation eines Leitungselements mit verschiedenen Simulationsschrittweiten

In Abbildung 6.14 sind vier Simulationen des gleichen Leitungselements mit unterschiedlicher Schrittweite zwischen Simulationszeitpunkten zu erkennen. Es sind gravierende Unterschiede beispielsweise in der Amplitudenhöhe auszumachen. Die fehlende Amplitudenhöhe resultiert

aus der Programmierung der Modelle. Hierbei wurde die Induktivität und Kapazität wie folgt modelliert:

$$u(t) = L\frac{di(t)}{dt} \tag{6.1}$$

$$i(t) = C\frac{du(t)}{dt} \tag{6.2}$$

In beiden Gleichungen fließt durch die Ableitung die Steigung der Kurve ein. Wird nun die Schrittweite dem Simulator fest vorgegeben, ist es möglich, daß die Kurve zu flach ansteigt (Abbildung 6.15). Dadurch kann der Maximalwert nicht erreicht werden und das Problem pflanzt sich durch die weiteren Berechnungen fort. Man sieht folglich eine viel schwächere Amplitude der Über- und Unterschwinger und dadurch eine schnellere Einschwingzeit (6.15). Diese Fehlinterpretation kann fatale Fehler bei der Programmierung der Wartezeiten für das stabile Anliegen der Signale nach sich ziehen.

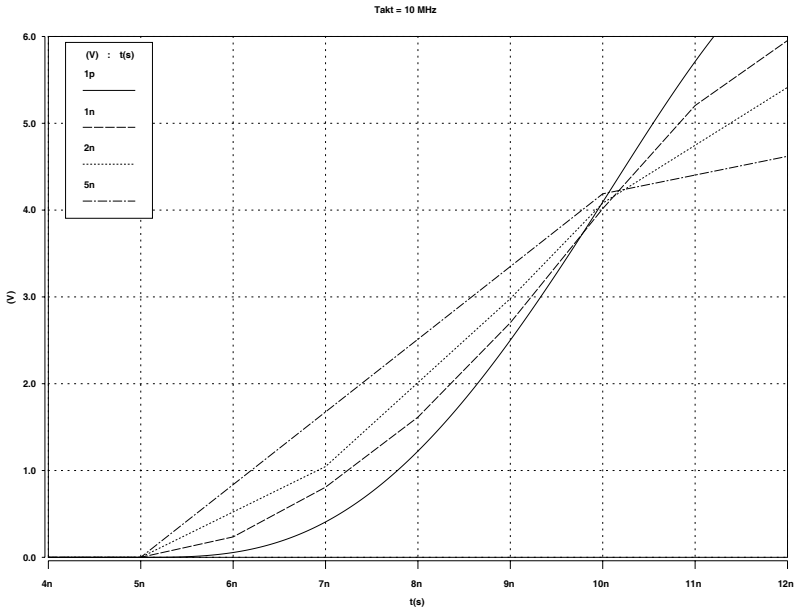



**Abbildung 6.15:** Simulation eines Leitungselements - Anstiegsphase

Die fehlerhaften Meßwerte, die sich aus der Simulation mit geringen Schrittweiten ergeben können, sind in Tabelle 6.3 zusammengefaßt. Als Referenz dient eine Simulation mit einer Schrittweite von 1ps.

Eine weitere Möglichkeit der Beschreibung des Einschwingverhaltens besteht in der Formulierung einer gedämpften Schwingung (Gleichung 6.3).

| Schrittweite [s] | gemessene Frequenz [MHz] | Abweichung [%] | gemessener Overshoot [V] | Abweichung [%] |
|---|---|---|---|---|
| 1p | 93,611 | | 8,4371 | |
| 1n | 85.413 | 8,76 | 7,4208 | 12,05 |
| 2n | 73,636 | 21,34 | 6,1865 | 26,68 |
| 5n | 49,183 | 47,46 | 5,2678 | 37,56 |

**Tabelle 6.3: Vergleich der Simulationen mit verschiedenen Schrittweiten**

$$U = \begin{cases} U_{high} + U_o \cdot \sin(2\pi \cdot Frequenz \cdot Zeit) \cdot e^{Dämpfung \cdot Zeit} \\ U_{low} - U_o \cdot \sin(2\pi \cdot Frequenz \cdot Zeit) \cdot e^{Dämpfung \cdot Zeit} \end{cases} \tag{6.3}$$

Hierin werden feste Annahmen über den Overshoot, die Einschwingfrequenz und die Dämpfung getroffen. Diese Annahmen lassen sich aus Messungen extrahieren, wobei der Faktor der Dämpfung natürlich vom Messaufbau abhängt und nach Gleichung 6.4 berechnet wurde.

$$\frac{u(t)}{u(t+nT)} = e^{Dämpfung \cdot Zeit} \text{ mit } n = 1, 2, 3 \ldots \tag{6.4}$$

Das Problem der fehlerhaften Steigung tritt bei der Näherung mittels der Gleichung 6.3 nicht auf, da hier die Werte an den einzelnen Zeitpunkten hart berechnet werden. Doch auch hier ist auf die passende Schrittweite zu achten, um das Signal eindeutig zu erkennen.

Bei beiden Verfahren müssen Werte für die Güte der Leitungsverbindungen auf dem DIB sowie der Steckverbindungen zwischen Testkopf, DIB und DUT bestimmt werden. Die Arbeiten [HEL99] und [HEL01] haben gezeigt, welcher Aufwand betrieben werden muß, um mit Hilfe von Messungen das Verhalten von Leitungen und Verbindungen zu erfassen, zu charakterisieren und daraus die richtigen Schlußfolgerungen für den idealen Aufbau von Verbindungen zwischen DUT und ATE zu ziehen. Diese Forderungen kann und soll der virtuelle Test nicht erfüllen.

Des weiteren haben die Veröffentlichungen [HEL96-1] und [HEL96-2] (siehe auch Abschnitt 3.2) bewiesen, daß es notwendig ist, einzelne Leitungen aus dem Gesamtgebilde des virtuellen Tests herauszulösen, um derart problematische Verhalten wie Einschwingen, Jitter oder Noise zu untersuchen. Das hier vorgestellte Konzept der virtuellen Tests hat dieses Vorgehen in der Definition der Abstraktionsebenen berücksichtigt (siehe Abschnitt 4.2.6.2).

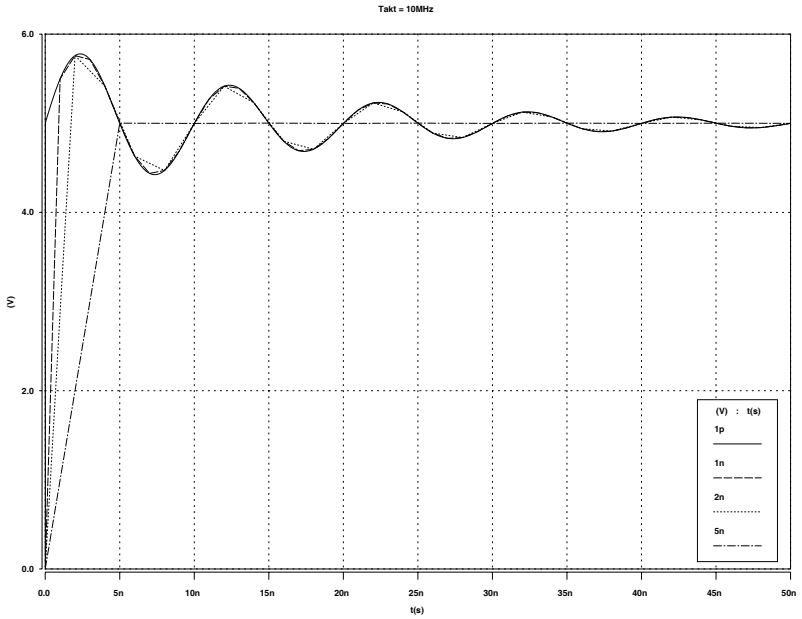



**Abbildung 6.16:** Simulation einer gedämpften Schwingung mit verschiedenen Simulationsschrittweiten

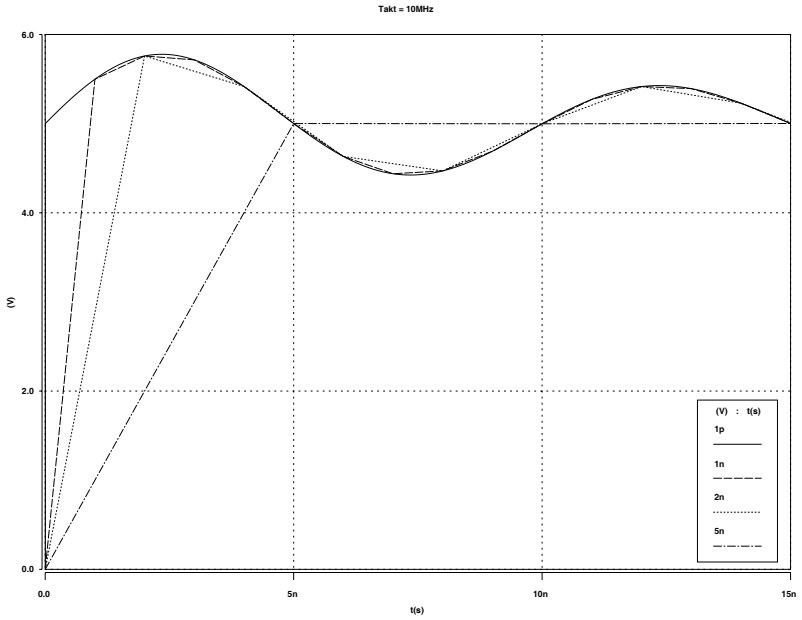



**Abbildung 6.17:** Simulation einer gedämpften Schwingung - Anstiegsphase

## 6.4 Evaluierung

Während verschiedener Evaluierungs- und Betatestphasen bei den Industriepartnern wurde die virtuelle Testumgebung von industriellen Anwendern überprüft.

### 6.4.1 Evaluierungswoche

Im Verlauf einer Woche wurde von den Projektpartnern eine Evaluierung der virtuellen Testumgebung mit folgenden Ergebnissen durchgeführt. [TEN02]

Die Umgebung war funktional und enthielt zum Zeitpunkt der Evaluierung keine gravierenden Mängel. Es konnten mit Hilfe zweier Demonstratoren (Soundkartenbaustein und Auswertechip eines Klopfsensors) vielfältige Tests erfolgreich durchgeführt werden.

Ein lauffähiges Testprogramm eines neuen Demonstrators (Auswerte-Einheit für einen Drehratensensor) wurde inklusive DUT- und Loadboardmodell während der Evaluierungswoche entwickelt. Es konnte in der virtuellen Testumgebung überprüft und fehlerbereinigt werden.

Die verwendeten DUT-Modelle für die drei Demonstratoren zeigten verschiedene Abstraktionsstufen, von der ganz einfachen Wertetabelle bis zu vereinfachten Übertragungsfunktionen und gemischten Modellen. Dabei konnte gezeigt werden, daß die entwickelte Modellierungsmethodik ein sehr brauchbarer Ansatz ist, der sowohl genügende Genauigkeit als auch sehr gute Simulationsgeschwindigkeit ermöglicht.

Die Benutzung der virtuellen Umgebung VSPACE ist ebenso einfach. Es sind keine besonderen Erfahrungen im Umgang mit einem Simulator erforderlich. Besondere Erfahrungen mit dem Simulator können jedoch die Effizienz der Benutzung entscheidend steigern, da mit Hilfe der Simulationsumgebung sowohl der Tester als auch die Umgebung (DUT und Loadboard) transparent gemacht wird. Es kann mehr beobachtet werden als es an der realen Hardware möglich ist. Damit wird die Aufdeckung von Fehlern oder Unzulänglichkeiten entscheidend verbessert. Dieses wurde bei der Testvorbereitung des dritten Demonstrators bereits positiv vermerkt.

Die Testabteilung sieht auf Grund der Ergebnisse der Evaluierung deutliche Vorteile bei der Benutzung der virtuellen Testumgebung während der Testprogrammentwicklung und sieht die Notwendigkeit, die virtuelle Testumgebung schnellstmöglich als Werkzeug zu etablieren.

### 6.4.2 $\beta$-Test

Ein weiterer Halbleiterhersteller hat an einer Testphase der virtuellen Testumgebung teilgenommen und dabei folgende Erfahrungen gemacht. [ANT02]

Während der Testphase dieser neuen Umgebung stellte die Verifikation des Testprogramms in der VT-Umgebung eine schwierige Aufgabe dar. Lieferte ein Testschritt nicht den gewünschten Wert, kann das Problem auf Fehlern in den DUT- und DIB-Modellen, der Prüfvorschrift, dem Testprogramm oder den Instrumentenmodellen beruhen. Daher war in dieser Phase eine enge Zusammenarbeit aller Beteiligter sehr wichtig.

Bei der Frage nach der Effizienz des virtuellen Tests stellte sich für den Halbleiterhersteller auch die Frage nach den Simulationszeiten. Als Anhaltspunkt konnte festgestellt werden, daß der Test des ersten Demonstrators (einem intelligenten Leistungstreiber mit $\mu$C-Interface sowie 4 High- und Lowside Treibern) mit 150 Testschritten einschließlich umfassender dynamischer Tests ca. 90 Minuten auf einer Sun Ultra-60 Workstation benötigte. Ohne die dynamischen Tests (110 Testschritte) betrug die Simulationszeit 20 Minuten. Ein Durchlauf des zweiten Demonstrators, der mit Hilfe digitaler Steuersignale zwei Leistungshalbbrücken ansteuert, mit 96 Testschritten (vornehmlich Open-Short-, ESD- und Leckstrom-Tests) in der gleichen Umgebung benötigte lediglich 3 Minuten.

Die Frage, welche ein Industrieunternehmen in erster Linie interessiert, ist die Frage nach der Einsparung. So konnte im Falle des ersten Demonstrators eine Verkürzung der Inbetriebnahme am realen Testsystem auf drei Tage erzielt werden. Es mußten lediglich Wartezeiten in das Testprogramm aufgenommen werden, die sich mit dem Einschwingverhalten des Testers begründen und Feinabstimmungen bei rein digitalen Tests vorgenommen werden. Diese Ergebnisse wurden vom zweiten Demonstrator bestätigt. Durchschnittlich beträgt die Debugging-Phase am realen Testprogramm, ohne vorhergehenden Virtuellen Test, ca. 70% - 80% der kompletten Zeit, die für ein Testprogramm benötigt wird. Diese Werte dürfen jedoch nicht als genereller Anhaltswert betrachtet werden, da sie von Faktoren wie Struktur bzw. Funktion des DUT und Testabdeckung im Virtuellen Test stark abhängig sind.

In der Evaluierung konnte gezeigt werden, daß deutlich früher mit der Verifikation des Testprogramms, des DUTs und des Loadboards begonnen werden konnte, typische Probleme bereits im Vorfeld erkannt wurden und daß die Debugging-Phase am realen Testsystem erheblich verkürzt wurde. Ferner trägt die Arbeit in der Virtual-Test-Umgebung zum Schaltungsverständnis des Testingenieurs bei.

## 6.5 Auswertung

Aus den vorangegangenen Betrachtungen ist ersichtlich, daß zur Beseitigung von Fehlern in Testprogrammen eine Simulation notwendig ist, die in der Ausführungsgeschwindigkeit zügig arbeitet. Die Simulation von aufwändigen Vorgängen wie Einschwingverhalten ist aufgrund der stark anwachsenden Anzahl von Simulationszeitpunkten, der dadurch rapide ansteigenden Simulationszeit sowie der komplexen Bestimmung von Leitungsparametern für die obersten Ebenen des virtuellen Tests nicht zu empfehlen.

Es zeigt sich, daß eine Verifikation des Testprogramms im virtuellen Test einen Zeitvorteil durch die Verlagerung des Debugging-Prozesses bringen kann.

Der Testingenieur ist nicht auf die Verfügbarkeit einer Produktionstestmaschine angewiesen.

Bei passender Auswahl der Abstraktionsebene, auf der simuliert wird, liegen die Simulationszeiten im Minuten-Bereich.

Die Simulation der gesamten Umgebung eröffnet dem Testingenieur die Analyse von Signalen, die er in der Realität nur unter Zuhilfenahme von Oszilloskopen und geeigneten Zusatzmessungen an der Hardware erzielen kann. Beispielsweise wird man in der Realität äußerst selten den Stromverlauf an einem Pin erfassen können, denn hierzu muß entweder die Leiterbahn aufgetrennt werden oder eine Stromzange zum Einsatz kommen. Diesen Aufwand muß man im virtuellen Test nicht betreiben, denn diese Informationen stehen dem Anwender sofort zur Verfügung. Durch Messungen an der realen Hardware kann es ebenso vorkommen, daß zusätzliche Fehler beispielsweise durch Meßköpfe eingeschleust werden, die dann Fehlinterpretationen erlauben.

# Kapitel 7

# Zusammenfassung und Ausblick

## 7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Umgebung für den virtuellen Test aufgebaut. Basierend auf einer vorangegangenen Arbeit [GOL98-1] wurde das System dahingehend erweitert und verbessert, daß es bei Industriepartnern mehr Akzeptanz fand und erste Evaluierungen erfolgreich abschloß.

Es wurde eine Kommunikation zwischen nebenläufigen Prozessen entwickelt, die beide Module zeitgenau und netzwerkweit miteinander synchronisiert, obwohl aufgrund der unterschiedlichen Ausführungszeiten die Zeitbasen auseinanderlaufen.

Am Beispiel digitaler Pins wurden Probleme beim Abarbeiten digitaler Signalfolgen, die als echte Strompegel zur Verfügung stehen sollten, eingehend behandelt. Es wurde beschrieben, wie die vorhandene Komplexität und Flexibilität durch Abstrahierungen modellierbar gemacht wurde.

Die vorgestellte Variante, ein bestehendes Testsystem zu modellieren, um damit ein Testprogramm - vor der Verfügbarkeit des ersten Siliziums - zu verifizieren, stellt eine erste Anwendungsmöglichkeit dieser Methode dar.

In Zukunft können damit jedoch nicht nur Testprogramme vorweg verifiziert werden, es ist auch möglich, neue Testsysteme im virtuellen Raum zu entwerfen. Weiterhin ist es damit möglich, die Leistungsfähigkeit applikationsspezifischer Testsysteme auf die jeweilige Applikation hin auszuloten.

Im Rahmen dieser Arbeit wurde ferner viel Wert auf die Analyse bestehender Hardwarekomponenten und ihrer Nachfolgeprodukte gelegt. Es zeigte sich, daß einige Neuentwicklungen nicht

nur Verbesserungen bringen. Zum Teil war die Komplexität der Instrumente nicht mehr in Software und damit in der Ansteuerung der Module nachzubilden. Ferner konnte festgestellt werden, daß die untersuchten Ressourcen heutiger Testsysteme nur unter großem Aufwand zu modellieren sind.

Um den Forderungen der ITRS nach flexiblen, modularen und dennoch komplexen Testsystemen Rechnung zu tragen ist es unerläßlich, künftige ATEs in einer Simulationsumgebung zu entwickeln, um somit zu einer ausführbaren Spezifikation für Testsysteme zu gelangen. Dieses Vorgehen ist jedoch nur dann möglich, wenn die zu entwerfende Hardware leicht modellierbar ist, was auch wieder dem virtuellen Test zu gute kommt. Konzepte für dieses Vorgehen wurden vorgestellt.

## 7.2 Ausblick

Die Analyse der Hardware hat gezeigt, daß heutige Testsysteme und deren Instrumente derartig komplex sind, daß ihre komplette Funktionalität nur schwer in Software nachzubilden und nur unter großem Aufwand in Simulationsmodellen erfaßbar ist. Um eine möglichst effiziente Testprogrammgenerierung zu erreichen, sollte die Programmierung mit einer High-Level-Sprache erfolgen, wobei allerdings die Feinheiten der Hardware nicht angesprochen werden können. Deshalb ist es notwendig die Hardware zu entfeinern. Hierzu sind Konzepte für den Aufbau modularer, plug&play-fähiger Testsysteme zu entwerfen. Dies kann mit Hilfe eines noch zu implementierenden Konzepts zur Entwicklung neuer Testsystem-Architekturen erfolgen (siehe Abschnitt 5). Bei einem derartigen entfeinerten System sind dann Schnittstellen zwischen dem DUT und dem ATE zu definieren. Hier spielen Aufgaben aus dem „Test Resource Partitioning“ eine essentielle Rolle. Die Aufteilung von Testmitteln zwischen dem Testsystem und dem Prüfling muß in Zukunft automatisiert erfolgen.

Mit so konzipierten Systemen ist es möglich, die Testprogramm-Entwicklungszeit zu reduzieren, da auf einer abstrakteren Ebene programmiert werden kann. Außerdem wird die Modellierung der Instrumente vereinfacht, da die Instrumente sowie die Architektur des Testsystems bereits bei der Entwicklung simulationsfreundlich entworfen wurden.

# Anhang A

# Software-Architektur

## A.1 Software-Hierarchie

Im Folgenden soll am Beispiel der SZ M3650 der hierarchische Aufbau der Software-Architektur analysiert werden.

### A.1.1 Basic-Instrumente

Basic Instrumente bilden die tiefste Programmierebene. Auf diesem Niveau sind alle Einstellungen, Verbindungen zwischen verschiedenen Instrumenten und Ressourcen-Verteilungen von Hand vorzunehmen, deshalb empfiehlt der Testsystemhersteller diese Ebene nicht.

Ein Beispiel soll die verschiedenen Ebenen im Folgenden veranschaulichen. Das Testsystem M3650 verfügt über mehrere Quellen (Tabelle A.1), die für unterschiedliche Aufgabenbereiche konzipiert sind. Diese Instrumente können einzeln als Basic-Instrumente programmiert werden. Dazu ist das jeweilige Instrument auszuwählen und seine Parameter zu programmieren. Ferner muß das Signal von der Quelle bis hin zum DUT über diverse Relais und Matrizen von Hand verschaltet werden. Das Beispiel in Abbildung A.1 zeigt dieses Vorgehen.

### A.1.2 Master-Instrumente

Auf der Ebene der Master-Instrumente werden mehrere Basic-Instrumente zu einem übergeordneten Instrument gruppiert. Das heißt Hardware, die in verschiedenen Komponenten implementiert

| Instrument Name | Force Range | Clamp Range | programmable clamp? |
|---|---|---|---|
| VS1 .. VS6 | -16 .. +16 V | +/-219 mA | nein |
| I_SOURCE | +/-250 mA | +/-13 V | nein |
| I_LEAK | +/-10 mA | +/-51 V | nein |
| VI1 .. VI16 | +/-52 V | +/-500 mA | ja |
| VIS1 | +/-100 mA | +/- 30 V | ja |
| VIS2 | +/-200 mA | +/-100 V | ja |
| VIS21 | +/-200 mA | +/-150 V | ja |
| VIS3 | +/-300 A | +/-30 V | ja |
| VIS5 | +/-1.0 A | +/-2000 V | ja |
| VIS51 | +/-5.0 A | +/-51 V | ja |
| VIS53 | +/-20.0 A | +/-25 V | ja |
| VIS54 | +/-2.5 A | +100 V / -10 V | ja |
| VIS8 | +/-55.0 A | +120 V / -10 V | ja |

**Tabelle A.1: Verfügbare Quellen im Testsystem M3650**

```
CTRL VS_IF
    DO_FOR_VS        VS1_LINE  /* force and sense lines of the VS1 */
    CONNECT_SENSE    CB        /* are connected to the CB          */
    CONNECT_FORCE    CB
END;
CTRL VS1
    SET_OUTPUT_RELAY TRUE   /* The output relays are closed                    */
    SET_HI_CAP       TRUE   /* allwos capacitors larger than 5 uF at the output */
    SET_VOLTAGE      5.00 V
END;
```

**Abbildung A.1: Programmierung über Basic-Instrumente**

ist, wird syntaktisch sinnvoll zu einem Überinstrument zusammengefaßt.
In unserem Beispiel heißt das, daß die in der Tabelle A.1 aufgeführten Quellen ebenso über das Master-Instrument SOURCES programmiert werden können. Das Master-Instrument SOURCES beinhaltet aber ebenso die Steuerung folgender Interfaces:

- AI_IF (Analog Instruments Interface) verbindet verschiedene Meßgeräte mit dem Configuration Board
- VS_IF (Voltage Source Interface) verbindet die Force- und Sense-Leitungen der Spannungsquellen mit dem Configuration Board
- MF_VIS_IF (Voltage/Current Source Mainframe Interface) verbindet den ABUS (analoger Bus) mit der Backplane des Mainframe
- TH_VIS_UF (Voltage/Current Source Testhead Interface) verbindet die Ausgänge der Strom- und Spannungsquellen, die sich im Mainframe befinden, mit dem Testkopf

Außerdem werden folgende Matrizen von dem Masterinstrument SOURCES angesprochen:

- VM_MUX_INT (Voltmeter Multiplexer Internal) ist ein 12-zu-1-Multiplexer, der sich in zwei Gruppen (1..6 und 7..12) aufteilt. Beide Gruppen haben separat die Möglichkeit auf Masse verschaltet zu werden. Der Ausgang des Multplexers wird auf eine Systemressource-Leitung gelegt, die je nach Konfiguration des Testsystems an weitere Hardware angebunden ist.
- AMUX (Analog Multiplexer) verschaltet analoge Signale im Testkopf zu verschiedenen Instrumenten und stellt eine weitere Verbindung zum Configuration Board her.
- MPCM (Meduim Power Crosspoint Matrix) ist eine 12x6 Kelvin-Matrix, die sich im Testkopf befindet. Sie stellt sechs Kelvin-Leitungen zum Configuration Board und zwölf Kelvin-Leitungen zu Meßinstrumenten oder Quellen im Testkopf zu Verfügung. Force und Sense Leitungen werden einzeln geschaltet.
- HPCM (High Power Crosspoint Matrix) ist eine 10x6 Kelvin-Matrix, die sich im Testkopf befindet. Sie stellt sechs Kelvin-Leitungen zum Configuration Board und zehn Kelvin-Leitungen zu Meßinstrumenten oder Quellen im Testkopf zu Verfügung. Force und Sense Leitungen werden einzeln geschaltet.

Aufgrund der Eingliederung der Interfaces und Matrizen in das „Software-Instrument“ SOURCES, braucht sich der Testingenieur keine Gedanken mehr machen, über welche Instrumente er beispielsweise Relais für den Signalpfad programmieren muß. Das heißt, nicht nur die Parameter der Quellen sind dem Instrument SOURCES zugewiesen, es beinhaltet jetzt auch alle Befehle für die Anschlußmöglichkeiten an die Peripherie. In dem angeführten Beispiel (Abbildung A.2) ist das an den Verbindungen zum CB (Configuration Board) zu erkennen, die auf dieser Ebene nicht mehr über das Basic-Instrument VS_IF programmiert werden.

```
CTRL SOURCES
    DO_FOR_INST    VS4
    CONNECT_HIGH_F CB       /* connect source high force output to CB  */
    CONNECT_HIGH_S CB       /* connect source high sense output to CB  */
    SET_VOLTAGE    10.3 V
    WAIT           2 ms
    GET_CURRENT    &current /* measure source output via VM              */
                            /* measpath connections are set automatically */
END;
CTRL SOURCES
    DO_FOR_INST    VS4
    CONNECT_HIGH_F OPEN     /* disconnect source high force output to CB  */
    CONNECT_HIGH_S OPEN     /* disconnect source high sense output to CB  */
                            /* force voltage is automatically set to 0 V  */
 END;
```

**Abbildung A.2: Programmierung über Masterinstrumente**

Schließlich kann noch der HRSD (High Resolution Signal Digitizer) über die Quellen verwendet werden. Mit dem HRSD können DC-Spannungen, AC-Spannungen und beliebige Signalformen am Configuration Board oder an der AMUX abgetastet werden. Der HRSD ist auch dem Überinstrument DSP_MASTER untergeordnet.

Es ist ersichtlich, daß Instrumente in dieser Software-Struktur zu syntaktisch sinnvollen Gruppen zusammengefaßt werden. Der Vorteil der Programmierung auf Überinstrumentenebene liegt darin, daß das Testerbetriebssystem dem Benutzer verschiedene Aufgaben in der Programmierung abnimmt, die auf Unterinstrumentenebene durchgeführt werden müßten. Beispielsweise tauchen Wartezeiten bei Schaltvorgängen nicht mehr im Code auf, da sie im System schon optimiert implementiert wurden. Das Testprogramm wird dadurch ebenfalls lesbarer, da zusammenhängende Vorgänge sich auch in der Syntax widerspiegeln.

### A.1.3 Super-Instrumente

Super-Instrumente stellen die oberste Stufe in der Software-Architektur dar. Sie übernehmen für den Benutzer das Routing des A-Bus und der digitalen Leitungen. Der AUTOROUTER ermöglicht es den Masterinstrumenten dabei direkt eine Verbindung zu den Anschlußpins aufzubauen.

Verfügbare Super-Instrumente sind:

- AUTOROUTER
- DIFF_VM
- VM
- SOURCES
- DSP_MASTER
- TSM
- MPIN
- APIN
- DPIN
- SCM1/SCM2

Master-Instrumente werden normalerweise über den CTRL-Befehl programmiert, Super-Instrumente mit Hilfe des CAR-Befehls (Control And Route) angesprochen. Super-Instrumente sollten als Master-Instrumente mit CTRL programmiert werden, wenn keine Verbindungsbefehle vorliegen. Liegen solche Befehle in einem Befehlsblock vor, wird die Ausführungsgeschwindigkeit durch den CAR-Befehle erhöht.

Mit Hilfe der Super-Instrumente ist es also möglich die Verschaltung von Quelle und Ziel der Software zu überlassen. Sie achtet dabei auch auf Ressourcen-Konflikte sowie Zugriffsrestriktionen.

## A.2 Testprogramm-Aufbau

### A.2.1 Initialisierung

Dieser Funktionsblock, bei SZ tpInit genannt, wird beim Laden eines Testprogramms zuerst abgearbeitet. In diesem Modul sollen alle Initialisierungen behandelt werden, welche die Software betreffen.

- Initialisierung von Variablen und Arrays
- Reservierung von Speicher
- Anlegen von Datenstrukturen
- Festlegung der Pfade, in denen sich Wave- und Patterndateien befinden
- Softwaretechnisches Erzeugen von Wave- und Patterndaten und ablegen der Daten im Speicher
- Definition der DUT-Pinnamen und deren Kombination mit der Pinnummer und Testkanalnummer
- Zusammenfassen von Pins zu Pinlisten (z.B. Eingangspins, Bidirektionale Pins, Ausgangspins, analoge Pins)

So sind alle globalen Variablen und Arrays des Testprogramms zu initialisieren. Benötigter Speicher ist zu reservieren, Datenstrukturen sind anzulegen und Pfade, in denen sich Wave- und Patterndateien befinden, sind festzulegen.

Wave- oder Patterndaten, die softwaretechnisch zu erzeugen sind, werden ebenfalls in dieser Funktion angelegt und im Speicher des Rechners abgelegt.

### A.2.2 Setup

Das Setup (bei SZ: tpSetupFunction) wird nach der Initialisierung aufgerufen und nur einmal beim Laden des Testprogramms behandelt. In dieser Funktion werden die grundlegenden Hardware-Einstellungen vorgenommen, die sehr zeitintensiv sind und über das gesamte Testprogramm unverändert bleiben:

- Laden der Kalibrierdaten
- Einstellen der Interfaces: Ressourcen, die während des gesamten Testprogramms an einem Pin anliegen, können schon in dieser Phase mittels interner Relais mit dem DUT verbunden werden.
- Laden von digitalen und analogen Signalen, die in Dateien vorliegen, und ablegen der Daten im Speicher des Testsystems.

### A.2.3 Testschritte

Nachdem alle Vorbereitungen getroffen wurden können die einzelnen Tests abgearbeitet werden. Man kann hierbei im wesentlichen zwischen folgenden Tests unterscheiden:

- Kontinuitätstests: Diese Tests sollen Bauelemente mit Kontaktfehlern und groben Schäden aussortieren. Dabei kommen in der Regel folgende Testschritte zur Anwendung:
  - Kontakttest
  - Kurzschlußtest
  - Stromaufnahmetest
- DC-Tests (statische Tests): Eine statische Strom- oder Spannungsmessung wird am Prüfling mit Hilfe von DC-Instrumenten durchgeführt. Üblicherweise werden folgende Größen gemessen:
  - Treiberstärke
  - Eingangsströme
  - Leckströme
  - Messungen bei verschiedenen Eingangsspannungen
- AC-Tests: Bei diesen Testschritten werden Zeitmessungen durchgeführt. Genauer teilen sich diese Messungen wie folgt auf:

  - Verzögerungszeit (propagation delay)
  - Übergangszeiten (T-Rise, T-Fall)
  - Setup und Holdzeiten von Registern
  - Signalperiode
  - Jitter (Rauschen des Flankenzeitpunkts)

- Funktionale Tests: Hierbei wird das Verhalten des Bausteins mit dessen Spezifikation verglichen. Der Prüfling wird stimuliert und muß die spezifizierte Bausteinantwort liefern. Weicht diese über bestimmte Toleranzen ab, ist der Baustein fehlerhaft.

- Spezielle Tests: Weitere Tests, die applikationsspezifisch sind, lassen sich in das obige Raster nicht einordnen.

Die Reihenfolge der Testschritte ist nicht starr festgelegt. Es muß versucht werden eine testzeitoptimierte Abfolge der Einzeltests herauszufinden. Dabei spielen Setup-Zeiten zwischen den einzelnen Testschritten eine Rolle sowie die Ausfallquoten der fehlerhaften Bauelementen in den Einzeltests. Innerhalb eines Testschritts kann folgende Einteilung vorgenommen werden:

- Setup: Während der Setup-Phase eines Testschritts werden die zu verwendenden Quellen und Meßgeräte via Relais an die Pins des Prüflings angeschlossen. Anschließend werden dem zu testenden Bauelement zunächst die Stromversorgungen zur Verfügung gestellt, danach werden die einzelnen Stimuli an das Device geliefert. Dies geschieht durch die Programmierung der Instrumente.

- Messung: Nach instrumentenspezifischen Einstellzeiten können die Messungen gestartet werden. Bei digitalen funktionalen Pattern kann der Punkt anlegen des Stimulus und Messung nicht mehr exakt getrennt werden, da in dem Funktionalpattern Stimuli und Vergleichswerte gleichzeitig enthalten sind und behandelt werden.

- Reset: Die Treiber und Spannungsversorgungen der verwendeten Instrumente werden wieder heruntergefahren, anschließend werden Relais geöffnet und verwendete Ressourcen freigegeben.
  Dieser Punkt kann im Zuge der Testprogrammoptimierung gekürzt werden, wenn der nachfolgende Testschritt die gleichen Instrumente, Einstellungen oder Ressourcen verwendet [BEY00].

- Auswertung: Am Ende jedes Testschrittes erfolgt ein Vergleich zwischen dem gemessenen- und dem Erwartungs-Wert. Liegt der Meßwert innerhalb eines spezifizierten Toleranzbandes um den Sollwert, hat der Prüfling den Test bestanden. Liegt er außerhalb dieses Bereichs, ist der Baustein fehlerhaft und wird aussortiert.

### A.2.4 Shutdown

Bevor das Testprogramm aus dem Speicher entfernt wird, muß der Grundzustand des Testers wiederhergestellt werden. Dies geschieht durch den Aufruf der Shutdown-Funktion (bei SZ: tpRelease), dabei sind Patterndaten aus dem Speicher zu entfernen und ein Hardware-Reset aller verwendeten Instrumente durchzuführen.

## A.3 Pattern-Dateiformate

### A.3.1 WDB

Die Waveform-Data-Base (WDB) ist eine maschinenunabhängige Datenbasis. Sie stellt die zentrale Basis bei der Arbeit mit Pattern dar, denn von hier aus werden testsystemspezifische Patterncodes sowie menschenlesbare systemunabhängige Austauschformate generiert. Pattern können mit Hilfe des Programms Wavemaker graphisch editiert werden. Dazu stehen folgende Fenster und Funktionen zur Verfügung.

- Signal-Definition-Editor (Abbildung A.3):
  Hier werden Signal-Listen angelegt, die Pinnamen den ATE-Pins zugeordnet und die Signalrichtung (Input, Output, Bidirectional) festgelegt.

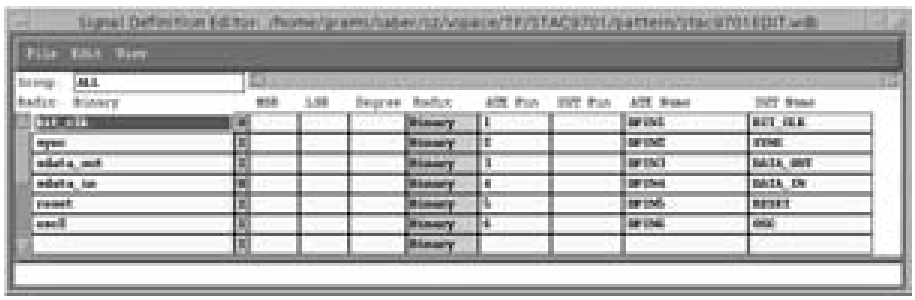


**Abbildung A.3: Signal-Definition-Editor**

- Timing-Editor (Abbildung A.4):
  Im Timing-Editor werden die Treiberformate der einzelnen Pins festgelegt. Dazu können zusammengesetzte Standardformate (wie beschrieben) verwendet werden oder eigene Formate aus Grundformaten generiert werden. Die Beschreibung der Treiberformate für alle Pins über eine Periodenlänge (Cycle) nennt man Timeplate. Mehrere dieser Timeplates können definiert werden und während der Ausführung eines Patterns verwendet werden.

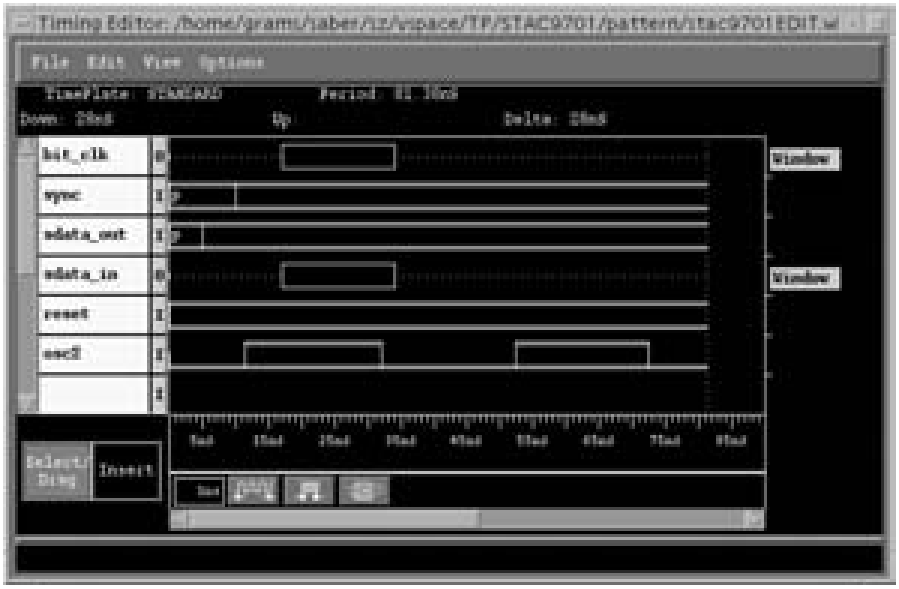



**Abbildung A.4: Timing-Editor**

- Equation-Editor (Abbildung A.5):
  Um eine bessere Skalierbarkeit der Timeplates zu ermöglichen, können im Equation-Editor Gleichungen eingegeben werden, die das zeitliche Verhalten der Pins untereinander charakterisieren. Durch den Einsatz von Variablen in den Gleichungen und Verknüpfungen der Gleichungen untereinander können zeitliche Verhältnisse auch bei Änderung der Patternfrequenz beibehalten werden.

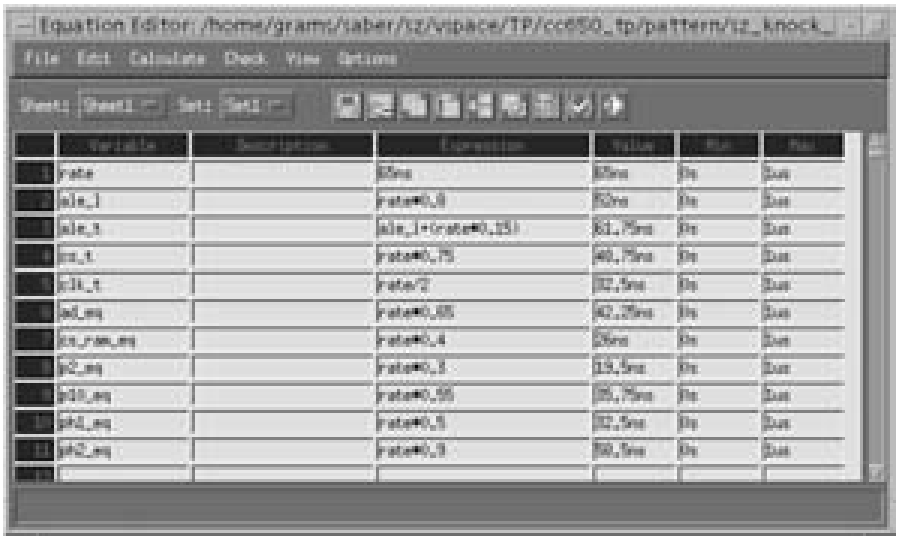



**Abbildung A.5: Equation-Editor**

- Pattern-Editor (Abbildung A.6):
  Die Vektoren, die in Verbindung mit dem Timing die Abfolge der zu treibenden und zu messenden Signale festlegen, werden in diesem tabellenartigen Fenster eingetragen. Ebenso sind hier alle Einsprungpunkte, Befehle für den Sequencer sowie Unterprogramme festzulegen.

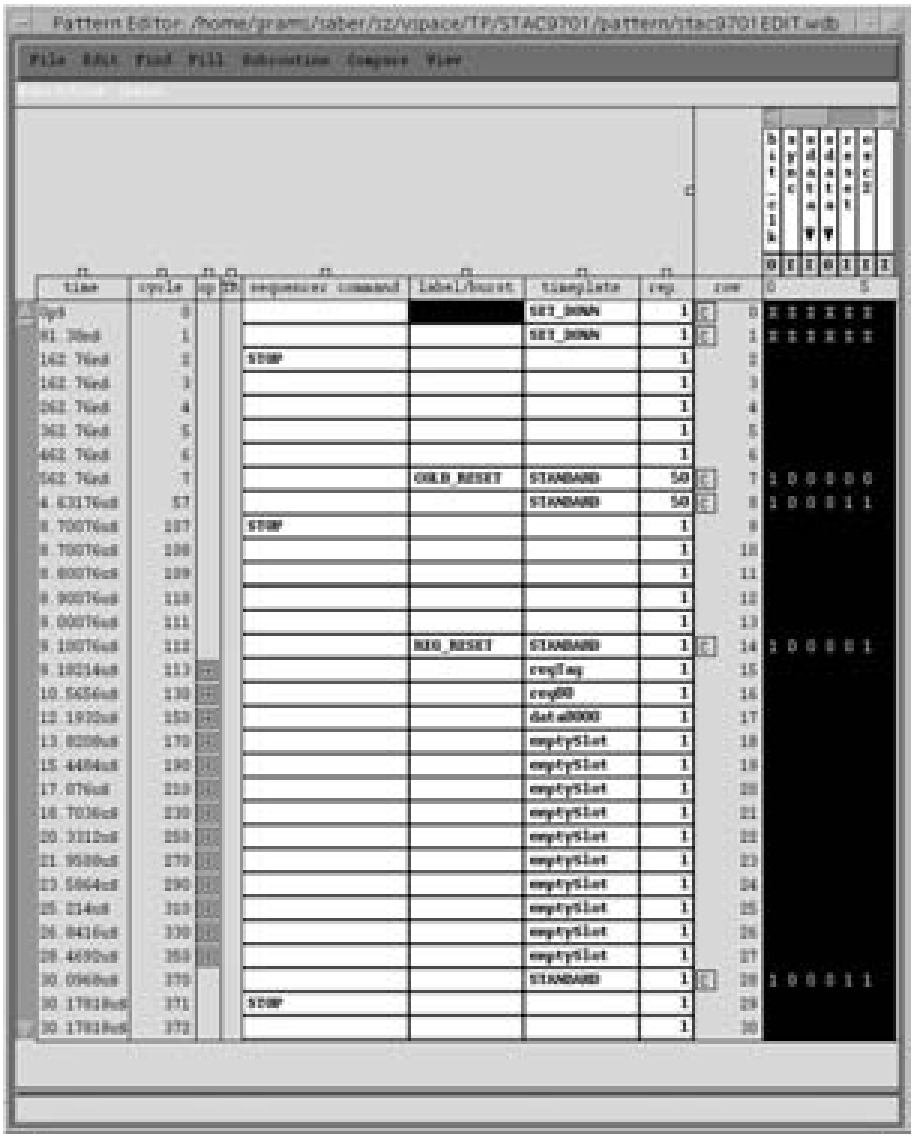



**Abbildung A.6: Pattern-Editor**

- Waveform-Editor (Abbildung A.7):
  Er zeigt die resultierenden Signalformen an, die sich aus der Kombination der Vektoren aus dem Pattern-Editor mit den Timeplates aus dem Timing-Editor ergeben.

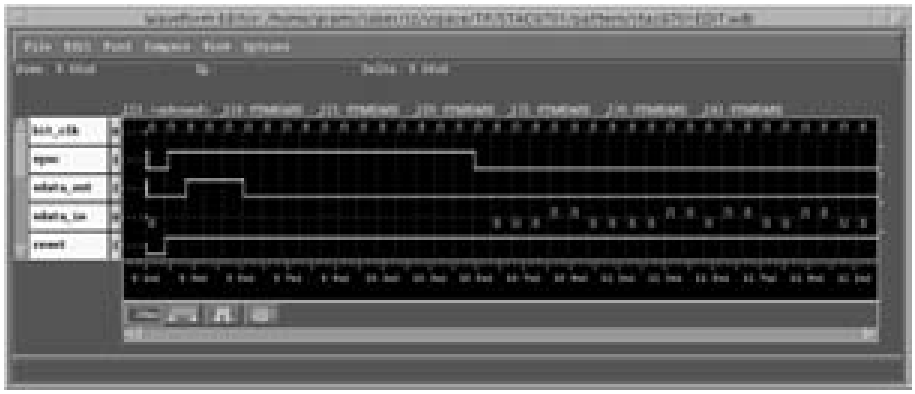


**Abbildung A.7: Waveform-Editor**

- Desk Top (Abbildung A.8):
  Er ermöglicht den Aufbau von Konvertern. Um die maschinenunabhängigen Datenformate wie WDB oder WGL in maschinenspezifische Formate zu konvertieren.

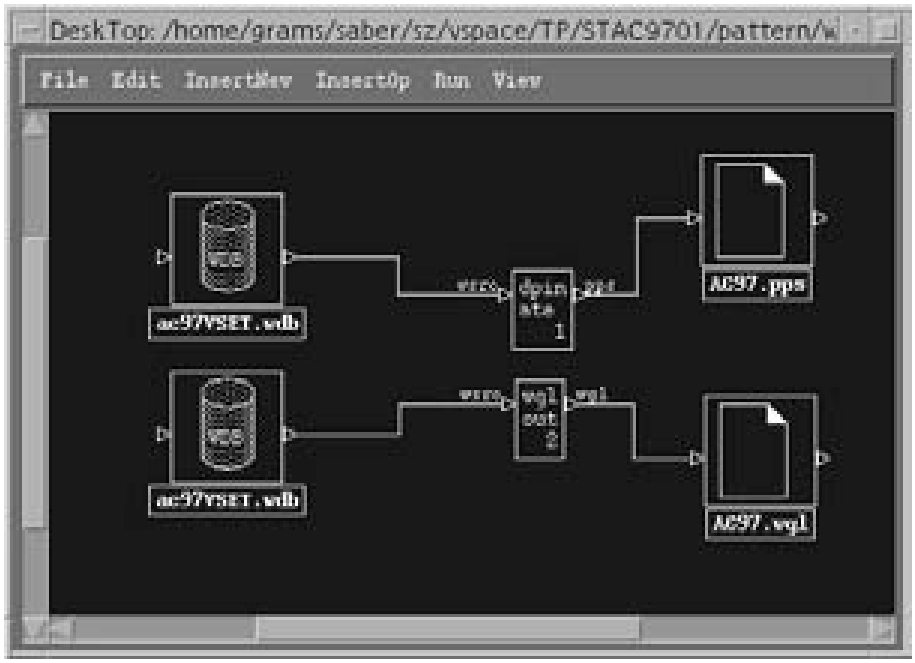



**Abbildung A.8: Konverter-Editor**

### A.3.2 WGL-File

Das WGL-Format als Form der Patterndarstellung ist relativ übersichtlich und auch vom Menschen lesbar. Die Daten liegen in diesem Format im ASCII-Code vor.

In der Timeplate-Sektion (Abbildung A.9) wird das zeitliche Verhalten der Pins festgelegt. Dies basiert auf den Signalformaten: DRZ, DNRZ, DRO, ... Die Bezeichnung input oder output ist von der DUT-Seite zu betrachten. Hinter den Zeichen in der Timeplate-Sektion verbergen sich folgende Bedeutungen (Tabelle A.2):

| Symbol | Bedeutung |
|---|---|
| D | Low treiben |
| P | Letzten Wert nochmals treiben |
| Q | Compare-Wert aus dem Pattern nehmen |
| S | Treiberwert aus dem Pattern übernehmen |
| U | High treiben |
| X | Dont't care |

**Tabelle A.2: Symbole der Timplate-Sektion**

```
timeplate STANDARD period 81.38nS
bit_clk   := output[0pS:X, 17nS:Q, 34nS:X];
sync      := input[0pS:P, 10nS:S];
sdata_out := input[0pS:P, 5nS:S];
sdata_in  := output[0pS:X, 17nS:Q, 34nS:X];
*reset    := input[0pS:S];
osc2      := input[0pS:D, 11.5nS:S, 32nS:D, 52.5nS:S, 72.5nS:D];
end
```

**Abbildung A.9: Timeplate-Section im WGl-Format**

Damit ergibt sich für die Signale der in Abbildung A.4 dargestellte Verlauf.

In der Pattern-Sektion (Abbildung A.10) stehen die Vektoren und die Befehle, die beim Patternablauf auszuführen sind (z.B. repeat, loop, conditional, Einsprungpunkte ...). Jedem Vektor wird ein Timeplate zugeordnet, welches das exakte zeitliche Verhalten des Vektors bestimmt. Weiterhin können häufig verwendete Vektorfolgen zu sogenannten subroutines zusammengefaßt werden, die über den Befehl call aufgerufen werden. Die Zeichen in den Vektoren haben hier folgende Bedeutung (Tabelle A.3):

| Zeichen | Bedeutung |
|---|---|
| 0 | Low (Treiben oder Compare steht im Timeplate und in der Richtung des Pins) |
| 1 | High (Treiben oder Compare steht im Timeplate und in der Richtung des Pins) |
| X | Don't care |
| Z | Tristate |
| - | Leervektor, dient nur zur Übersichtlichkeit |

**Tabelle A.3: Symbole der Pattern-Sektion**

Aus der Kombination der Vektordaten mit den entsprechenden Timeplatedaten sind die Vorgänge am Ausgang eines digitalen Pins definiert. Dies soll anhand der Abbildungen A.9 und A.10 kurz

```
pattern "group_ALL" (bit_clk,sync,sdata_out,sdata_in,*reset,osc2)
   vector(0, 0pS, SET_DOWN)        := [X Z Z X Z Z ];{LABEL INIT}
   vector(1, 81.38nS, SET_DOWN)    := [X Z Z X Z Z ];{STOP 0}
   vector(2, 162.76nS)             := [- - - - - - ];
   vector(3, 262.76nS)             := [- - - - - - ];
   vector(4, 362.76nS)             := [- - - - - - ];
   vector(5, 462.76nS)             := [- - - - - - ];
   repeat 50 vector( 6, 562.76nS,  STANDARD):= [1 0 0 0 0 0 ];{LABEL COLD_RESET}
   repeat 50 vector(56, 4.63176uS, STANDARD):= [1 0 0 0 1 1 ];{STOP 0}
   vector(106, 8.70076uS)          := [- - - - - - ];
   vector(107, 8.80076uS)          := [- - - - - - ];
   vector(108, 8.90076uS)          := [- - - - - - ];
   vector(109, 9.00076uS)          := [- - - - - - ];
   vector(110, 9.10076uS, STANDARD):= [1 0 0 0 0 1 ];{LABEL REG_RESET}
   call regTag();
   call reg00();
   call data0000();
   :
   :
   call emptySlot();
   call emptySlot();
   vector(368, 30.0968uS, STANDARD):= [1 0 0 0 1 1 ];{STOP 0}
   :
   :
subroutine emptySlot()
   vector(0, 0pS,      STANDARD)   := [1 0 0 0 1 1 ];
   vector(1, 81.38nS,  STANDARD)   := [1 0 0 0 1 1 ];
   vector(2, 162.76nS, STANDARD)   := [1 0 0 0 1 1 ];
end
```

**Abbildung A.10: Pattern-Section im WGL-Format**

erklärt werden.
Vektor 110 ist die Einsprungstelle (Label) für den Patternabschnitt REG_RESET. Das für diesen Vektor verwendete Timeplate heißt STANDARD (siehe Abbildung A.9). In diesem Timeplate steht für den Pin bit_clk ein Compare für einen bestimmten Zeitraum. Der Wert dieses Compare steht in den Vektordaten. In der Spalte für den Pin bit_clk (Spalte 1) findet sich eine „1“. Somit ist innerhalb des Comparezeitfensters ein High-Compare vorzunehmen. Die Aktionen an den anderen Pins ergeben sich analog.

Die Aufteilung der Daten in diesem Format entspricht der Vorgehensweise im Wavemaker, zu jedem Editor ist ein gewisser Abschnitt im WGL-Format vorhanden.

### A.3.3 PPS-Format

Das PPS-Format ist ein maschinenspezifisches Format, das speziell für den DPIN von SZ entwickelt wurde. Es ist ein sehr hardwarenahes Format, das heißt die Daten werden in einer Form abgelegt, die der Hardware der Pinelektronik entgegenkommt. Damit ist das File nicht vom Menschen lesbar. Um einen Einblick in die Datenstruktur zu bekommen, greift man auf die Routinen der Testerbibliothek zurück, mit deren Hilfe die Daten in das Testprogramm eingelesen und zum Testsystem übermittelt werden. Innerhalb dieser Routinen gibt es Funktionen, die eine lesbare

Darstellungsform erlauben, um die eingelesene Datenstruktur zu überprüfen. In Abbildung A.11 ist veranschaulicht, wie die Patterndaten, in der lesbaren Form dargestellt, in einem Teil der systemspezifischen Datenstruktur abgelegt werden.

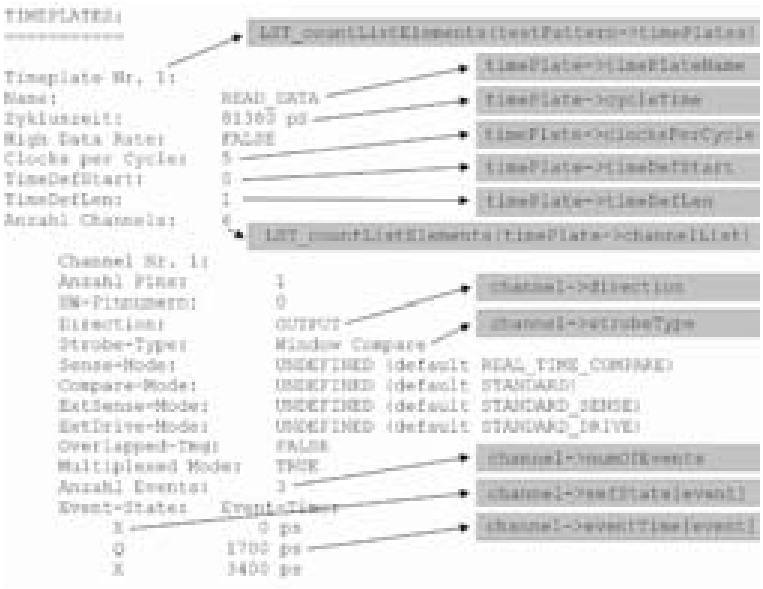



**Abbildung A.11: Timeplate-Section aus dem PPS-File**

# Anhang B

# Implementierung

## B.1 Datenfeld der Digitalen Pinelektronik - DPIN

Die vom Testprogramm ankommenden Befehle und Parameter werden in einem Datenfeld abgespeichert. Die Struktur ist den Unterbaugruppen bzw. den syntaktischen Zusammenhängen nachgebildet (Abbildung B.1).

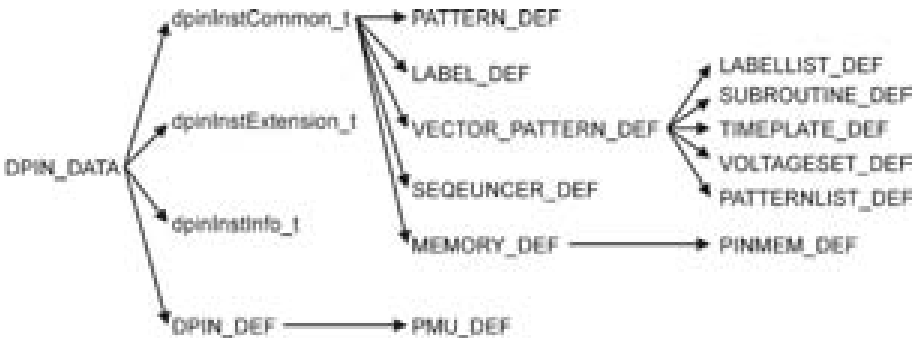



**Abbildung B.1: Globales Datenfeld des DPIN-Modells**

### B.1.1 PDCL und PMU

Die Werte, die über das PDCL-Panel (Abbildung 2.16) einstellbar sind, werden in der Struktur DPIN_DEF (Abbildung B.2) zusammengefaßt. Zusätzlich enthält diese Struktur Informationen über Drive-, Compare- und Patternmodi. Die Daten, die für PMU-Messungen notwendig sind, werden in einem eigenen Datenfeld zusammengefaßt, das an DPIN_DEF angegliedert ist. PMU_DEF (Abbildung B.3) enthält alle Daten, die vom Panel aus an der PMU (Abbildung 2.17) eingestelllt werden können und die zum Betrieb der PMU notwendig sind.

```
typedef struct
{
   double vih;                /* Drive High value                        */
   double vil;                /* Drive Low value                         */
   double voh;                /* Compare High value                      */
   double vol;                /* Compare Low value                       */
   double vcomm;              /* Commutaion voltage of load              */
   double iol;                /* Compare Low current                     */
   double ioh;                /* Compare High current                    */
   double tr;                 /* Driver output signal rise time          */
   double tf;                 /* Driver output signal fall time          */
   int  hyst;                 /* Hysteresis function                     */
   int  auto_ctrl;            /* Auto-ctrl of load                       */
   int  drive;                /* Signal state of driver                  */
                              /* 1 TRISTATE / 2 HIGH / 3 LOW / 4 PATTERN */
   int  load;                 /* State of load                           */
                              /* 1 ON / 0 OFF / 4 PATTERN                */
   int  compare;              /* 0 disable, 1 high compare, 2 low comp.  */
   int  pat_update;           /* 0 now pattern update, 1 pattern update  */
   char dpin_mode;            /* 0 drive, 1 measure                      */
   char output_relay;         /* 1 relay closed, 0 relay open            */
   char *dpin_relays;         /* 1 connected, 0 NOT connected            */
                              /* 1 lpmu / 2 dcl / 3-10 amux_line1-8      */
                              /* 11 tmux_line1 / 12 tmux_line2           */
   int dioWMask;              /* On which dio-lines write signal         */
   int dioMask;               /* From which dio-lines read signal        */
   int *LineSourceConnection; /* Which line connected with which source  */
   PMU_DEF * pmu;             /* Definition for PMU                      */
}DPIN_DEF;
```

**Abbildung B.2: Parameter des PDCL-Panel im Modell**

```
typedef struct
{
   char   gate;        /* 1 gate closed, 0 gate open             */
   char   pmu_mode;    /* 1->FVMC, 0->FCMV Z->nothing programmed */
   double fc;          /* current to force                       */
   double fv;          /* voltage to force                       */
   double expvolt;     /* voltage range for FCMV                 */
   double expcurr;     /* current range for FVMC                 */
   long   multi_meas;  /* multiple measure                       */
}PMU_DEF;
```

**Abbildung B.3: Parameter des PMU-Panel im Modell**

### B.1.2 Patterndaten

Die Datenstruktur „dpinInstCommon_t“ beinhaltet alle Informationen, die zur Ausführung eines Patterns notwendig sind. Die Abarbeitung des Pattern wird im Abschnitt 4.2.6.2 behandelt. Jetzt soll zunächst nur auf die Datenstruktur eingegangen werden (Abbildung B.4 und B.5).


**Abbildung B.4: Einträge für das Pattern im Datenfeld**

```
typedef struct
{
   PATTERN_DEF       *pattern;
   LABEL_DEF         *labels;
   VECTORPATTERN_DEF *vectorpattern[MAX_VECTORS];
   SEQUENCER_DEF     *sequencer;
   MEMORY_DEF        *memoryRecord;
   char              act_labelburst;
   double            act_pattern_len;  /* duration time for active pattern */
   double            *act_pattern_val; /* pattern value for DPINs          */
   int               act_pattern;      /* actual pattern in event (pattern) list           */
   int               end_pattern;      /* end position in event list of actaul pattern     */
   int               pattern_id;       /* Pattern-ID, if not the complete pattern will be simulated */
   int               start_pattern;    /* 0 -> do not start pattern, 1->start pattern      */
   char              update;           /* 0, command needs no DPIN-update, 1-> update DPIN */
   char              *file_path;       /* file path for pattern            */
   int               pattern_counter;  /* More DO_DOWN_LOADs are possible  */
   int               act_pattern_ptr;  /* Actual used pattern              */
   int               *act_signals;     /* List of active DPINS             */
   int               count_act_sig;    /* Number of active DPINs           */
   int               actUsedLine;
   long              pinInit[MAX_PINS];
}dpinInstCommon_t;
```

**Abbildung B.5: Struktur: dpinInstCommon**

#### B.1.2.1 Starre Eventliste

PATTERN_DEF enthält die Informationen der starren Eventliste. Im Sequencer-Modus werden die Daten aus dem Pattern heruntergerechnet und der nächste anstehende Event in diese Eventliste eingetragen (Abbildung B.6).

```
typedef struct
{
   double event_time;   /* time at which the event has to occur           */
   int    pin_number;   /* DPIN number                                     */
   int    pat_value;    /* Pattern value, 1,0,L,H,Z,X                     */
   int    vector;       /* Vector number in PATTERN!!!, not in event list */
   int    setDioLine;   /* 1 something happens on dio-Lines               */
}PATTERN_DEF;
```

**Abbildung B.6: Struktur: PATTERN_DEF**

#### Einsprungpunkte der starren Eventliste

In LABEL_DEF sind die Daten zusammengefaßt, die im Zusammenhang mit den Einsprungpunkten (Labels) in das Pattern stehen. Diese beziehen sich in diesem Abschnitt aber nur auf die starre Eventliste. Außerdem wird jedem Pattern eine eigene ID zugewiesen (Abbildung B.7).

```
typedef  struct
{
   char   label_name[256]; /* Label string                              */
   int    event_pos;       /* Start position of pattern in event list  */
   int    end_pos;         /* End  position of pattern in event list   */
   double run_time;        /* duration time for pattern                */
   int    pattern_id;      /* Pattern-ID, if not the complete pattern will be simulated */
}LABEL_DEF;
```

**Abbildung B.7: Struktur: LABEL_DEF**

### B.1.2.2 Sequencer

In der SEQUENCER_DEF sind Daten, die während einer Patternausführung zwischengespeichert werden müssen. Während einer Patternausführung wird der nächste Event des Simulators berechnet. Danach wird die entsprechende Funktion verlassen (storePatternVal), um in den Simulator zurückzuspringen. Nach der Abarbeitung des Events im Simulator muß der nächste Vorgang gesucht werden. Dabei muß bekannt sein an welcher Stelle die Suche angehalten wurde, um wieder korrekt einzusetzen. Des weiteren müssen die letzten Vorgänge an jedem Pin gespeichert werden, um redundante Vorgänge zu erkennen. Außerdem müssen Rücksprung-Adressen aus Unterprogrammen des Patterns gespeichert werden (Abbildung B.8).

#### Protokolldaten

Die Daten-Struktur MEMORY_DEF dient dem Aufzeichnen von Vorgängen im Pattern. So können alle Fehler oder Compares mitprotokolliert werden. Die Festlegungen über Speichergröße und pinunabhängige Daten werden hier gespeichert (Abbildung B.9).

```
typedef struct
{
   int    start_pos;
   int    act_position;
   int    return_pos;
   int    repeat_times;
   int    repeat_counter;
   int    count1;
   int    count2;
   double tot_time;
   double search_time;
   double delta_time;
   double last_time;
   int  * last_event;
   int    new_label_flag;
   int    subroutine_flag;
   int  * act_vector;
   int  * last_vector;
   int    break_count;       /* DO_STOP, DO_BREAK, x vectors have to be simulated */
   int    jump_in_pattern;   /* Vectorposition to start or restart a pattern      */
   int    stop_break_flag;   /* DO_STOP = 1, DO_BREAK = 2, DO_SUSPEND = 3         */
   int  * act_pins;
   int    get_result;        /* GET_VECTOR_COUNT = 1           */
   long   vector_counter;
   int    cycle_mode;        /* SINGLE = 1, AUTO = 2, STEP = 3 */
}SEQUENCER_DEF;
```

**Abbildung B.8: Struktur: SEQUENCER_DEF**

```
typedef struct
{
  int   memoryType;         /* ID of memory type 1..4 */
  long  memoryAlloc;        /* Allocated Memory       */
  long  memorySize;         /* Set Memory Size        */
  long  sampleDestination;
  long  memoryBlock;
  long  memoryOffset;
  long  transferSize;
  long *failEventCount;
  long *failCount;
  int  *pin;
  int   debugMode;
  PINMEM_DEF *pinMem[MAX_PINS]; /* Each pin has its own Memory */
}MEMORY_DEF;
```

**Abbildung B.9: Struktur: MEMORY_DEF**

In der PINMEM_DEF ist der Speicher für jeden Pin nachgebildet, in dem die protokollierten Daten abgelegt werden (Abbildung B.10)

```
typedef struct
{
  char  *** subroutineName;
  int    ** vector;
  char   ** expectedPattern;
  char   ** sensedPattern;
}PINMEM_DEF;
```

**Abbildung B.10: Struktur: PINMEM_DEF**

### Patternstruktur im Sequencer-Modus

Hinter VECTOR_PATTER_DEF verbirgt sich eine Struktur, die das Pattern (Label, Timeplate, Subroutine, Voltageset und Vektoren) beinhaltet, wie es zur flexiblen Behandlung im Sequencer-Modus benötigt wird (Abbildung B.11 und B.12). Die Struktur orientiert sich hierbei sehr stark an der allgemeinen Form der WDB.


**Abbildung B.11: Einträge der Vektoren im Datenfeld**

```
typedef struct
{
   char             name[256];
   int              act_pin;
   int              timeplate_count;
   int              voltageSetCount;
   LABELLIST_DEF   *labellist;
   VOLTAGESETS_DEF *voltageSets;
   SUBROUTINE_DEF  *subroutine;
   TIMEPLATE_DEF   *timeplate;
   PATTERNLIST_DEF *patternlist;
   long             pinAssignment[MAX_PINS];
}VECTORPATTERN_DEF;
```

**Abbildung B.12: Struktur: VECTORPATTERN_DEF**

### Einsprungpunkte im Sequencerbetrieb

LABELLIST_DEF speichert alle Labelnamen, die dazugehörigen Einsprungpunkte, die Zeitdauer des Patternabschnitts und ob dieses Label einen Patternburst nach sich zieht (Abbildung B.13).

```
typedef struct
{
   char   name[50];      /* Label names                                        */
   int    pos_start;     /* Start position of pattern part in pattern list  */
   double pattern_len;   /* Length of patternburst or label                 */
   long   labelOrBurst;
}LABELLIST_DEF;
```

**Abbildung B.13: Struktur: LABELLIST_DEF**

### Subroutines

Die SUBROUTINE_DEF beinhaltet alle Namen sämtlicher Subroutinen und wo sie im Patterndatenfeld zu finden sind (Abbildung B.14). Subroutinen sind Unterprogramme im Pattern, in denen - ähnlich wie im Programmcode - immer gleiche Vorgänge gespeichert werden, die in verschiedenen Patternabschnitten benötigt werden.

```
typedef struct
{
   char name[50];   /* Subroutine name */
   int  position;   /* Position of subroutine in pattern list */
}SUBROUTINE_DEF;
```

**Abbildung B.14: Struktur: SUBROUTINE_DEF**

### Timeplates

In der TIMEPLATE_DEF (Abbildung B.15) werden alle Daten der Timeplates aus dem Pattern gespeichert. In der eingebundenen PINTIME_DEF (Abbildung B.16) sind die einzelnen Event-Zeitpunkte pinweise aufgeschlüsselt.

```
typedef struct
{
   char         name[50];           /* Timeplate name       */
   double       period_time;        /* Period time          */
   PINTIME_DEF *pintime;            /* Timing for each pin  */
   int          pin_count;          /* Number of pins       */
   int          max_timepoint_count;
}TIMEPLATE_DEF;
```

**Abbildung B.15: Struktur: TIMEPLATE_DEF**

Die PATTERNLIST_DEF beinhaltet die Befehle, die in der Patternstruktur stehen sowie die Vektoren des Patterns (Abbildung B.17).

```
typedef struct
{
   int    timepoint_count;              /* Number timepoints for one pin */
   double time[MAX_TIMEPOINT_COUNT];    /* Timeset                       */
   char   value[MAX_TIMEPOINT_COUNT];   /* Value for timepoint           */
}PINTIME_DEF;
```

**Abbildung B.16: Struktur: PINTIME_DEF**

```
typedef struct
{
   int cmd_id;               /* Commands in pattern list: 130=LABEL, 131=STOP,   132=REPEAT , */
                             /* 133=CALL,  134=VECTOR, 135=SUBROUTINE                        */
   int vector[2*MAX_PINS];   /* Parameters for Commands e.g label_id, repeat_times... or      */
                             /* vector including vector_nr, timeplate_id, vector_values      */
}PATTERNLIST_DEF;
```

**Abbildung B.17: Struktur: PATTERNLIST_DEF**

## B.2 Verhaltensmodellierung der aktiven Last

Die aktive Last ist aus einer Dioden-Brücke aufgebaut, die an zwei Stromquellen angeschlossen ist. Die Knoten der Brücke (Abbildung B.18) sind wie folgt belegt:

1. IOL (DUT-Senke)
2. VCOM (Schwellenspannung)
3. DUT-Anschluß
4. IOH (DUT-Quelle)

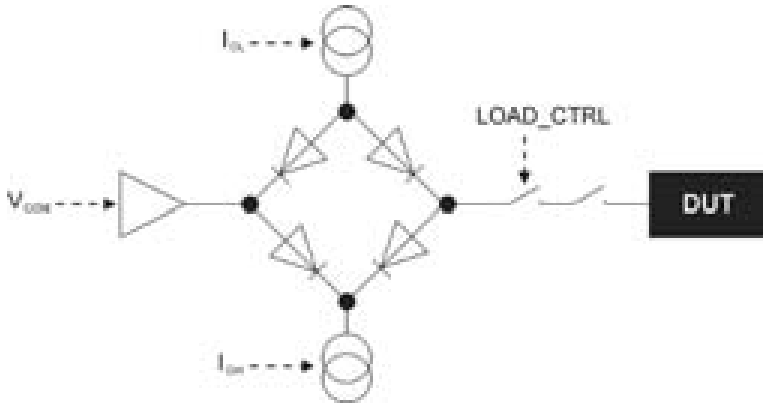



**Abbildung B.18: Aktive Last im SZ-Testsystem**

Die Leitung LOAD_CTRL steuert den Anschluß der aktiven Last an das DUT. LOAD_CTRL kann über das Pattern gesteuert werden oder aus dem Testprogramm über eigene Befehle programmiert werden.

Wird die aktive Last mit dem DUT verbunden steuert der Zustand des DUT-Ausgangs, welche der beiden Stromquellen angelegt wird. Auf diese Weise ist es möglich einen Tristate-Zustand am DUT eindeutig zu detektieren. Hierbei soll der hochohmige Ausgang des DUT überprüft werden. Das heißt, ob der Ausgang hochohmig gegenüber Masse und Betriebsspannnung ist. In diesem Fall liefert der Ausgang kein definiertes Potential. Somit ist dieser TRISTATE nur mit einem Komparator nicht eindeutig zu erkennen. Wird die aktive Last zugeschaltet, kann dieser Zustand jedoch eindeutig detektiert werden (Fall VDUT = VCOM). Generell kann man zwischen drei Fällen beim Betrieb der aktiven Last unterscheiden:

- VDUT > VCOM:
  Der Widerstand zwischen zu testendem Pin und VDD ist nicht hochohmig. Die punktiert dargestellten Ströme werden sich einstellen (Abbildung B.19). Am DUT-Pin ist eine Spannung zu messen, die eindeutig unter der Schwellspannung VCOM liegt.

- VDUT = VCOM:
  Die Widerstände zwischen zu testendem Pin zu VDD und VSS sind hochohmig: Die schwarz dargestellten Ströme werden sich einstellen (Abbildung B.19). Am DUT-Pin ist exakt die Spannung VCOM zu messen.

- VDUT < VCOM:
  Der Widerstand zwischen zu testendem Pin und VSS ist nicht hochohmig. Die gestrichelt dargestellten Ströme werden sich einstellen (Abbildung B.19). Am DUT-Pin ist eine Spannung zu messen, die eindeutig über der Schwellspannung VCOM liegt.

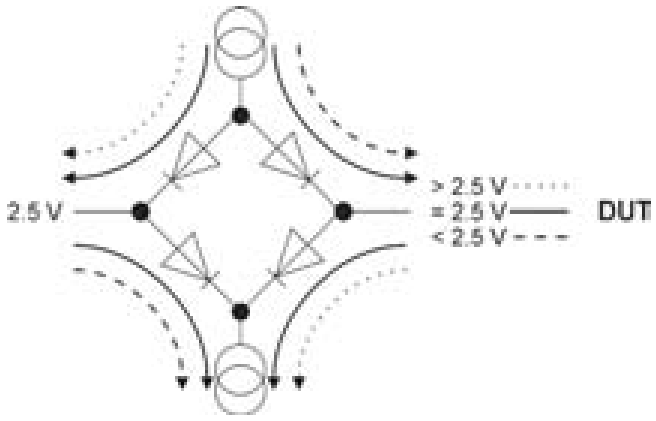



**Abbildung B.19: Funktionsweise der aktiven Last**

### B.2.1 Steuerung der aktiven Last

Die aktive Last wird über das Instrument DPIN programmiert (Abbildung B.20). Die drei Parameter IOL, IOH und VCOM werden mit SET_IOL, SET_IOH und SET_VCOM eingestellt.

```
CTRL DPIN
    DO_FOR_SIGNAL P1
    SET_IOL       0.01700
    SET_IOH      -0.01700
    SET_VCOMM     2.50000
END
```

**Abbildung B.20: Programmierung der aktiven Last**

Es gibt zwei Möglichkeiten, die aktive Last während des Testprogrammlaufs zu steuern:

- Steuerung mit Pattern
  Im Default-Zustand ist die Steuerung der aktiven Last mit dem Pattern vorgesehen. Mit dem Befehl SET_LOAD_AUTO_CTRL kann diese Steuerung ab- und wieder angeschaltet werden.

- Steuerung über Befehle
  Die aktive Last kann ebenso aus dem Testprogramm gesteuert werden (Abbildung B.21) Mit den Befehlen DO_FOR_SIGNAL oder DO_FOR_SIGNAL_LIST werden die zu programmierenden DPINs ausgewählt. Mit SET_LOAD_AUTO_CTRL OFF wird die Patternsteuerung der aktiven Last abgeschaltet. Die LOAD_CTRL wird mit Hilfe der Befehle DO_FOR_LINE und SELECT_DATA_SOURCE auf LOGIC_ONE gesetzt, d.h. die aktive Last ist permanent an den Prüfling angeschlossen. Der Befehl SET_LOAD macht nur Sinn, wenn über SELECT_DATA_SOURCE der Parameter PATTERN ausgewählt wurde. In diesem Fall übernimmt wiederum das Pattern die Steuerung der aktiven Last. Wird das Pattern angehalten, kann die aktive Last über den Befehl SET_DRIVE an das DUT angeschlossen oder abgetrennt werden. Läuft das Pattern weiter, übernimmt es die Steuerung wieder.

```
CTRL DPIN
    DO_FOR_SIGNAL      P1
    SET_LOAD_AUTO_CTRL FALSE
    DO_FOR_LINE        LOAD_CTRL
    SELECT_DATA_SOURCE LOGIC_ONE
END
```

**Abbildung B.21: Steuerung der aktiven Last aus dem Testprogramm**

### B.2.2 Modellierung der aktiven Last

Die aktive Last spielt im Testbetrieb nur bei Messungen an DUT-Pins eine Rolle. Das heißt sie fällt nur in's Gewicht, wenn im Pattern ein Compare vorliegt, ganz egal ob sie speziell durch

diesen Vorgang zugeschaltet wurde oder aufgrund der Programmierung im Testprogramm ständig anliegt.

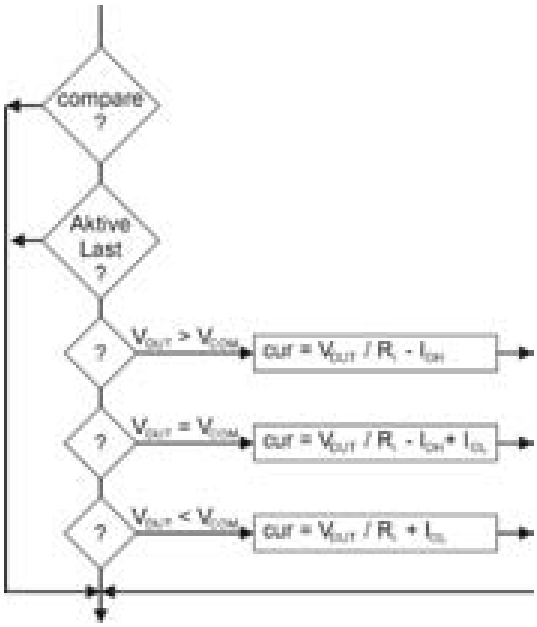



**Abbildung B.22: Modellierung der aktiven Last**

Liegt also ein Compare vor so ist zu überprüfen, ob die aktive Last zugeschaltet ist (Abbildung B.22). Die dazugehörende Auswertung erfolgt in den C-Routinen updateDpin.c und readPatternVal.c.

- updateDpin.c
  Hier wird die Nutzung der aktiven Last in Verbindung mit der Programmierung überprüft. Die aktive Last wird aktiviert, wenn auto_ctrl abgeschaltet und SELECT_DATA_SOURCE auf LOGIC_ONE gesetzt ist.
- readPatternVal.c
  Hier wird die Nutzung der aktiven Last in Verbindung mit dem Pattern überprüft. Die aktive Last wird aktiviert, wenn
  - ein Z-Compare vorliegt und auto_ctrl gesetzt ist
  - ein Z-Compare vorliegt und SELECT_DATA_SOURCE auf PATTERN gesetzt ist
  - SELECT_DATA_SOURCE auf LOGIC_ONE gesetzt ist.

Liegt die Aktive Last aufgrund dieses Entscheidungsprozesses an, muß der Strom berechnet werden, der in das DUT fließt bzw. aus dem DUT gezogen wird. Dies kann nach den vorherigen Überlegungen mit Hilfe einfacher „if-Abfragen“ geschehen. Der Strom errechnet sich dann gemäß dem anliegenden Fall und wie in Abbildung B.19 beschrieben.

### B.2.3 Referenzmodell der aktiven Last

Abschließend muß die Frage gestellt werden, ob eine derart abstrahierte Modellierung ausreichend ist. Um diese Frage zu beantworten, wurde ein Modell der aktiven Last mit Standard-Bauelementen im Saber nachgebildet. Dabei wurde die restliche Testmaschine vernachlässigt und nur das Verhalten dieser Schaltungskomponente untersucht. Die aktive Last ist in diesem Modell (Abbildung B.23) über eine Diodenbrücke realisiert, an der die Stromquelle bzw. -senke sowie die Vergleichsspannung und das DUT angeschlossen sind. Das DUT wird hierbei mit einer Spannungsquelle realisiert, das die verschiedenen Spannungen simuliert, die an diesem Knoten auftreten können.

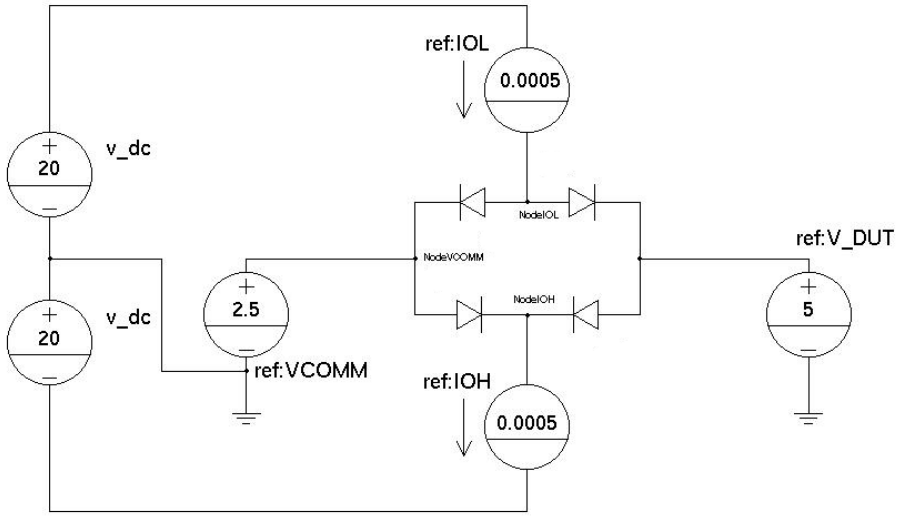



**Abbildung B.23: Modell der aktiven Last mit Dioden**

Variiert man also die Spannung des DUT erhält man den charakteristischen Spannungsverlauf an den Knoten der Diodenbrücke (Abbildung B.24) und es stellt sich der erwartete Stromverlauf am Pin zum DUT ein (Abbildung B.25).

An diesen Kurvenverläufen ist zu erkennen, daß die abstrahierte Modellierung vollkommen ausreicht. Nur an den Knickpunkten, das sind die Grenzen der Fallunterscheidungen, liefert eine genauere Simulation einen besseren, kontinuierlichen Kurvenverlauf. Dieser geht jedoch mit einer erheblich höheren Simulationszeit einher. Dieser Mehraufwand ist für die zu erwartenden Ergebnisse nicht akzeptabel.

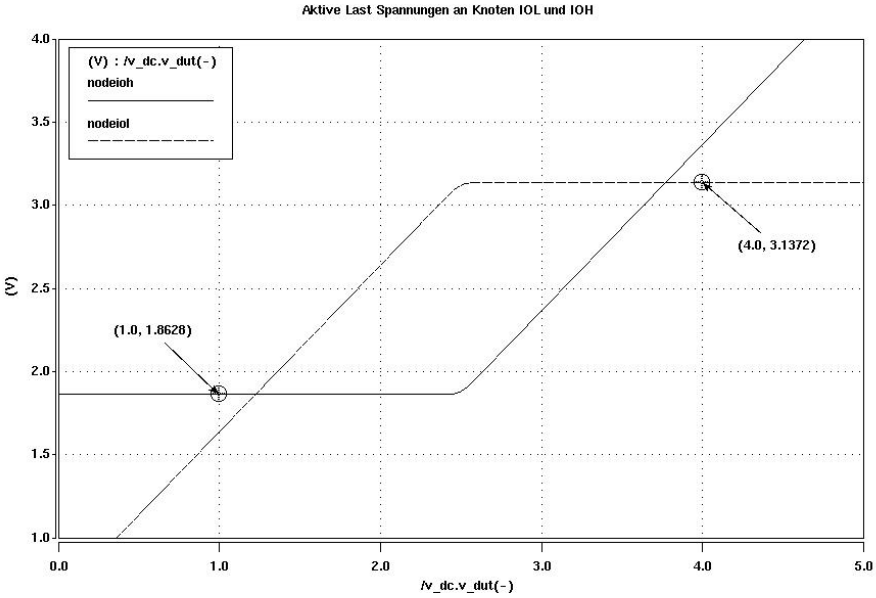



**Abbildung B.24: Spannungsverlauf an der aktiven Last**

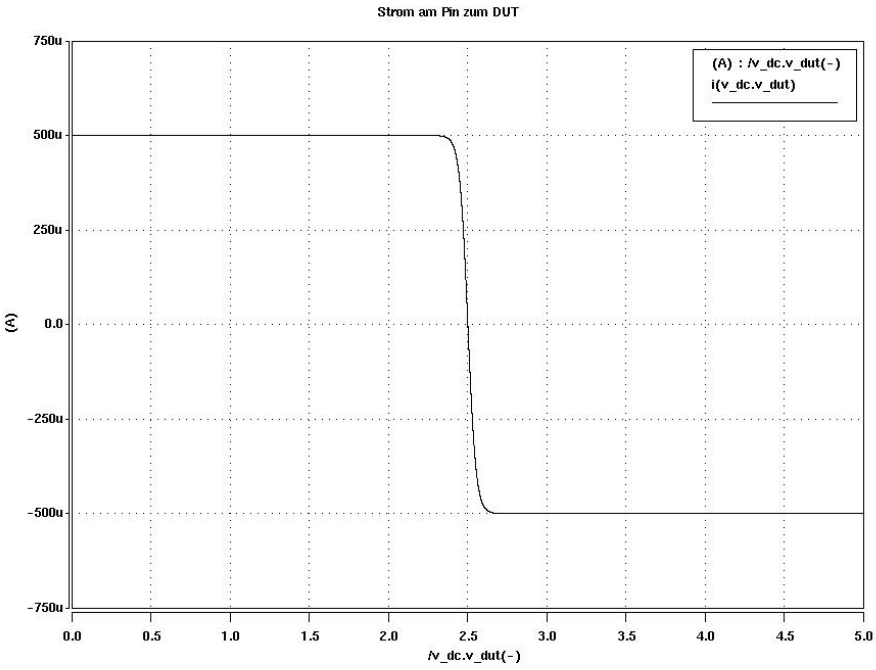



**Abbildung B.25: Stromverlauf am DUT-Pin der aktiven Last**

## B.3 Patternabarbeitung auf verschiedenen Abstraktionsebenen

Die Abfolge digitaler Signalfolgen ist im Pattern festgelegt (siehe Abschnitt 2.2.4.2). Diese Daten werden in der realen Umgebung durch die Entwicklungsumgebung geladen und über das Bussystem an die digitalen Pinkarten weitergeleitet. Im virtuellen Test wäre dieser Weg ebenfalls machbar. Wir haben jedoch im Abschnitt 4.2.3.2 gesehen, daß die Kommunikation zwischen Testprogramm und Simulator einen erhöhten Zeitbedarf, unter anderem aufgrund der Kodierung hat. Die Schnittstelle stellt somit einen Engpaß dar. Warum soll also diese Kommunikation zusätzlich belastet werden. Es ist ebenso möglich, die Daten direkt in den Simulator zu laden. Dazu wird nur der Befehl zum Laden einer Pattern-Datei vom Testprogramm an den Simulator gesendet. Das DPIN-Modell importiert die ausgewählte Datei direkt. Bei dieser Aktion kann es auf die SPACE-Bibliothek zum Bearbeiten von Patterndaten zugreifen. Diese Variante ist erheblich schneller als der Umweg über das Kommunikations-Interface.

Die Problematik der Abstraktionsebenen oder Genauigkeitsstufen wurde schon angesprochen (Abschnitt 4.2.6.2). Auch auf der Ebene der Pattern muß es verschiedene Ausführungsvarianten geben, die davon abhängen, welche Ergebnisse der Testingenieur sehen will. Deshalb muß die zu den Abstraktionsebenen (Abbildung 4.21) passende Patternsimulation ausgeführt werden. Die Abstraktionsebenen umfassen im Patternbereich folgende Stufen:

- Pattern-ID (Abschnitt B.3.1)
- Starre Eventliste (Abschnitt B.3.2)
- Sequencer (Abschnitt B.3.3)
- Master-Clock (Abschnitt B.3.4)

### B.3.1 Pattern-ID

Diese Modellierung läßt sich auf „Abstraktionsebene 1“ (siehe Abschnitt 4.2.6.2) integrieren. Auf dieser Abstraktionsebene wird jedem Pattern eine ID zugewiesen. Zusätzlich sind mit der Pattern-ID Informationen über die Länge des Patterns verbunden. ID und Laufzeit des Patterns werden an das DUT auf einer eigenen Leitung geliefert. Das DUT erkennt die ID und verhält sich so, als würde das reale Pattern laufen. Ein Zeitsprung über die Länge der Patternlaufzeit beschleunigt die Simulation und stellt wieder Synchronität mit der Testprogramm-Entwicklungsumgebung her.
Für die Anwendung dieser Ebene spielen zwei Gründe eine wesentliche Rolle:

- Es sollen einfache Messungen durchgeführt werden.
  Das Pattern liefert Stimuli und Setup an das DUT, die für diese Messung erforderlich sind. Es beinhaltet keine Compares. In diesem Fall gewinnt man mit einer detaillierten Simulation des Pattern keine neuen Erkenntnisse über den Testschritt. Es ist wichtig zu erkennen, ob das Pattern richtig aufgesetzt wurde. Das jedoch kann hiermit kontrolliert werden. Liegt eine fehlerhafte Programmierung vor (Pattern nicht geladen, falsche Einsprungpunkte gewählt ...), wird dem DUT eine fehlerhafte oder keine Pattern-ID mitgeteilt. Des weiteren ist es wichtig die korrekte Antwort des Bausteins zu analysieren, die nicht mit Hilfe des Patterns erfaßt wird.

- Bei sehr großen Pattern ist der Simulationsaufwand sehr zeitintensiv.
  Somit ist es möglich, daß der Zeitaufwand in keinem Verhältnis zum Informationsgewinn steht. In diesem Fall ist ebenso zu prüfen, ob eine abstrakte Simulation mit Pattern-ID zunächst ausreicht.

Ein Beispiel soll die Vorteile der Simulation mit Pattern-ID verdeutlichen:
Beim Test des Mixed-Signal-ICs AC97, ein Audio Codec Baustein, der bei Soundkarten Anwendung findet, soll der Digital-Analog-Converter in einem Testschritt getestet werden. Dazu wird ein digitaler serieller Datenstrom, der die digitalisierten und PCM-kodierten Daten eines analogen Sinus-Signals enthält, an die digitalen Eingangspins des DUT angelegt. Der Prüfling erfaßt die Daten und liefert an seinem analogen Ausgang das erwartete Sinus-Signal. Dieser Testschritt beinhaltet ein Pattern mit 131.310 Vektoren, die über 2 Millionen Pegeländerungen an den digitalen Schnittstellen zum Prüfling hervorrufen. Die Simulationszeit hierfür lag bei über sieben Stunden. (Abbildung B.26 zeigt nur einen sehr kleinen Ausschnitt aus dieser Simulation, um das Pattern noch zu erkennen.)

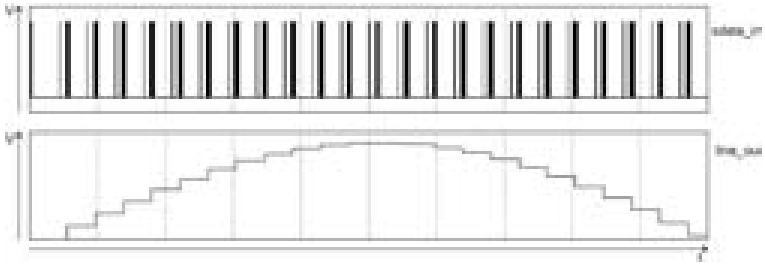


**Abbildung B.26: Simulation mit Pattern**

Das ausgeführte Pattern enthält kein einziges Compare, denn die Analyse der analogen Baustein-Antwort erfolgt durch Abtastung dieses Signalverlaufs und mit Hilfe von DSP-Routinen.

Bei der DSP-Programmierung können die Anweisungen innerhalb eines CTRL-Blocks in „Sequenzen“ und „Jobs“ aufgeteilt werden (Abbildung B.27). Ein „Job“ ist die kleinste Einheit. Es

stehen vorgefertigte „Jobs“ und anwender-definierte „Jobs“ zur Verfügung. Ein „Job“ besteht aus einem Namen als Kennung und einer Abfolge von Testbefehlen. Mehrere „Jobs“ können zu einer „Sequenz“ zusammengefaßt werden. Diese werden direkt in den Programmspeicher des DSP geladen und können nach ihrem Aufruf schnell ausgeführt werden.
Neben dem Programmspeicher beinhaltet der DSP einen X- und Y-Speicher für den Real- und Imaginärteil der DSP-Berechnungen.

Diese Operationen, die sehr viele Speicherzugriffe benötigen, sind oftmals fehlerbehaftet und sind deshalb aufwendig in der Fehlersuche.

```
void doFFT( long dspNr, long dspSourceAdr, long dspCmplxAdr, long dspEffAdr, long sampleCnt )
{
   static dspOsId_t  fft,fftjob1,fftjob2;

   CTRL DSP_MASTER
      DO_FOR_INST DSP_OS       /* Abtastung  Signal RW0A */
      DO_FOR                      DSP_CD1
      DO_OPEN_SEQUENCE            fft
        ADD_JOB                   DSP_REAL_FFT
          SET_DSP_NR_OF_SAMPLES   sampleCnt
          SET_DSP_SOURCE1        "sampledSine1kHz"
          SET_DSP_DESTINATION1   "sampledFFT"
          SET_DSP_REF1           "refWave512"
        END_JOB                   fftjob1

        ADD_JOB                   DSP_COMPL_TO_EFF
          SET_DSP_NR_OF_SAMPLES   sampleCnt/2
          SET_DSP_SOURCE1        "sampledFFT"
          SET_DSP_DESTINATION1   "sampledEFFT"
        END_JOB                   fftjob2
      DO_CLOSE_SEQUENCE

      EXECUTE_SEQUENCE            fft
   END
}
```

**Abbildung B.27: Programmierung des DSP mittels Job-Konzept**

Es ist also ersichtlich, daß das Hauptaugenmerk bei der Überprüfung dieses Testschritts nicht auf der Ausführung des Patterns liegt, sondern auf der richtigen Programmierung der DSP-Routinen. Deshalb macht es wenig Sinn sieben Stunden auf die Abarbeitung eines Patterns zu warten, das für den Testingenieur keinen zusätzlichen Informationsgewinn bedeutet.
Um dieses Problem zu lösen, wird jedem Pattern eine Identifikationsnummer zugewiesen. Zusätzlich wird eine Leitung eingeführt, die es an keinem realen Testsystem gibt. Auf dieser Leitung wird die Patter-ID vom Testsystem an den Prüfling übertragen. Eine modifizierte intelligente Version des DUT erkennt die ID und verhält sich genauso, als ob das Pattern angelegt wurde, und liefert an seinem analogen Ausgang den erwarteten Sinus. Das Testsystem tastet das Signal ab und wertet es mit Hilfe der DSP-Routinen aus (Abbildung B.28).
Diese Simulation dauert nur drei Minuten und erlaubt es dem Testingenieur die gewünschte Programmierung zu untersuchen. Damit ist die Notwendigkeit dieser Abstraktion bewiesen.

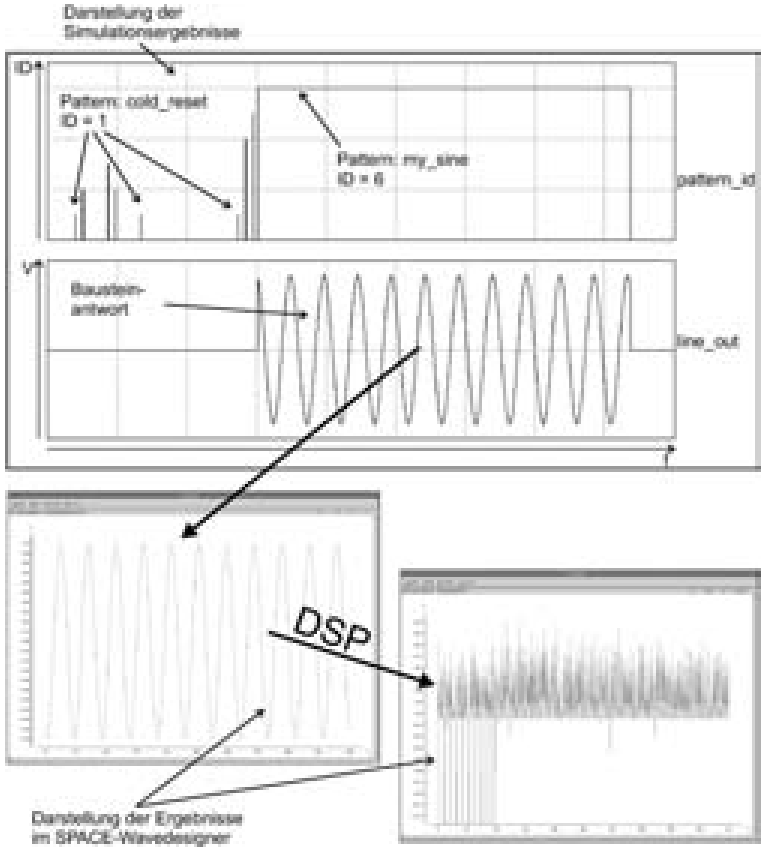



**Abbildung B.28: Simulation mit Pattern-ID**

### B.3.2 Starre Eventliste

Bei der starren Eventliste wird der Ansatz verfolgt, dem Simulator eine zeitlich exakte Abfolge der Vorgänge an den Pins beim Ausführen eines Patterns vorzugeben. Auf diese Weise soll eine hohe Ausführungsgeschwindigkeit der Pattern erzielt werden.
Aus den Patterndaten (WGL oder PPS, siehe Abschnitt 2.2.4) wird vor dem Start der Simulation mit Hilfe eines Parsers eine Liste errechnet, in der die Reihenfolge der ablaufenden Vorgänge hart festgelegt ist. Diese Liste wird in der Datenstruktur (Abbildung B.1) unter PATTERN_DEF abgespeichert. Bei der Testprogrammausführung kann keinerlei Einfluß auf das Pattern genommen werden, da die starre Liste keinerlei Veränderungen zur Laufzeit zuläßt.

Abbildung B.29 zeigt den Ausschnitt einer Eventliste. Das Beispiel-Pattern umfaßt 6767 Vektoren mit 5 Einsprungsmarkierungen (Labels). Das Label INIT startet bei Vektor 0 und dauert 162.76 ns, COLD_RESET startet bei Vektor 6 und dauert 8138 ns. Im nachfolgenden Abschnitt der Eventliste befinden sich die Zeitpunkte, an denen an den Pins Werte angelegt werden müssen. Die Zeitpunkte sind immer auf die Startpunkte der Labels bezogen.

```
6767
5
LABEL INIT        0    162.760000
LABEL COLD_RESET  6    8138.000000
LABEL REG_RESET   615  21077.420000
LABEL REG26_FF00  2664 21077.420000
LABEL BLANK_FRAME 4720 21077.420000
0.000000 1 X 0
0.000000 2 Z 0
0.000000 3 Z 0
0.000000 4 X 0
0.000000 5 Z 0
0.000000 6 Z 0
0.000000 1 X 6
0.000000 2 Z 6
0.000000 3 Z 6
0.000000 4 X 6
0.000000 5 0 6
0.000000 6 0 6
5.000000 3 0 6
10.000000 2 0 6
17.000000 1 H 6
17.000000 4 L 6
34.000000 1 X 6
34.000000 4 X 6
98.380000 1 H 6
```

**Abbildung B.29: Aufbau der Event-Liste**

`5.000000 3 0 6` bedeutet, zum Zeitpunkt 5 ns nach Aufruf des Labels wird am Pin 3 das Signal low getrieben. Dieser Wert stammt aus dem Vektor 6. Diese Information ist für Debugging-Zwecke notwendig. (Die verschiedenen Zeichen für diesen Abschnitt sind in Tabelle B.1 erklärt.)

| Zeichen | Bedeutung |
|---|---|
| 0 | Treibe LOW-Level |
| 1 | Treibe HIGH-Level |
| L | Compare LOW-Level |
| H | Compare HIGH-Level |
| X | Don't care |
| Z | Tristate |

**Tabelle B.1: Zeichenerklärung in der Eventliste**

Der Nachteil einer starren Eventliste ist die eingeschränkte Debugging-Fähigkeit. Es kann nur festgestellt werden, daß an einem bestimmten Vektor ein Fehler aufgetreten ist. Eine Veränderung des Timings oder der Spannungspegel kann mit einer starren Eventliste nicht vorgenommen werden. Ebenso können mit einer vorweg berechneten Eventliste keine Shmooplots über den Parameter Patternfrequenz durchgeführt werden. Diese Analyseart ist notwendig, um den Grenzbereich des DUT über bestimmte Paramter auszuloten. Dazu werden die entsprechenden Parameter variiert und das Pass-/Fail-Verhalten des Prüflings protokolliert (Abbildung B.30). Dieses Verfahren wird vor allem während der Charakterisierung von Bausteinen angewendet und sollte somit im virtuellen Test unterstützt werden.

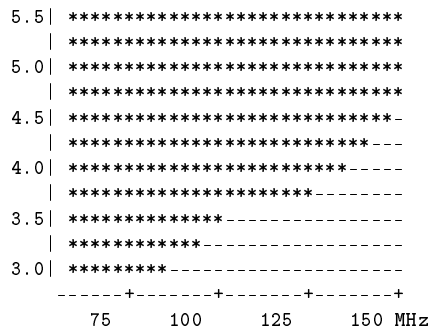



**Abbildung B.30: Frequenz-Betriebsspannungs-Shmoo**

### B.3.3 Sequencer

In einer starren Eventliste kann zur Simulationszeit keine Änderung vorgenommen werden. Dies ist aber für eine effektive Debugging-Arbeit unerlässlich. Deswegen ist es auf einer detaillierteren Ebene notwendig, die Events für den Simulator erst zur Laufzeit zu berechnen. In der realen Testmaschine wird dies vom sogenannten Sequencer realisiert.

Das exakte Zeitverhalten ist in den Timeplates festgelegt. Aus der Kombination der Timeplatedaten mit den Vektordaten sind die Vorgänge an den Ausgängen der digitalen Pins exakt definiert (siehe Abschnitt 2.2.4). Aufgrund dieser Trennung ist es zur Laufzeit möglich, Änderungen im Timing vorzunehmen und direkt im gleichen Testprogrammlauf auszuprobieren. Diese Überprogrammierung des Timings, wie es am realen Tester möglich ist, muß in der Simulation ebenso unterstützt werden.

Die Datenstruktur „dpinInstCommon_t“ (Abbildung B.4) beinhaltet alle Informationen, die zur Ausführung eines Patterns notwendig sind. Während der Ausführung können verschiedene Abstraktionsebenen ausgewählt werden (siehe Abschnitt 4.2.6.2). Deshalb war es notwendig die Daten in verschiedene Unterstrukturen aufzuspalten, die hier eingehängt sind. Des weiteren sind auf dieser Ebene Variablen untergebracht, die zum Laden und Starten des Pattern notwendig sind (Abbildung B.5). Mit diesem Datenfeld (siehe Abschnitt B.1) kann der Sequencer flexibel zur Laufzeit alle Vorgänge am DPIN aus den Patterndaten heraus berechnen. Dabei gliedert sich diese Steuerung (gleiches gilt auch für die starre Eventliste und Pattern-ID) in das Konzept des DPIN-Modells (Abbildung 4.20) ein.

Der Sequencer muß also zunächst im Timeplate den nächsten möglichen Zeitpunkt herausfinden, an dem sich an einem Pin ein Zustand ändert. Um den nächsten Zustand zu finden kann man sich vorstellen, Suchalgorithmen wie Bubble-Sort zu implementieren. Es ist jedoch fraglich, ob sich dieser Aufwand zeitlich lohnt, bei einer Bestückung von 128 DPINs, die in der Regel maximal fünf Zustandsänderungen pro Cycle erfahren. Es ist also einfacher und effektiver das Timeplate

im kleinsten verfügbaren Raster, das sich aus der Taktfrequenz des DPIN ergibt, abzufahren, um den nächsten Event zu suchen. Ist also die nächste Zustandsänderung gefunden, wird diese mit dem aktuellen Wert des Vektors angepaßt (siehe Abschnitt 2.2.4) und in der Eventliste abgelegt. Danach wird anhand von Schatten-Variablen überprüft, ob der gefundene und einzustellende Wert bereits aufgrund vorheriger Vorgänge an diesem Pin eingestellt ist. Ist dies der Fall, muß die Simulation für den gefundenen Vorgang nicht erneut angestoßen werden. Die Erkennung dieser Redundanten Vorgänge spart erheblich Simulationszeit ein. Muß der Vorgang simuliert werden, müssen die Spannungspegel für den Vektor an das Simulationsmodell übergeben und die Simulation dieses Schrittes durchgeführt werden. Danach beginnt der Prozeß von vorne und der nächste Event wird gesucht.

Das Sequencer-Modell kann entweder mit den Daten aus dem WGL-File oder direkt mit den PPS-Daten gespeist werden. Bei der Variante über das WGL-File muß ein Parser das WGL-File aufbereiten und in der Datenstruktur der Modelle ablegen. Dieser Parser muß von Hand geschrieben und ständig aktualisiert werden. Ein weiteres Problem bei dieser Implementierung ist, daß im Standard-WGL-Format nur ein beschränkter Sprachsatz für das jeweilige Instrument vorhanden ist. Spezial-Einstellungen werden über Kommentare im Code ausgegeben, die nicht zum regulären Sprachumfang von WGL gehören. Deshalb muß der Parser ständig auf dem neuesten Stand gehalten werden. Ferner können Fehler bei der Umsetzung auftreten, die beim regulären Weg nicht in Erscheinung treten. Deshalb ist die Variante über das PPS-File die ideale Lösung. Hier kann die Bibliothek, die zum Laden und Bearbeiten der Patterndaten in der Testentwicklungsumgebung vorhanden ist, auf der Simulationsseite ebenso genutzt werden. Auf diese Weise wird sichergestellt, daß die Daten auf die gleiche Weise behandelt werden, wie beim realen Test.

### B.3.4 Master-Clock

Die genaueste Stufe der Simulation beim DPIN basiert auf dem Master-Clock. Dies ist die hardwarenaheste Modellrealisierung. Hierbei treten mehr Events auf als im Sequencer Mode, da sich der Master-Clock aus allen Vorgängen innerhalb eines Patterns berechnet. Diese so generierten Events sind testerintern und werden häufig zur Synchronisation verschiedener Instrumente benötigt.

Der Master-Clock wird vom Instrument Clock-Source geliefert. Die Frequenz ist die größte gemeinsame Frequenz aller definierter Timeplates im ausgewählten Pattern-File. Der Master-Clock liegt in einem Bereich zwischen 50 MHz und 100 MHz. Dieser Takt wird für den DPIN und die DSP-Instrumente (DSP_MASTER, HRSG, HRSD, VHSSG, VHSSD) von der CLOCK_-SOURCE generiert. Die Master-Instrumente DPIN und DSS steuern die Clock-Source, so daß sie nicht separat programmiert werden muß, wenn sie verwendet wird.

# Anhang C

# Abkürzungen

| | |
|---|---|
| ADC | Analog Digital Converter |
| ATE | Automatic Test Equipment |
| ATPG | Automatic Test Pattern Generation |
| BIST | Build in Self Test |
| CAR | Control And Route |
| CPU | Central Processing Unit |
| CTRL | Control |
| DAC | Digital Analog Converter |
| DIB | Device Interface Board |
| DNRZ | Delayed Non Return to Zero |
| DPIN | Digital Pin |
| DSP | Digital Signal Processing |
| DSS | Digital Sub System |
| DUT | Device Under Test |
| DVM | Digital Volt Meter |
| DVT | Digital Virtual Tester |
| EDA | Electronic Design Automation |
| GPIB | General Purpose Interface Bus |
| HDL | Hardware Description Language |
| IC | Integrated Circuit |
| IEEE | Institute of Electrical and Electronics Engineers |
| IMS | Integrated Measurement Systems, Inc. |
| ITRS | International Technology Roadmap for Semiconductors |
| LPMU | Local Parametric Measurement Unit |
| LRS | Lehrstuhl für Rechnergestützten Schaltungsentwurf |

| | |
|---|---|
| MPIN | Mixed-Signal Pin |
| MSPU | Mixed-Signal Pin Unit |
| NRZ | Non Return to Zero |
| PMU | Parametric Measurement Unit |
| PPS | Patter Pin Sequencer |
| RMS | Root Mean Square |
| RO | Return to One |
| ROI | Return to One Inverted |
| RZ | Return to Zero |
| SBC | Surround By Complement |
| SNR | Signal to Noise Ratio |
| SOC | System On Chip |
| STIL | Standard Test Interface Language |
| TAC | Technical Activity Council |
| TRP | Test Resource Partitioning |
| TSDL | Test Signal Description Language |
| tttc | Test Technology Technical Council |
| UUT | Unit Under Test |
| VHDL | Very High Speed Integrated Circuit Hardware Description Language |
| VLSI | Very Large Scale Integration |
| VME | Versa Module Europe Bus |
| VTE | Virtual Test Emulator |
| VXI | VMEbus eXtension for Instrumentation |
| WDB | Waveform Data Base |
| WGL | Waveform Generation Language |

# Anhang D

# Definitionen

**ATE**

*Automatic Test Equipment, Automatisches Testsystem*

Als **ATE** wird das System bezeichnet, das einer zu testenden Schaltung - dem $\rightarrow DUT$ - die zu deren Test benötigten Eingangssignale zur Verfügung stellt und die vom $\rightarrow DUT$ zurückgegebenen Ausgangssignale auswertet. Dabei wird der Ablauf des Zurverfügungstellens von Signalen und der Auswertung durch ein $\rightarrow$*Testprogramm* gesteuert, das auf einem Steuerrechner abläuft.

**BIST**

*Build In Self Test, Eingebaute Selbsttest-Funktionalität*

Unter **BIST** werden Schaltungsteile verstanden, die zur eigentlichen Schaltung hinzugefügt werden und in der Lage sind, die Schaltung in bestimmten Grenzen einem Selbsttest zu unterziehen. Der **BIST** vereinfacht den Test der Schaltung erheblich. Da er jedoch nicht alle Tests durchführen kann, werden auch für $\rightarrow DUT's$ mit **BIST** weiterhin Testsysteme benötigt. Der Nachteil einer mit **BIST** erweiterten Schaltung kann eine verringerte Steuer- und Beobachtbarkeit sein. Außerdem birgt auch der **BIST** potentielle Fehlerquellen, so daß auch die Selbsttesteinrichtungen einer Schaltung getestet werden müssen.

**CMOS**

*Complementary Metal Oxide Semiconductor, Komplementärer Metalloxid-Halbleiter*

Bei **CMOS** handelt es sich um eine Technologie, bei der Schaltungen, bezogen auf die Versorgungsanschlüsse, komplementär aufgebaut werden. Der Vorteil ist im wesentlichen eine geringe (idealerweise keine) Leistungsaufnahme während der Ruhephasen der Schaltung.

**DfT**

*Design for Testability, Entwurf unter Berücksichtigung der Testbarkeit*

Mit **DfT** werden Designmaßnahmen bezeichnet, die die Testbarkeit der Schaltung schon bei ihrem Entwurf berücksichtigen und so den Test der gewünschten Parameter der Schaltung vereinfachen, wenn nicht sogar erst ermöglichen.

**DFT**

*Discrete Fourier Transformation, Diskrete Fouriertransformation*

Die **DFT** ist die Übertragung der allgemeinen Fouriertransformation in einen Werteraum, in dem die Werte nur an diskreten Stellen, bezogen auf die Basis ihrer Gewinnung, vorliegen. Sie wird im Zusammenhang mit dieser Arbeit genutzt, um - unter Annahme bestimmter Voraussetzungen - die zu diskreten Zeitpunkten erfaßten Meßwerte in den Frequenzbereich zu transformieren und eine Spektralanalyse der Werte zu ermöglichen.

**DIB**

*Device Interface Board, Beschaltungsplatine*

Das **DIB** ist im Bereich des Schaltungstests die Adapterplatine, die das $\rightarrow ATE$ mit dem $\rightarrow DUT$ verbindet. Dabei wird der in den meisten Fällen $\rightarrow DUT$-unabhängige Anschluß am Testsystem auf die speziellen Gegebenheiten des $\rightarrow DUT$ umgesetzt. Zusätzlich zu dieser Adaptionsaufgabe wird das **DIB** häufig auch dafür verwendet, zusätzlich benötigte Beschaltungen aufzubauen.

**DSP-basiertes Testen**

*Testen das auf Digital Signal Processing beruht*

**DSP-basiertes Testen** bietet ein leistungsfähiges Verfahren zur Erzeugung, Verarbeitung und Auswertung analoger Signale. Dazu wird heute aus Kosten- und Flexibilitätsgründen verstärkt versucht, aufwendige Analyse-Hardware wie z.B. Spektral-Analysatoren im Rechner nachzubilden ($\rightarrow FFT$), wofür aber zunächst eine Abtastung der analogen Signale erforderlich ist.

**DUT**

*Device under Test, zu testendes Bauteil*

Mit **DUT** ist die zu testende Zielschaltung gemeint, für die das $\rightarrow ATE$ konfiguriert und das $\rightarrow$*Testprogramm* entwickelt worden ist. In manchen Veröffentlichungen wird auch von CUT (engl.: Circuit under Test, zu testende Schaltung) gesprochen, der Begriff bezieht sich aber genauso auf die Zielschaltung.

**Echtzeit** [WIK04]

**Echtzeit** bedeutet, daß das Ergebnis einer Berechnung innerhalb eines gewissen Zeitraumes garantiert vorliegt, d.h. bevor eine bestimmte Zeitschranke erreicht ist.

Ein Echtzeit-System (engl. real-time system) muß also ein Berechnungsergebnis nicht nur mit dem richtigen Wert, sondern auch noch rechtzeitig liefern. Andernfalls hat das System versagt.

Eine genaue Definition der Zeitschranke hängt von der Anwendung ab und kann daher nicht allgemein angegeben werden. Ein Echtzeit-System reagiert also auf alle Ereignisse rechtzeitig und verarbeitet die Daten „schritthaltend“ mit dem technischen Prozess.
Um die Echtzeit-Fähigkeit eines Echtzeit-Systems theoretisch nachweisen zu können, müssen die Häufigkeit der externen Ereignisse, die Laufzeit der einzelnen Programmteile und die Zeitschranken bekannt sein.
Rechner zur Steuerung von technischen Einrichtungen oder Prozessen wie Maschinen, verfahrenstechnische Anlagen, Autos, Flugzeugen usw. sind immer Echtzeit-Systeme.

**FFT**
*Fast Fourier Transformation, schnelle Fouriertransformation*
Die **FFT** ist eine Spezialform der →*DFT*. Ist die Anzahl der Meßwerte eine Potenz von „2”, so beschleunigt die **FFT** die →*DFT* durch Einhalten einer bestimmten Berechnungsfolge erheblich.

**Heterogenes Simulationssystem**
Ein **heterogenes Simulationssystem** ist ein System, in dem Komponentenmodelle unterschiedlicher Abstraktionsgrade und mit unterschiedlichen Ausführungskonzepten gemeinsam simuliert werden. [KLU03]

**Laufzeit**
Unter **Laufzeit** versteht man die Zeit, die ein Testprogramm zu seiner Ausführung benötigt.

**Loadboard**
siehe →*DIB*

**Mainframe**
*Hauptrahmen*
Als **Mainframe** wird bei einem →*ATE* in der bis heute zumeist verwendeten Form der Schaltschrank bezeichnet, in dem sich die Stromversorgung des Testsystems und weitere Komponenten befinden, die aus Platz- oder Wärmegründen nicht im →*Testhead* untergebracht sind.

**Pass-/Fail-Entscheidung**
Die **Pass-/Fail-Entscheidung** meint die Unterscheidung zwischen getesteten Bauteilen, die alle Tests im Rahmen der Testgrenzen bestanden haben, und Bauteilen, die bei einem oder mehreren Tests die Testgrenzen über- bzw. unterschritten haben. Bei ersten lautet das Urteil „Pass” (engl. durchgekommen, passiert), bei letzteren „Fail” (engl. scheitern, fehlgeschlagen).

**Pinelektronik**
Die **Pinelektronik** eines →*ATE* ist die Elektronik, die die Testsignale an das oder vom →*DUT* überträgt. Da diese Elektronik in einem Testsystem mehrfach, im besten Fall für jeden Pin des →*DUT*, vorhanden ist, wird von **Pinelektronik** gesprochen.

**Prüfprogrammsimulation**
Unter dem Begriff **Prüfprogrammsimulation** wird die Simulation des systemspezifischen Testcodes zur Programmierung der Testschritte verstanden. Ziel der **Prüfprogrammsimulation** ist der Nachweis des korrekten Testcodes im Zusammenspiel mit den Testerinstrumenten und dem Prüflingsverhalten.

**Reale Zeit**
Die **Reale Zeit** ist die Zeit, die ein Vorgang in der Realität benötigt.

**Rechenzeit** [GRM01]
Benötigte Zeit für die Analyse eines Systems über einen gegebenen →*Simulationszeitraum*.

**Simulationszeit** [LAN95]
Die Größe, welche im Modell den Bezug zur Zeit im realen System herstellt, bezeichnet man als **Simulationszeit**.

**Simulationszeitraum**
Zeitspanne über die ein System analysiert wird.

**SOC**
*System on Chip, System in einer Schaltung integriert*
Als **SOC** werden Schaltungen bezeichnet, bei denen mehrere, an sich separate Einzelschaltungen in einer gemeinsamen Schaltung realisiert sind. Bei den Einzelschaltungen kann es sich bereits um komplexe Schaltungen handeln, die nun im **SOC** entweder wegen der jetzt vorhandenen technologischen Voraussetzungen oder zur Vermeidung von Signalintegritätsproblemen in einer einzigen Schaltung integriert sind.

**Testhead**
*Testkopf*
Der **Testhead** ist der Teil des →*ATE*, der die Schnittstellen zum →*DUT* zur Verfügung stellt und die vom →*ATE* erzeugten Stimuli dem →*DUT* über eine Adapterplatine (das sogenannte →*Loadboard*) zur Verfügung stellt und dessen Ausgangssignale zur Verarbeitung und Auswertung im Testsystem weiterleitet.

**Test-Pattern**
*Testmuster*
Ein **Test-Pattern** setzt sich aus einer zeitlichen Folge von →*Testvektoren* zusammen. Das Testmuster definiert den Ablauf des Digitaltests, bei dem das →*DUT* in der zum Test benötigten Weise stimuliert wird und die an den Ausgängen anliegenden Zustände mit den erwarteten aus dem Testmuster verglichen werden.

**Testprogramm**
Das **Testprogramm** ist ein Abfolge von Maschinenbefehlen, mit denen der Steuerrechner im Testsystem die verschiedenen Komponenten im Testsystem z.B. dazu veranlaßt, in der gewünschten Abfolge das →*DUT* zu stimulieren oder Signale von diesem kommend zu messen. Das **Testprogramm** steuert den gesamten Testablauf, die Auswertung des Tests und das Zusammenspiel der verschiedenen benötigten Komponenten und Geräte.

**Testsimulation**
Unter dem Begriff **Testsimulation** wird die Simulation des Testverfahrens und seiner Parameter verstanden. Ausgangspunkt für die **Testsimulation** ist eine Testspezifikation mit zugehörigem Testsetup.

**Testsystem-Emulator**
Der Emulator eines Testsystems bildet in Software jeden Zustand des Testers nach. Das ermöglicht zunächst einen Konsistenz-Check, ob alle im Testprogramm verwendeten Instrumente auch als Hardware im Testsystem vorhanden sind. Des weiteren erlaubt es eine erste Überprüfung der Syntax auf Fehler, ohne einen Simulator oder reale Testsystem-Hardware heranzuziehen.

**Testvektor**
Als Testvektor wird eine zu einem bestimmten Zeitpunkt gültige Belegung der Dateneingänge einer Schaltung sowie die Angabe der zu diesem Zeitpunkt erwarteten Zustände an den Ausgängen der Schaltung bezeichnet. Im wesentlichen wird der Begriff im Zusammenhang mit dem Test digitaler Schaltungen verwendet. Dabei wird die Schaltung zum gewünschten Zeitpunkt mit einem Satz von Zuständen aus der Menge der gültigen Zustände stimuliert und die Ausgangssignale mit den dort erwarteten Zuständen verglichen.

**Time-to-Market**
*Zeit bis zur Markteinführung*
Ein Kriterium für ein produzierendes Unternehmen ist die Zeit, wie lange es dauert, ein neues Produkt von der Entwicklung bis zu Markteinführung zu bringen. Diese **Time-to-Market** bestimmt, wie weit ein Produkt vor oder nach der Konkurrenz eingeführt werden kann. Sie hat damit einen großen Einfluß auf die möglichen Gewinnspannen und sollte deshalb natürlich so kurz wie möglich sein. Im Bereich des Schaltungstests, der häufig das letzte Element des Herstellungsprozesses ist, wird versucht, die **Time-to-Market** durch kurze, effektive Testmethoden zu verringern.

**Virtueller Test**
Unter dem Begriff **Virtueller Test** werden simulationsbasierte Methoden zur Verifikation des Tests und des für den Test erforderlichen Testcodes inklusive dem Zusammenspiel mit →*Test-Pattern* verstanden. Verfahren zur Generierung von Testmustern (digital wie analog) werden

ebensowenig berücksichtigt wie Testbeschreibungssprachen zur teilautomatischen Generierung des Testcodes.

# Literaturverzeichnis


[ANA98-1] Analogy, Inc.:
*Testify - Data Sheet*
http://www.analogy.com/Test/Apptech/testify.htm
Analogy, Inc., Beaverton, Oergon, 1998

[ANA98-2] Analogy, Inc.:
*Saber-IC Pro - Data Sheet*
http://www.analogy.com/Test/Apptech/sabericpro.htm
Analogy, Inc., Beaverton, Oergon, 1998

[ANT02] Dr. Mario Anton, Jürgen Weber, Jens Schuster, Andreas Lehmann,
ATMEL Germany GmbH, Heilbronn
*Verhaltensmodellierung von Mixed-Signal Automotive ICs für die Anwendung im virtuellen Test*
ANALOG 2002, Bremen, Germany, 13.05.2002 - 14.05.2002

[ANT03] Mario Anton, Jürgen Weber
*Simulationsmethodik für integrierte Mixed-Signal Automotive Schaltkreise*
Analog'03, Heilbronn, 10.-12.September 2003

[AUS93] Tom Austin
*Creating a Mixed-Signal Simulation Capability for Concurrent IC Design and Test Program Development*
International Test Conference, 1993

[BAT92] Stephan C Bateman, William H Kao
*Simulation of an Integrated Design and Test Environment for Mixed-Signal Integrated Circuits*
International Test Conference, 1992

[BEL99] David San Segundo Bello, Ronald Tangelder, Hans Kerkhoff
*Modeling of a verification test system for mixed signal cicuits in hSpice*
5th International Mixed Signal Testing Workshop (IMSTW), Whistler, Canada, June 1999

[BEY00] H. Beyer, W. Tenten, P. Schmidhuber
*A New Approach to Test Program Time Analysis and Optimization*
6th International Mixed-Signal Testing Workshop (IMSTW), Montpellier, 2000

[BLU98] K. Blume
*Manging Large Projects*
Java Magazin, Ausgabe 3.98, Software & Support Verlag GmbH, 1998

[CAU95] Pascal Caunerge, Claude Abraham
*Achieving Simulation-Based Test Program Verification and Fault Simulation Capabilities for Mixed-Signal Systems*
The European Design and Test Conference, Paris, Frankreich, März 1995

[CHA98] Ed Chang, David Cheung, Robert Huston, Jim Seaton, Gary Smith
*A Scalable Architecture for VLSI Test*
ITC '98, Washinton, USA

[DAN95] Presse-Information
*IMS TEST SOFTWARE und CREDENCE SYSTEMS demonstrieren Virtual Test für Duo und Vista*
ITC '95, Washington D.C. / Haar, 7. November 1995

[DAN97-1] Integrated Measurement Sytems, Inc., Virtual Test Division
*Dantes - Data Sheet*
http://www.virtualtest.com/dantesdocs.html
IMS, Beaverton, Oregon, 1997

[DAN97-2] Integrated Measurement Systems, Inc.
*Virtual Test Success Story - Dantes Virtual Test System Speeds Time-to-Market at Harris Semiconductors*
IMS, Beaverton, Oregon, 1997

[DAN97-3] Integrated Measurement Systems, Inc.
*Dantes Component Library*
IMS, Beaverton, Oregon, 1997

[DAN98-1] Integrated Measurement Systems, Inc.
*Dantes HP94000 ATE Module - Technical Description*
IMS, Beaverton, Oregon, 1998

[DAN98-2] Integrated Measurement Systems, Inc.
*Dantes HP83000 Multimedia Series - Technical Description*
IMS, Beaverton, Oregon, 1998

[DVT97] Presse-Information
*IMS' Virtual Test Division Announces $50,000 Test Development Alternative to Multimillion Dollar Automated Test Equipment*
IMS, Beaverton, Oregon, 7 July 1997

[DVT98-1] Nat Reeves, Level One Communications
*Using Virtual Testers for Test-Vector Verification*
http://www.virtualtest.com/news/eearticle.html
CN-Communications News (July 1998)
Evaluation Engineering article, July 1998

[DVT98-2] Integrated Measurement Systems, Inc.
*Digital VirtualTester Backgrounder*
IMS, Beaverton, Oregon, 1998

[DVT98-3] Integrated Measurement Systems, Inc.
*Digital VirtualTester - Off-Tester IC Pattern & Timing Debug with Turbomodel Emulation Technology*
IMS, Beaverton, Oregon, 1998

[DVT98-4] Integrated Measurement Systems, Inc.
*Success Story - Motorola Leverages Virtual Test for 50% Savings*
IMS, Beaverton, Oregon, 1998

[DVT98-5] Integrated Measurement Systems, Inc.
*Success Story - Virtual Test Drives Major Savings at Level One*
IMS, Beaverton, Oregon, 1998

[DVT98-6] James F. Boettcher, H. Glenn Carson, Jr.
*A Virtual Test Revolution*
Integrated System Design, Dezember 1998

[DVT00] Integrated Measurement Systems, Inc.
*Digital VirtualTester data sheet*
http://www.virtualtest.com/dvtdocs.html
IMS, Beaverton, Oregon, 2000

[EET98] Semiconductor Business News
*HP, IMS team up to create virtual test tools for ATE*
EETimes, 21. Juni 1998

[EIN98] K. Einwich, P. Schwarz, P. Trappe, T. Chambers, G. Krampl, H. Zojer, S. Sattler,
*Virtual Test of complex mixed-signal Telecommunication Circuits reusing System-level Models*
4th International Mixed Signal Testing Workshop (IMSTW),
The Hague, Netherlands, June 1998

[EIN99] K. Einwich, S. Altmann, T. Leitner, R. Hoppenstock, G. Krampl, S. Sattler,
*Applying High-Level Virtual Test to a Complex Mixed-Signal Telecommunication Circuit*
5th International Mixed Signal Testing Workshop (IMSTW), Whistler, Canada, June 1999

[EIN00] K. Einwich, S. Altmann, T. Leitner, G. Krampl, R. Hoppenstock, S. Sattler,
*Application of Virtual Test for high complex Telecommunication Circuits*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli, 2000

[FAS97] Chad Fasca, Dylan McGrath
*Virtual Test Comes Of Age*
Electronic News, November 1997

[FIT97] Michael Fitchett,
*Virtual Test at Motorola*
Virtual Test Workshop, Erlangen, Germany, 10. Juli, 1997

[FIT98] Michael Ftichett, Jim Revie, Andrew Patterson,
*Mixed-Signal Simulation techniques applied to Virtual Test*
4th International Mixed-Signal Testing Workshop (IMSTW),
The Hague, Netherlands, June 1998

[FOR99] Craig Force
*Integrating Design and Test Using New Tools and Techniques*
ISD, Februar 1999

[FOR98] Craig Force, Tom Austin
*Testing the Design: The Evolution of Test Simulation*
International Test Conference, 1998

[GENT03] T. Gentner, H.Beyer, C. Spircu, H. Grams, J.Schuster
*Entwurf einer Ebenenstruktur für die Modellierung einer neuen Testerarchitektur*
ITG, 2003

[GLA00] Wolfram Glauert, Harald Grams
*Status of Virtual Test*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[GOL97] M. Goldbach, W. Glauert, T. Schlaaf, H. Grams
*Simulator Independent Test Program Verification*
3rd International Mixed-Signal Testing Workshop (IMSTW),
Seattle, Washington, USA, June 1997

[GOL98-1] Michael Goldbach,
*Zur Simulation von Prüfprogrammen für Integrierte Schaltungen*
Dissertation, Erlangen, April 1998

[GOL98-2] M. Goldbach, H. Grams, W.Glauert, W. Hartl, G. Voit
*Simulation-Based Test Program Verification Using the SZ Testsystem Environment*
4th International Mixed Signal Testing Workshop (IMSTW),
The Hague, Netherlands, June 1998

[GRA91] Hartmut Grabinksi,
*Theorie und Simulatioon von Leitbahnen*
Springer Verlag, Berlin Heidelberg, 1991

[GRA00] Harald Grams
*Status of the SZ Virtual Test Environment*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[GRM00] Alexander Grassmann
*New Simulation-Techniques*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[GRM01] Alexander Grassmann
*Adaptive Methoden zur effinzienten Mixed-Signal-Simulation von Prüfprogrammen für integrierte Schaltungen* Dissertation, Erlangen, Oktober 2000, VDI-Verlag 2001,
ISBN 3-18-333020-2

[HAR96] Waltraud Hartl
*Tests einfacher programmieren*
Elektronik 8/1996

[HEI99] Ulrich Heinkel
*Formale Spezifikation und Validierung digitaler Schaltungsbeschreibungen mit Zeitdiagrammen*
Dissertation, Erlangen, April 1999

[HEL96-1] Klaus Helmreich,
*How Virtual Test Becomes Reality*
The European Desgin & Test Conference (ED&TC), Paris, France, 11-14 March, 1996

[HEL96-2] K. Helmreich, G. Reinwardt,
*Virtual Test of Noise and Jitter Parameters*
International Test Conference, Washington, USA, 1996

[HEL99] Klaus Helmreich, Armin Lechner,
*Fixturing of High Data Rate Memory ICs for Production Test*
3rd IEEE Workshop on Signal Propagation on Interconnects,
Titisee-Neustadt, Germany, 1999

[HEL00] Klaus Helmreich
*Test Path Modelling and Simulation*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[HEL01] Klaus Helmreich, Armin Lechner,
*RF Signal Integrity Characterisation of Probe Cards*
2001 SouthWest Test Workshop, San Diego, CA, USA, 2001

[HOH99] Georg Hohmann (Hrsg.) *Simulationstechnik*
13. Symposium, Weimar, September 1999

[HOL97] Jim Holmes, Felicia James, Ian Getreu
*Mixed-Signal Modeling for ICs*
Integrated System Design, Juni 1997

[HP96] *HP 75000 Family of VXI Components, Systems, and Services*
Hewlett Packard, 1996 Source Book

[HP97] Hewlett-Packard Company
*HP and IMS Virtual Test Division Announce Software Tools That Bridge the Gap Between MCU Desgin and Test*
Hewlett-Packard Press Release, Washington, November 1997

[IMS99-1] Integrated Measurement Systems, Inc.
*IMS' Virtual Test Division Allies with Teradyne and Motorola to Slash IC Development Time and Test Cost*
IMS, Beaverton, Oregon, 1999

[IMS99-2] Integrated Measurement Systems, Inc.
*Advantest and IMS' Virtual Test Division Form Strategic Alliance to Cut Customer IC Development Time and Test Cost*
IMS, Beaverton, Oregon, 1999

[INT02] Intel Corporation,
*Audio Codec '97, Revision 2.3*
Intel Corporation, April 2002

[ITR03] *International Technology Roadmap for Semiconductors, 2003 Edition, Test and Test Equipment*
http://public.itrs.net/

[KAT99] Keith Katcher
*The Virtues of Virtual Test for Fabless IC Developers*
Fabless Forum, Volume 6 Number 3, September 1999

[KAO92] Wiliam Kao, Jean Xia, Tom Boydston
*Automatic Test Program Generation for Mixed Signal ICs via Design To Test Link*
ITC, 1992

[KHO97] Nash Khouzam
*Simulating Mixed-Signal Tests to Reduce Time-to-Market*
Integrated System Design Magazine, April 1997

[KLE04] Jürgen Kleinöder, Franz J. Hauck
*Übungen zu Middleware*
Universität Erlangen-Nürnberg, Universität Ulm, WS2003/2004

[KLU03] Steffen Klupsch
*Entwurfsmethodik für heterogene Systeme*
Dissertation, Darmstadt, 2004

[KRA99] O. Kraus, M. Miegler, G. Krampl, H. Tauber, S. Sattler, E. Sax
*A Flexible Test Pattern Data Management using High Level Descriptions*
5th International Mixed Signal Testing Workshop (IMSTW), Whistler, Canada, June 1999

[LAN95] Philip Lanchès
*Parallele Logiksimulation - effiziente Methoden für Multicomputer*
Esslingen, 1995, VDI-Verlag, ISBN 3-18-315420-X

[LES95] Gary J. Lesmeister
*A Tester for Design (TFD)*
ITC, 1995

[LTX98-1] LTX
*AMS Selects LTX Fusion for Testing Advanced Mixed-Signals ASICS*
LTX, Press Release, Westwood, 2.Juni 1998

[LTX98-2] LTX
*enVision++: Fusion's advanced test development environment*
LTX, Product Information, Westwood, 1998

[LTX98-3] LTX
*Fusion: One Test Platform 0 Compromises*
LTX, Product Information, Westwood, 1998

[MIC00] microLEX Systems A/S
*Architecture of the Series 20 Test Solution*
http://www.microlex.dk/products/series20.htm
http://www.microlex.dk/products/series20-basic.htm
microLEX Systems A/S, Dr. Neergaards Vej 5C, 2970 Hoersholm, Denmark

[MIE96] M. Miegler, W. Wolz
*Development of Test Programs in a Virtual Test Environment*
14th IEEE VLSI Test Symposium, Priceton, New Jersey, Mai 1996

[MIE99] Max Miegler
*Konzeption einer Datenbasis zur Spezifikation und Generierung von Prüfprogrammen für gemischt analog-digitale integrierte Schaltungen*
Dissertation, Erlangen, 1999, VDI-Fortschritt-Berichte, ISBN 3-18-330520-8

[MOV03] Glauert, Miegler *MoVIS - Modellierung und Verifikation integrierter Schaltungen*
Vorlesung - LRS (S. 3-28, 3-29)

[NOV94] John Novellino
*Software Targets Test Bottleneck*
Electronic Design, 7. November 1994

[NOW00] E. Nowak
*Practical Results of Using Virtual Test*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[OON98] John Oonk
*Leveraging New Standards in ATE Software*
International Test Conference, 1998

[PAG88] B. Page, R. Bölckow, A. Heymann, R. Kadler, H. Liebert
*Simulation und moderne Programmiersprachen*
Springer-Verlag, Berlin, 1988

[PAN00] Neel Pandeya
*Enahncing Pre-Silicon Debug Productivity with Test Simulation*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[PAT97] Andrew Patterson,
*Virtual Test with Analogy and LTX*
Virtual Test Workshop, Erlangen, Germany, 10. Juli, 1997

[PAU97] Ed Paulsen
*Is Virtual Test the Real Thing?*
LTX-Online, 1997

[PRO98] Dan Proskauer
*High Quality, Easy to Use, On Time Software...Can it be done?*
International Test Conference, 1998

[REI02] Gerd Reinwardt, Filippo Borgonovo
*Hart an der Grenze*
Elektronik, 19/2002

[REV97] Jim Revie,
*LTX's vision for Virtual Test*
Virtual Test Workshop, Erlangen, Germany, 10. Juli, 1997

[RIO99] JJ O Riordan,
*Design of a Test Simulation Environment for Test Program Development*
ITC '99, Atlantic City, USA

[ROL97] Dave Rolince
*Applying Virtual Test Principles to Digital Test Program Development*
http://www.teradyne.com/prods/cbt/products/library/lasar/wp_atc_97_lasar_rolince.pdf

[RON00] Marco Rona
*Model and Simulator Coupling for Virtual Test*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[RON01] Marco Ronta, Gunter Krampl
*A VHDL-based Virtual Test Concept for Pre-Silicon Test-program Debug*
European Test Workshop 2001, Stockholm, Schweden, 29. Mai - 1. Juni 2001

[ROS89] Eric Rosenfeld
*Issues for Mixed-Signal CAD-Tester Interface*
ITC, 1989

[ROT97] Franz Rottner
*Automatic test equipment links design and production*
Electronics Engineer, September 1997

[SAB04] Synopsys,
*Saberbook - Online Hilfe*

[SAN99] Rick K. Sanborn
*A Methodology for Analog Fault Modeling and Diagnosis in Mixed-Signal Virtual Test*
5th International Mixed Signal Testing Workshop (IMSTW), Whistler, Canada, June 1999

[SAV97] Peter Savage, Steve Walters, Mark Stephenson
*Automated Test Methodology for Operational Flight Programs*
Proceedings of IEEE Aerospace Conference, Vol.4, pp 293-305, 1997

[SCH80] Günther Schmidt
*Simulationstechnik*
Oldenbourg-Verlag, München, 1980

[SCH01] Michael Schlegel, Frank Irmisch
*Standard mit Zukunft: VHDL-AMS*
Elektronik Auomotive, September 2001

[SCH02] Johann Schlichter
*Distributed Applications, Student Script*
Institut für Informatik, TU München, März 2002

[SIE98] Craig Siegel
*The True Value of Virtual Test Software*
Integrated System Design, Dezember 1998

[SIE99] Craig Siegel
*Virtual Test of Mixed-Signal ICs: Two Approaches*
5th International Mixed Signal Testing Workshop (IMSTW), Whistler, Canada, June 1999

[SHO98] Lee A. Shombert, John W. Sheppard
*A Behavior Model for Next Generation Test Systems*
Journal of Electronic Testing: Theory and Applications 13, 299-314,
Kluwer Academic Publishers, Niederlande, 1998

[SIG98] SigmaTel Inc,
*STAC9701/3, Multimedia Audio Codec for AC97*
SigmaTel, Inc., Austin, Texas, September 24, 1998

[SPI85] Hans Spiro
*Simulation integrierter Schaltungen durch universelle Rechnerprogramme*
Oldenbourg-Verlag, München, 1985

[STI03] *IEEE 1450 - Standard Test Interface Language (STIL)*
http://grouper.ieee.org/groups/1450/

[STR94] Dan Strassberg
*It' a New Ball Game*
EDN, 29. September 1994

[SZ00] SZ Testsysteme AG,
*Technical Documentation, ,Concept of Test System*
02.05.2000, Amerang, Germany

[TAC99] Test Technology Technical Council
*Technical Activity Comitee on Verification and test*
TAC Chair Jacob Abraham
Internet: http://www.computer.org/tab/tttc/Activities/ver_test.html, 7. Dezember 1999

[TAY93] Tony Taylor
*Tools and Techniques for Converting Simulation Models into Test Patterns*
International Test Conference, 1993

[TDI97-1] Press release
*IMS' Virtual Test Division Announces New IC Test Pattern Generation Solution*
IMS, Beaverton, Oregon, March 31, 1997

[TDI97-2] Integrated Measurement Systems, Inc.
*Test Direct White Paper - Digital Virtual Test for Mixed-Signal*
IMS, Beaverton, Oregon, 1997

[TDI97-3] Integrated Measurement Systems, Inc.
*Test Direct, Turbo Bridge*
IMS, Beaverton, Oregon, 1997

[TDI00] Integrated Measurement Systems, Inc.
*TestDirect/TurboBridge data sheet*
http://www.virtualtest.com/tddocs.html
IMS, Beaverton, Oregon, 2000

[TEK11] *VXI plug&play ist ein Erfolg*
TekView Vol. 11, Nr. 2, S. 10-11,

[TEK96] *Tektronix VXI-Tutorial-Disk*

[TEN00] Wilfried Tenten
*An envisaged route to get best fitted test signals*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[TEN02] Wilfried Tenten, Robert Bosch GmbH
*Evaluierung der VIRTUS SPACE Software - Protokoll*
28.11.2002, Reutlingen, Germany

[TER97-1] Teradyne
*Introducing: The VX Family of Test Simulation Tools*
VX Demonstration, Productonica, 1997

[TER97-2] Teradyne
*VX Demonstration - Simulation for Test Engineers*
VX Demonstration, ITC, 1997

[TER97-3] Teradyne
*Teradyne Introduces Next Generation of Test Simulation Software:*
*VX Family Provides Fully Integrated, Closed-Loop Simulation for Test Engineers*
Teradyne Online, Boston, Massachusets, November 1997

[TER98] Teradyne
*Teradyne introduces Digital VX Test Simulation Software*
ITC, 1998, Washington

[THY99] *Thesys Design-Flow*
http://www.thesys.de/asic/asic_flow.htm
16. März 1999

[TIE93] U. Tietze, Ch. Schenk
*Halbleiterschaltungstechnik*
Springer Verlag, Erlangen 1993

[VTE98] Presse-Information
*IMS Delivers Virtual Test Emulator[tm] Software for Off-Line Test Program Debug*
http://www.virtualtest.com/news/vtepr.html
IMS, Beaverton, Oregon, June 29, 1998

[VTE99] Integrated Measurement Systems, Inc.
*Virtual Test Emulator Datasheet*
IMS, Beaverton, Oregon, 1999

[VTE00-1] Integrated Measurement Systems, Inc.
*Virtual Test Emulator data sheet for V1000*
http://www.virtualtest.com/vtedocs.html
IMS, Beaverton, Oregon, 2000

[VTE00-2] Integrated Measurement Systems, Inc.
*Virtual Test Emulator data sheet for Kalos*
http://www.virtualtest.com/vtedocs.html
IMS, Beaverton, Oregon, 2000

[VXI96] *Charter Document*
The VXI plug&play System Alliance

[WEB89] Bruce Webster
*An Integrated Analog Test Simulation Environment*
ITC, 1989

[WEB03] Jürgen Weber, Mario Anton, Sorin A. Huss
*Verhaltensmodellierung von Ein- und Ausgangsstufen für den Virtuellen Test von Mixed-Signal Automotive Schaltkreisen*
Analog'03, Heilbronn, 10.-12.September 2003

[WIK04] Wikipedia
*Wikipedia, die freie Enzyklopädie*
http://de.wikipedia.org

[WIL00] Jens Willibald
*A general test signal description environment*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[WOL96] W. Wolz, M. Miegler, M. Goldbach, H. Hamberger, B. Leykauff, S. Böttcher
*Virtual Test - new aspects of Design & Test Integration*
2nd International Mixed Signal Testing Workshop (IMSTW),
Quebec City, Canada, Mai 1996

[WOL00] Werner Wolz
*Perspectives of future ATE*
2nd Virtual Test Workshop, Erlangen, Germany, 12. Juli 2000

[XIA95] Jean Qincui Xia, Tom Austin, Nash Khouzam
*Dynamic Test Emulation for EDA-Based Mixed-Signal Test Development Automation*
ITC, 95

[ZAM98] Naveed Zaman, Antony Spilman,
*Triggering and Clocking Architecture For Mixed Signal Test*
ITC'98

# Lebenslauf

## Persönliche Daten

| | |
|---|---|
| Name | Harald Grams |
| geboren am | 16. Februar 1970 |
| in | Garmisch-Partenkirchen |

## Werdegang

| | |
|---|---|
| 1976 - 1980 | Grundschule Stein |
| 1980 - 1989 | Gymnasium Stein |
| 1990 - 1996 | Studium der Elektrotechnik<br>Friedrich-Alexander-Universität Erlangen-Nürnberg |
| 1996 - 2004 | Wissenschaftlicher Mitarbeiter am<br>Lehrstuhl für Rechnergestützten Schaltungsentwurf der<br>Friedrich-Alexander-Universität Erlangen-Nürnberg |