M U L T I P R O J E K T
C H I P - G R U P P E

B A D E N - W Ü R T T E M B E R G

W O R K S H O P J U N I 1 9 9 1

E S S L I N G E N / G Ö P P I N G E N

H E R A U S G E B E R : F A C H H O C H S C H U L E U L M

INHALTSVERZEICHNIS

Diplomarbeit aus der Schaltungsintegration

1.

"Integration eines Viterbi-Decoders auf einem ACTEL-FPGA"

am Labor fuer Schaltungsintegration

an der Fachhochschule Ulm

im Auftrag der AEG Mobile Communication, Ulm

Bearbeiter:	Dieter Wengler
Fachbereich:	Technische Informatik
Zeitraum:	Sommersemester 1991
Betreuer:	Prof. Fuehrer (Fachhochschule)
	Dr. Kappei (AEG Mobile Communication)

TABLE OF CONTENTS

TABLE OF CONTENTS [continued]

LIST OF FIGURES

1. Vorwort

Das Thema zur vorliegenden Diplomarbeit wurde von der AEG Mobile Communication, Ulm gestellt. Der VITERBI-Decoder stellt einen Teil eines Projektes dar, bei dem das Mobilgeraet fuer ein zukuenftiges europaweites, mobiles, digitales Kommunikationssystem entwickelt wird. Neben einer besseren Frequenzauslastung, hoeheren Uebertragungsgeschwindigkeiten und einem verbesserten Angebot an Uebermittlungsdiensten soll das Digitalsystem auch in Bezug auf Abhoersicherheit und Fehlererkorrekturverfahren neue Masstaebe setzen. Grundlage der Entwicklungen stellen die sogenannten GSM-Vorschriften dar, auf die unten etwas naeher eingegangen wird. Meine Arbeiten gliederten sich in einen umfangreichen Einarbeitungsteil im Hause AEG sowie in den Realisierungsteil an den Rechnern des Labors fuer Schaltungsintegration an der Fachhochschule Ulm.

2. Einfuehrung zum GSM-System

2.1 Allgemeines

Die GSM*, der viele bedeutende Unternehmen aus der Kommunikationsindustrie angehoeren, stellt ein internationales Normungsgremium dar. Ihre Aufgabe besteht darin, umfangreiche Spezifikationen fuer das gesamte Uebertragungsverfahren im Digitalen Kommunikationssystem mit Kanaldefinition, Frequenzbelegungen, Uebertragungsprotokollen, Codierungs- und Decodierungsverfahren etc. zu erstellen.

2.2 Kanalstrukturen im GSM-System

Allgemeines

Im GSM-System findet eine digitale Phasenmodulation (GMSK) statt. Bei der Uebertragung werden Informationsbloecke gleichzeitig auf mehreren Frequenzen im Zeitscheibenverfahren gesendet (TDMA/FDMA**). Hierbei werden einer Traegerfrequenz innerhalb von 8 aufeinanderfolgenden Zeitscheiben 8 verschiedene logische Informationskanaele zugeordnet. Man spricht in diesem Zusammenhang beim Senden von einem Mapping der logischen Kanaele auf physikalische.
Dem Anwender wird nicht nur die Uebertragung von digitalisierter Sprache, sondern auch von anderen Daten (FAX etc.) mit einer Vielzahl von Uebertragungsraten angeboten. Diese verschiedenen Dienste werden auch auf gesonderten logischen Kanaelen abgewickelt., die sich in

- Art der uebertragenen Informationen
- Anteil der Nutzinformationen im Datenstrom
- Belegung der Frequenzbaender
- Interleaving- und Codierungsschema

voneinander unterscheiden.

* Group Speziale Mobile

** Time Devision Multiple Access/Frequency Devision Multiple Access

Codierung eines Sprachkanales

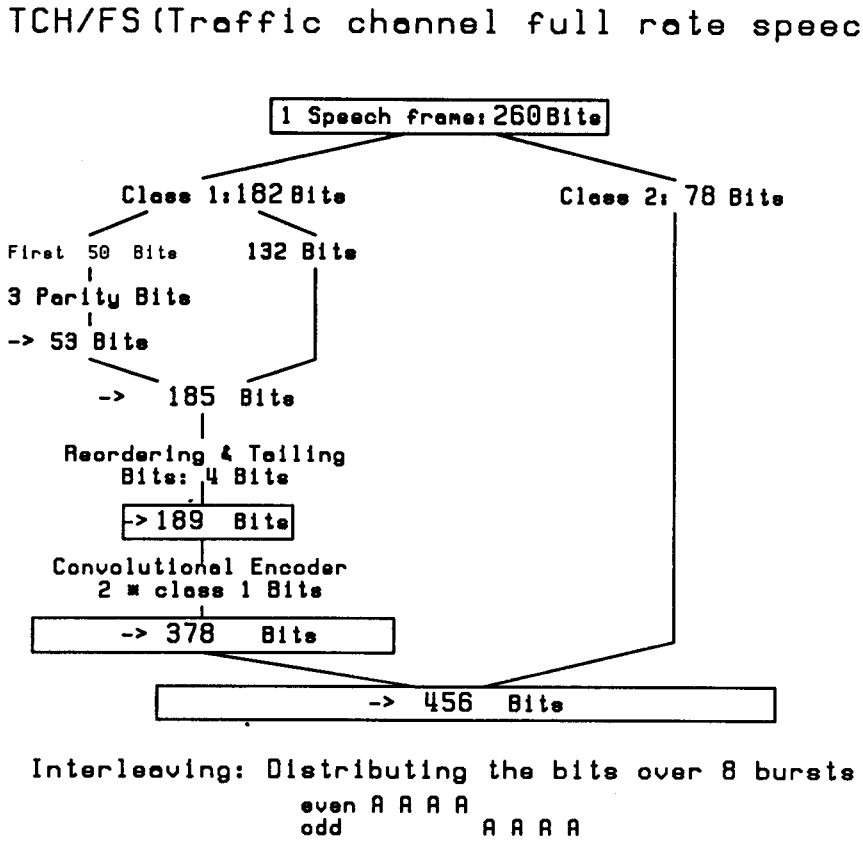


Figure 1 Channel scheme

Hier sei beispielsweise ein bestimmter Sprachkanal betrachtet, bei dem die Laenge der zu uebertragenden Informationsbloecke 260 Bit betraegt. Dieser Block wird in Bitgruppen unterschiedlicher Wichtigkeit unterteilt, den sogenannten Class 1 Bits und Class 2 Bits. Die ersten 50 Bits der Class 1 werden mit 3 Paritaetsbits zu 53 Bits ergaenzt, an sie werden die 132 in dieser Stufe noch ungeschuetzten Bits angehaengt. Nachdem diese 185 Bits durch Anhaengen von 4 Bits als Ende-Kennung auf eine Laenge von 189 Bits erweitert wurden, wird durch eine Faltungscodierung mit der Coderate 1/2 eine Verdopplung der Stellenzahl erreicht. Diese Redundanz ist fuer eine Fehlererkennung und -Korrektur bei der Decodierung notwendig. Zu diesen 378 durch Block- und Faltungscode geschuetzten Bits werden die 78 voellig ungeschuetzten aus dem rechten Zweig der Darstellung wieder hinzugefuegt.
Dieser Informationsblock von 456 Bits wird jedoch nicht auf einer Frequenz, sprich einem physikalischen Kanal gesendet, sondern im obigen Beispiel in Bloecken von 47 Bits abwechselnd auf zwei verschiedenen physikalischen Kanaelen. Sollte also bei der Uebertragung eine Frequenz zeitweilig gestoert werden, so gehen im worst case nur die Haelfte der Codebits verloren. Da somit systembedingt keine Bueschelfehler auftreten koennen, lassen sich die verlorengegangenen Bits bei der Decodierung meist fehlerfrei rekonstruieren. Dieses Interleaving-Schema ist ebenfalls fuer jeden logischen Kanal separat festgelegt.

2.3 Darstellung und Arbeitsweise von Faltungscodierern

Um die Arbeitsweise des VITERBI-Decoders besser zu verstehen, soll hier anhand eines **stark vereinfachten** Beispieles kurz auf die Faltungscodierung und -Decodierung eingegangen werden, wie sie innerhalb des GSM-Systems Anwendung findet.

2.3.1 Darstellung von Faltungscodierern

Ein Informationszeichen bestehe aus K Bit. Ein Faltungscode wird erzeugt durch ein Schieberegister der Laenge (M+1)*K und N linearen Verknuepfungsgliedern. Pro Schritt wird ein Zeichen (K Bit) in ein Schieberegister geschoben und zusammen mit den uebrigen (M*K) Bit des Registers zu N Codebits verknuepft. Die Coderate ist dann R = K/N und M*K nennt man das Gedaechtnis des Codierers. Es sind 2^{K*M} verschiedene Gedaechtnisinhalte oder Zustaende moeglich.

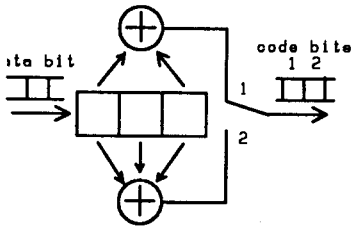


Figure 2 Funktionsbild: Convolutional Encoder mit 2 Gedaechtnisstellen

Beispiel:

Informationslaenge	K = 1
Gedaechtnislaenge	M = 2
Codebits	N = 2
Coderate	R = 1/2
Anzahl der moeglichen Zustaende	4

Somit werden aus einem Informationsbit zwei Codebits erzeugt, die nicht nur vom Eingabebit, sondern auch von den beiden Vorgaengerbits abhaengen. Damit koennen in einem bestimmten Zustand nur ganz bestimmte Zustandsuebergaenge auftreten, wie im folgenden Netzdiagramm zu sehen ist. Man sagt auch, der Codierer hat ein 'Gedaechtnis'. Wird beispielsweise bei einem Schieberegister-Startwert von '000' ein Informationsbit '0' in den Codierer geschoben, so ergibt die zwei- beziehungsweise dreifach XOR-Verknuepfung eine Codebitfolge von '00', beim Einschieben einer '1' ergeben sich beim Ausgangszustand '00' die Codebits '01'.
Dieses Wissen ueber die Zulaessigkeit nur ganz bestimmter Uebergaenge wird bei der Decodierung im VITERBI-Decoder herangezogen, um verbotene Zustandswechsel als Fehler zu erkennen und zu korrigieren.

2.3.2 Darstellung der Codierung/Decodierung von Faltungscodierern

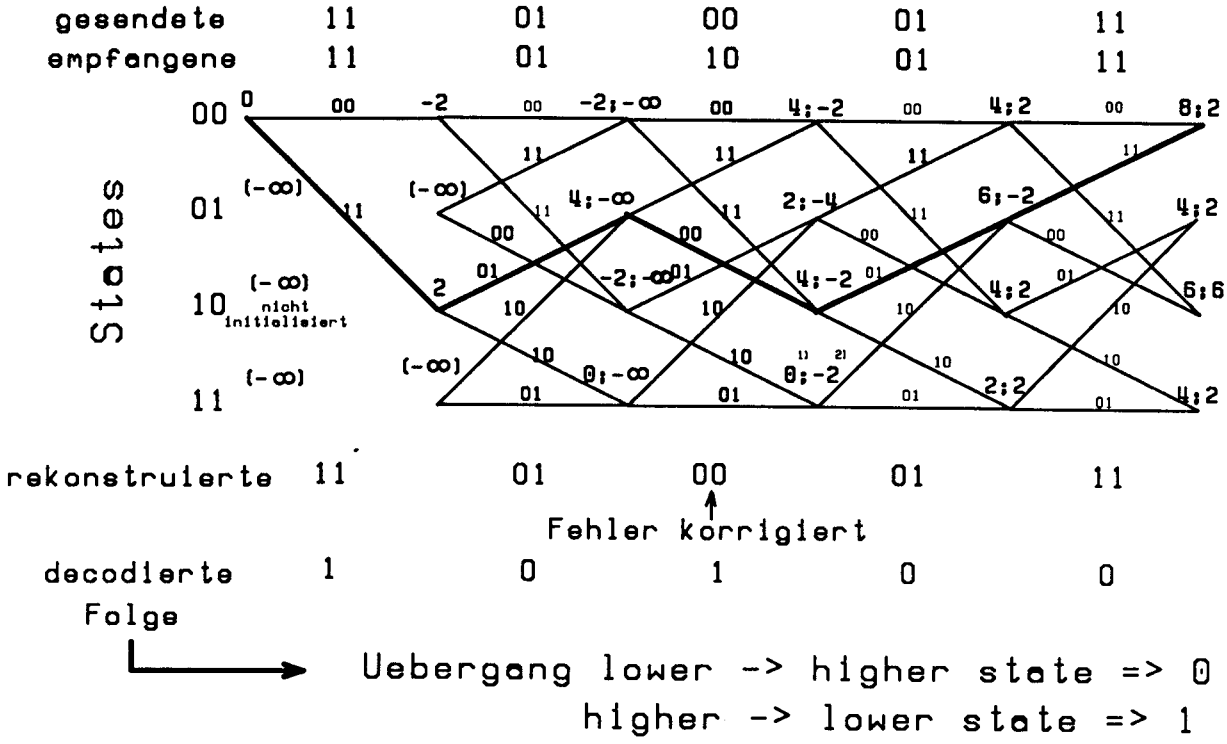


Figure 3 Ausschnitt aus dem Netzdiagramm (Trellis)

Im obigen Bild sind fuer einen Ausschnitt von fuenf Zustandswechseln saemtliche moeglichen Uebergaenge in Form von Linien eingezeichnet. Hierbei sind unter der Bezeichnung 'States' untereinander die vier moeglichen Zustaende und nach rechts hin die einzelnen Zustandswechsel aufgetragen. Jeder Zustandswechsel beschreibt ein Informationsbit, die gesendeten, empfangenen, rekonstruierten und decodierten Binaerfolgen sind entsprechend angegeben.
Jeder Zustandsuebergang (Linie) ist mit den beiden Codebits bezeichnet, die bei der Durchfuehrung dieses Ueberganges generiert werden.

Codierung:

Ausgehend vom Zustand '0' (State '00'), gelangt man durch Einschieben von '0' in den Folgezustand '0', (Codebits '00'), das Einschieben einer '1' wuerde einen Zustandswechsel nach '10' (Codebits '11') bewirken. Aus dem Zustand '10' gibt es jedoch keinen direkten Uebergang zurueck zum Zustand '00', es besteht nur die Moeglichkeit, durch Einschieben von binaeren Nullen ueber den Zustand '01' wieder in den 'Grundzustand' zu gelangen etc. Auf diese Weise lassen sich die Informationsbits als eine Folge von Zustandswechseln interpretieren.

Decodierung

Die Decodierung gliedert sich in zwei Schritte:

1. Berechnung der Trellis-Table in der TRTC*-Unit

Ausgehend vom Zustand '00', werden fuer jeden Folgezustand Wahrscheinlichkeitswerte (Metriken) berechnt, die ein Mass fuer sein Auftreten darstellen und sich aus der Summe der Metrik des Vorgaengerzustandes und der Delta-Metrik des betreffenden Zustandswechsels ergeben.
Dazu werden paarweise die empfangenen Codebits mit den Codebits verglichen, die beim Uebergang aus dem einen Zustand beim Start beziehungsweise den beiden Vorgaengerzustaenden erzeugt werden wuerden. Der Delta-Metrik-Wert fuer einen Uebergang errechnet sich hierbei aus der Anzahl der uebereinstimmenden Bits abzueglich der Zahl der Nicht-Uebereinstimmungen. Die akkumulierten Metriken berechnen sich nun aus der Summe von akkumulierter Metrik des Vorgaengerzustandes und der Delta-Metrik, die sich beim Uebergang vom Vorgaengerzustand zum aktuellen Zustand ergibt. Da jeder Zustand zwei moegliche Vorgaenger besitzt, werden pro Zustand auch zwei akkumulierte Metriken berechnet. Diese werden miteinander verglichen und die groessere zu diesem Zustand abgespeichert.
Die Groesse der Metrikwerte nimmt also mit steigender Informationsbit-Nummer innerhalb des zu decodierenden Blockes zu.
Die Initialisierungswerte der Metriken fuer die Zustaende '01' bis '11' betragen definitionsgemaess minus undendlich. Die Metrik des Anfangszustandes wird mit '0' initialisiert..

Beispiel: Ausgangspunkt sei der Zustand '00' mit der Metrik '0'. Die empfangene Codefolge '11' wird mit den beiden Codefolgen '00' und '11' verglichen, die bei den moeglichen Uebergaengen in die Folgezustaende '00' und '10' auftreten koennen. Im ersten Fall ergibt sich fuer beide Binaerstellen keine Uebereinstimmung, entsprechend einer Delta-Metrik von '-2'. Zusammen mit der akkumulierten Metrik '0' des Zustandes '00' wird dem Folgezustand eine akkumulierte Metrik von '-2' zugeordnet. Beim zweiten moeglichen Uebergang in den Folgezustand '10' ist die Codefolge des Ueberganges mit der empfangenen in beiden Binaerstellen identisch, es ergibt sich eine Delta-Metrik von '+2'. Wuerden bei einem Zustandswechsel Uebergangs-Codefolge und empfangene Folge in einem Bit uebereinstimmen und im anderen voneinander abweichen, so ergaebe sich eine Delta-Metrik von '0', wie beispielsweise beim Uebergang vom Zustand '00' in den Folgezustand '00' beim zweiten dargestellten Zustandsuebergang.

Die Realisierung der TRTC-Baugruppe, die diese Funktionalitaet realisiert, war die Aufgabe dieser Diplomarbeit.

Es ist hervorzuheben, dass im GSM-System mit Faltungscodierern bis zur Gedaechtnislaenge 4 gearbeitet wird, womit sich maximal 16 verschiedene Zustaende codieren lassen. Der praktische Aufwand ist somit um ein Vielfaches hoeher, als im obigen Beispiel dargestellt werden konnte.

* Trellis Table Computation

2. Zurueckverfolgung der Trellis Table in der TB-Unit*

Wurde fuer einen ganzen Nachrichtenblock diese Tabelle mit den Zustandsuebergangs-Wahrscheinlichkeitsinformationen erstellt, so kann nun, beginnend mit dem letzten Informationszeichen der Pfad mit denjenigen Zustandsuebergaengen zurueckvervolgt werden, dessen Zustaende jeweils die hoechste Metrik aufweisen. Hierbei werden Pfade mit unmoeglichen Zustandswechseln nicht weiter verfolgt. Aus den Zustandsuebergaengen leiten sich gemaess Darstellung unmittelbar die decodierten Informationsbits ab.

2.3.3 Vorteile des Codierungs- und Decodieruns-Verfahrens

* Vermeidung uebertragungsbedingter Bueschelfehler durch Senden der Codierinformationen auf verschiedenen Frequenzen (FDMA)
* Erkennung und Korrektur von verbotenen Zustandsuebergaengen bei der Decodierung
* Beaufschlagung der empfangenen Codebits mit Wahrscheinlichkeitswerten
  -> ***Soft Decision <=> Hard Decision (siehe 2.4.1!)***

2.4 Der VITERBI-Decoder im vorliegenden Anwendungsfall

2.4.1 Die Arbeitsumgebung des VITERBI-Decoders

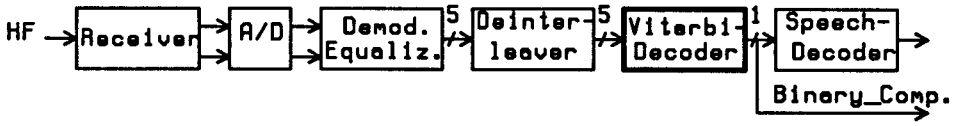


Figure 4 Blockschaltbild Empfaengerbaugruppe

Wie aus dem stark vereinfachten Bild zu ersehen ist, stellt der VITERBI-Decoder eines der letzten Glieder in der empfangsseitigen Signalflusskette dar. Er decodiert jeweils die Codebits eines kompletten Informationsblockes nacheinander, bevor er einen anderen Block decodiert. Dazu erhaelt er vom Demodulator zyklisch eine Sequenz von Codebits, die zusammen ein Informationsbit verschluesseln. Diese Codebits sind jedoch keine 'harten' Entscheidungswerte, sondern sind wiederum mit einer Wahrscheinlichkeitsinformation behaftet, die aussagt, dass das Codebit mit einer Wahrscheinlichkeit von +/-0,125 bis +/-0,8 eine 0 (-) oder eine 1 (+) ist (5 Bit binaercodiert). Somit verkoerpert jedes in der Trellis Table dargestellte Codebit in Wirklichkeit ein 5 Bit-codierte Festkommazahl. Ein Informationsbit wird also durch mehrere Codebits verschluesselt, die wiederum Dezimalwerte sind. Demnach handelt es sich auch bei den zu den Zustaenden zugeordneten Metriken oder Wahrscheinlichkeiten um Festkommazahlen. In diesem Zusammenhang spricht man auch von *Soft Decision Values.*

* Trace Back

2.4.2 Besondere Anforderungen

Im Labor wird gegenwaertig noch eine softwaremaessige Implementierung der Decodierungsalgorithmen angewendet. Wegen der enormen geforderten Taktraten erschien jedoch laengerfristig nur eine Realisierung in festverdrahteter Logig als sinnvoll.
Als Labormuster wurde hier auf ein FPGA zurueckgegriffen, um eine kostenguenstige anwenderspezifische Programmierung von Prototypen zu ermoeglichen. Fuer spaeter wird eine Realisierung auf einem in groesseren Stueckzahlen kostenguenstigeren Gate Array beabsichtigt.
Bei der Codierung wird ein Faltungscodierer der Gedaechtnislaenge 4 verwendet, womit 16 verschiedene Zustaende auftreten koennen.
Die TRTC-Unit muss ausserdem parametrierbar sein bezueglich der Laenge der zu decodierenden Informationsbloecke sowie der Coderate (1/2, 1/3 und 1/6). Sie wird im Polling-Mode durch einen externen Digitalen Signalprozessor mit Codevektoren versorgt, der auch die berechneten Metriken fuer die Trellis Table wieder vom Ausgangsregister abholt.
Die Gesamtschaltung wurde **synchron** aufgebaut.

3. Das ACTEL-Entwicklungspaket

Die ACTEL-Entwicklungsumgebung besteht aus: ALS*
ACTEL Hard- und Soft-Macro-Bibliothek

Mit der Entwicklung zweier Bausteinfamilien reagierte ACTEL auf die zunehmende Verbreitung von FPGA's auf Kosten der dominierenden Gate Arrays. Hierbei vereinigen FPGA's die Vorteile der Masked Gate Arrays, wie hohe Packungsdichte, hohe Systemgeschwindigkeiten, Flexibilitaet und Eignung fuer automatisiertes Design mit denen der PLD's, wie Anwenderprogrammierbarkeit, guenstige Lagerhaltung und geringe Durchlaufzeiten in sich.
ACTEL liefert zusammen mit seinem Designpaket das Werkzeug ALS zum anwenderspezifischen Bausteindesign sowie eine Bibliothek von Hard- und Soft-Macros zum Schaltungsentwurf aus.

* Action Logic System

3.1 Die ALS-Entwicklungswerkzeuge

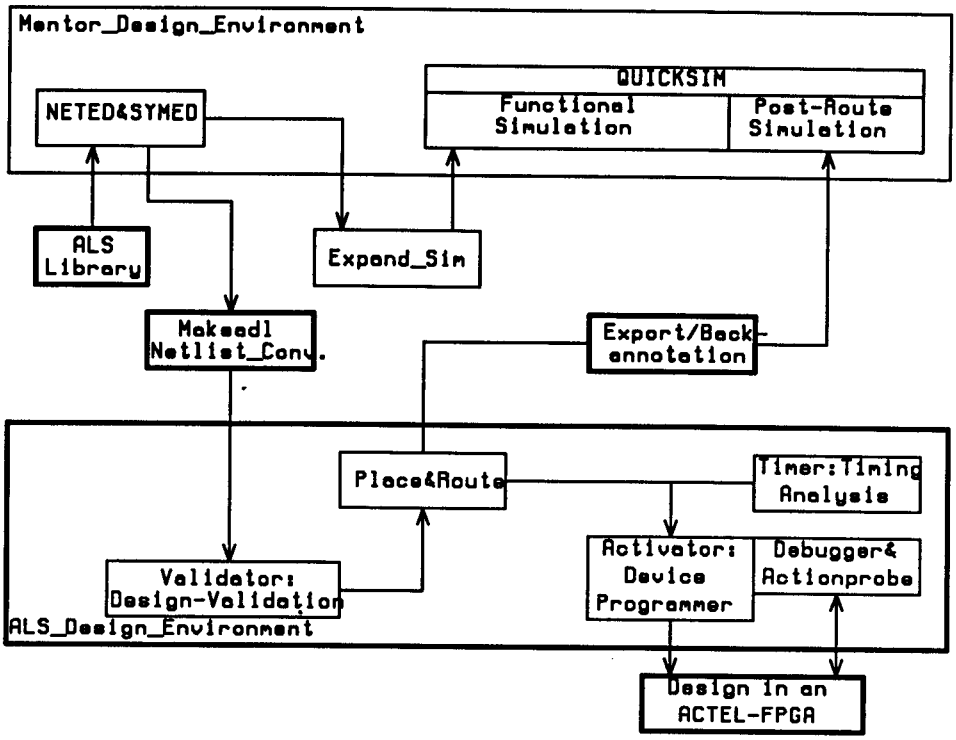
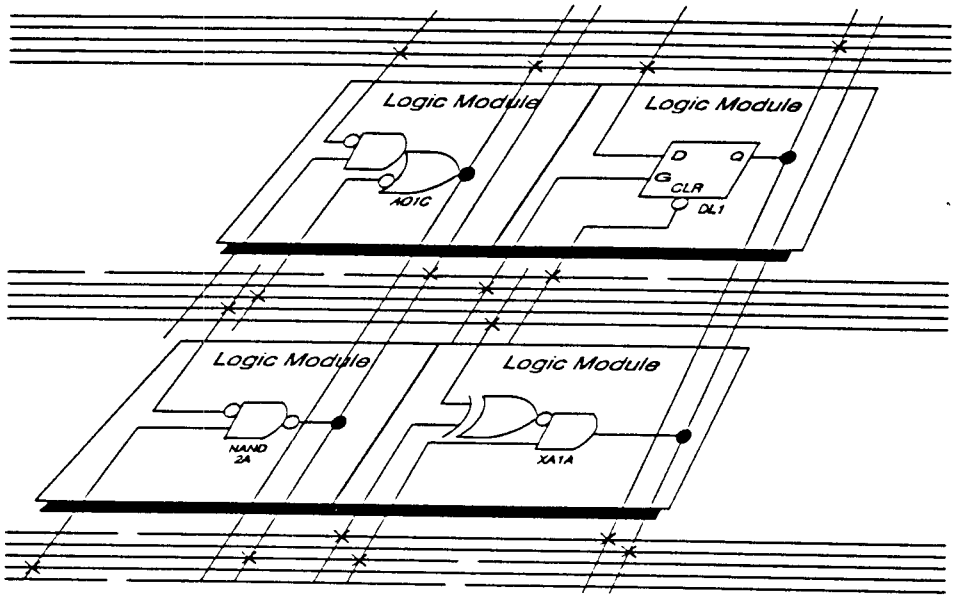


Figure 5 Desing Flow

Den folgenden Ausfuehrungen wird die MENTOR-Umgebung mit ihren spezifischen Werkzeugen zugrunde gelegt. Der Entwurfsprozess beginnt mit dem hierarchischen Stromlaufplanentwurf unter Zuhilfenahme des Programmes 'NETED' unter Verwendung der ACTEL-Bibliothek. Die zugehoerigen Blocksymbole werden parallel dazu mit dem Symbolgenerator 'SYMED' entworfen. Aus den auf diese Weise erzeugten Sheet-Files koennen durch den Aufruf von 'Expand_sim' Netzlisten fuer den Simulator 'QUICKSIM' generiert werden. Der Entwickler kann sich nun Stimuli-Files ueberlegen, in denen er sich mehrere Eingangs-Testvektoren formuliert, die zu bestimmten Simulationszeitpunkten an die Schaltung angelegt werden. Die zu bestimmten Zeitpunkten erwarteten Simulationsergebnisse koennen automatisch abgeprueft werden. Nachdem die Logiksimulation wunschgemaess verlaufen ist, kann durch Aufruf der ACTEL-Kommandoprozedur 'MAKEADL' eine Netzliste fuer die weitere Bearbeitung der Schaltung unter ALS erzeugt werden. Daran schliesst sich eine Entwurfsregelkontrolle an, bevor beim automatischen Plazieren und Verdrahten (Optimierung nach minimalen Verzoegerungszeiten oder geringen Leitungslaengen) das Schaltungslayout festgelegt wird. Die Post Layout Timing Analyse liefert Informationen ueber kritische Pfade und gibt Auskunft ueber die erreichte Systemgeschwindigkeit.
Laut Herstellerangaben kann wegen der enormen bereitgestellten Verdrahtungsressourcen eine Modulausnutzung von 85-95% erreicht werden.
Daran schliesst sich die Programmierung des Bauteiles mit der Activator Hardware sowie optional dessen Test im Programmieradapter oder im Zielsystem an.

3.2 Aufbau eines ACTEL-Bausteines



Das ACTEL-FPGA besteht aus den aussen angeordneten Randzellen als Schnittstelle zur Aussenwelt ueber die Pins und aus einem inneren Bereich, der aus vorbereiteten Logikmodulen und horizontalen und vertikalen Leitungen besteht, die im Origginalzustand noch keine Verbindung miteinander haben.

3.3 Die ACTEL-Bausteine

ACTEL vertreibt zwei Bausteinfamilien ACT1 und ACT2 mit unterschiedlichen Technologien:

Features	A1010	A1020	A1280
Gate Count	1200	2000	8000
Gate Utilization	85-95%	85-95%	90-100%
User-I/O	49	59	140
Logic Modules	295	547	1232
Preis ca. (ohne Mwst.)	65.- DM	120.- DM	?

Momentan sind jedoch nur Bausteine der ACT1-Familie lieferbar und programmierbar. Der A1280 soll neben weiteren Bausteinen im Herbst diesen Jahres erhaeltlich sein.

Hier noch einige technische Daten:

- Realisierung in 1,2u(ACT1) - bzw. 2u(ACT2)-Technik
- PLICE-Antifuse-Technik
- Widerstand programmiert: < 1k Ohm, unprogrammiert:> 100 MOhm
- verschiedene Gehaeuseformen: 44/68/84-pin chip carrier, 84-pin grid array
- fuer verschiedene Spezifikationen erhaeltlich (military, industrial, commercial, ...)
- 1 oder 2 symmetrische Clock-Netzwerke mit unbegrenzter Treibfaehigkeit.

3.4 Aufbau der ACTEL-Library

Die gesamte ACTEL-Bibliothek besteht aus Hard- und Softmacros unterschiedlicher Komplexitaet und unterschiedlichen Flaechenbedarfs auf dem Baustein.

3.4.1 ACTEL-Hardmacros

Bei den ACTEL-Hardmacros handelt es sich um ca. 200 verschiedene Makrozellen, die Gatter einfacher Funktionalitaet, wie beispielsweise NAND, NOR, MULTIPLEXER, Komplexgatter oder Flip-Flops darstellen.

Das elementare ACTEL-Logikmodul 'ACTMOD'
Alle Hardmacros sind aus der elementaren ACTEL-Macrozelle 'ACTMOD' aufgebaut, die auf dem Baustein eine Flaechenzelle benoetigt. Man sagt auch, sie hat einen Modul Count* von 1 (MC = 1). Die ACTEL-Macrozelle ist definiert als 'Black Box' mit 8 Eingaengen und einem Ausgang.

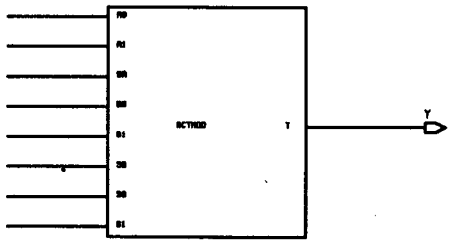


Figure 6 Blockschaltbild des ACEL-Moduls 'ACTMOD'

Die logische Funktion saemtlicher Hardmacros wird ausschliesslich durch eine bestimmte Eingangsbeschaltung von 'ACTMOD' mit Masse-, Versorgungsspan- nungs- oder Signalleitungen eingestellt.

Aussenbeschaltung des ACTEL-Moduls fuer ein Zweifach-NAND-Gatter

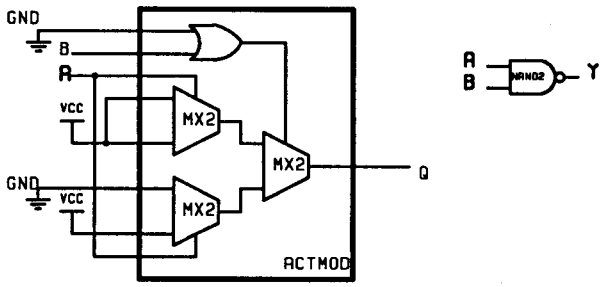


Figure 7 Konfiguration von 'ACTMOD' als 'NAND2'

* Modul Count: Anzahl der fuer eine bestimmte Funktion auf dem Baustein benoetigten ACTEL-Module

Die Hardmacro-Bibliothek besteht somit aus Beschreibungen der individuellen Eingangsbeschaltungen des universellen Moduls 'ACTMOD' zur Realisation einer bestimmten logischen Funktion, wie hier eines NAND-Gatters.
Die benoetigten ACTEL-Hardmacros werden bei der Programmierung eines Bausteines aus 'ACTMOD'-Modulen hergestellt.

Das Simulationsmodell fuer 'ACTMOD'

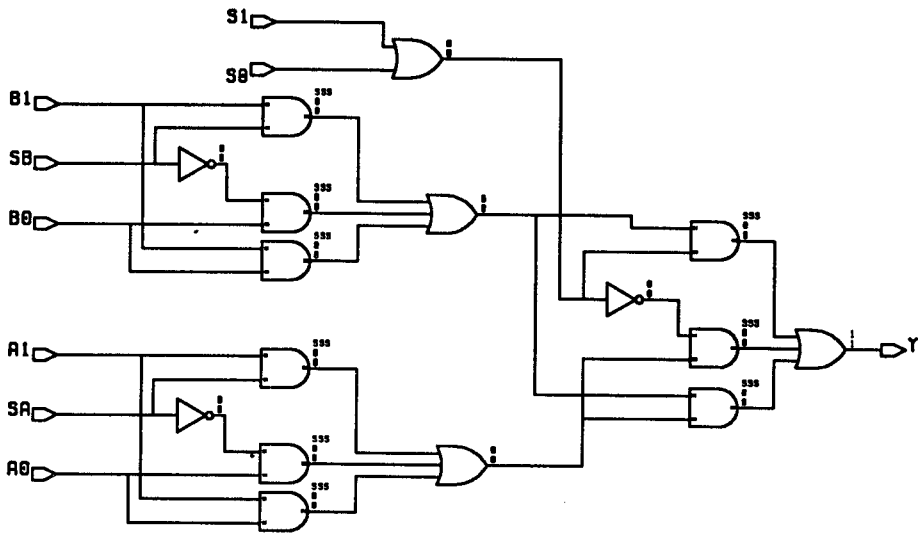


Figure 8 Simulationsmodell fuer 'ACTMOD'

Fuer die Schaltungssimulation unter 'QUICKSIM' findet das obige Modell, bestehend aus Elementen der 'Generic_Lib' Verwendung. Hier sind deutlich die drei Multiplexer-Strukturen sowie das Oder-Gatter aus dem vorhergehenden Schaubild zu erkennen.
Aus der gewaehlten Schaltungsstruktur ergeben sich nahezu identische Verzoegerungszeiten fuer alle Hardmacros aus der Bausteinbibliothek unabhaengig von der Komplexitaet ihrer logischen Funktion.

3.4.2 ACTEL-Softmacros

Die in der ACTEL-Library enthaltenen Softmacros wiederum sind aus 2 bis ueber 100 Hardmacros aufgebaut und benoetigen dementsprechend viele Module auf dem Baustein. Bei Zeitbetrachtungen ist zu beruecksichtigen, dass sie mehrere Logikebenen enthalten koennen.
Sie stellen komplexere Funktionen wie Addierer, Komparatoren, RAM-Strukturen, Zaehler, Peripheriebausteine etc. zur Verfuegung.

4. Realisierung der TRTC-Unit auf einem ACTEL-FPGA

4.1 Arbeitsgrundlage

Bei der Realisierung der Schaltung konnte auf eine Schaltungsbeschreibung in Form von Blockschaltbildern und in LOG/IC zurueckgegriffen werden.
Die Uebernahme der LOG/IC-Beschreibung in die ALS-Entwicklungsumgebung waere allerdings hoechst ineffektiv gewesen, da LOG/IC-Entwuerfe nur auf einfache Gatter aufbauen koennen, die denselben Platzbedarf auf dem Chip haben, wie komplexere ACTEL-Hardmacros. Darum wurde die Gesamtschaltung auf Grundlage der ACTEL-Bibliothek optimiert und voellig neu entworfen.

Die einzelnen Teilschaltungen waren bereits mit LOG/IC simuliert, allerdings interaktiv, was wiederum nur einen geringen Informationsgehalt bedeutete.
Eine Simulation der Gesamtschaltung war mit LOG/IC nicht moeglich, weil von LOG/IC in der damaligen Version nur 256 Variablen verarbeitet werden konnten.

Fuer die Gesamtschaltung ist mit einem Aufwand von ca. 1000 Registern und 1000 einfacheren Gattern, wie NAND, MULTIPLEXER, BUFFER etc. zu rechnen. Daraus ergibt sich ein Bedarf an Logikmodulen von ca. 3000 Stueck, was eine Aufteilung der Schaltung auf mehrere Bausteine erforderlich macht.

4.2 Beispiel des hierarchischen Schaltungsaufbaus

Im Folgenden soll exemplarisch ein Streifzug durch die einzelnen Hierarchieebenen innerhalb eines Blockes unternommen werden, angefangen von den oberen Ebenen, wo sich noch viele benutzerdefinierte Funktionsbloecke finden bis hinab zu den ACTEL-Hardmacrozellen auf Siliziumbasis.

4.2.1 Gesamtschaltbild TRTC

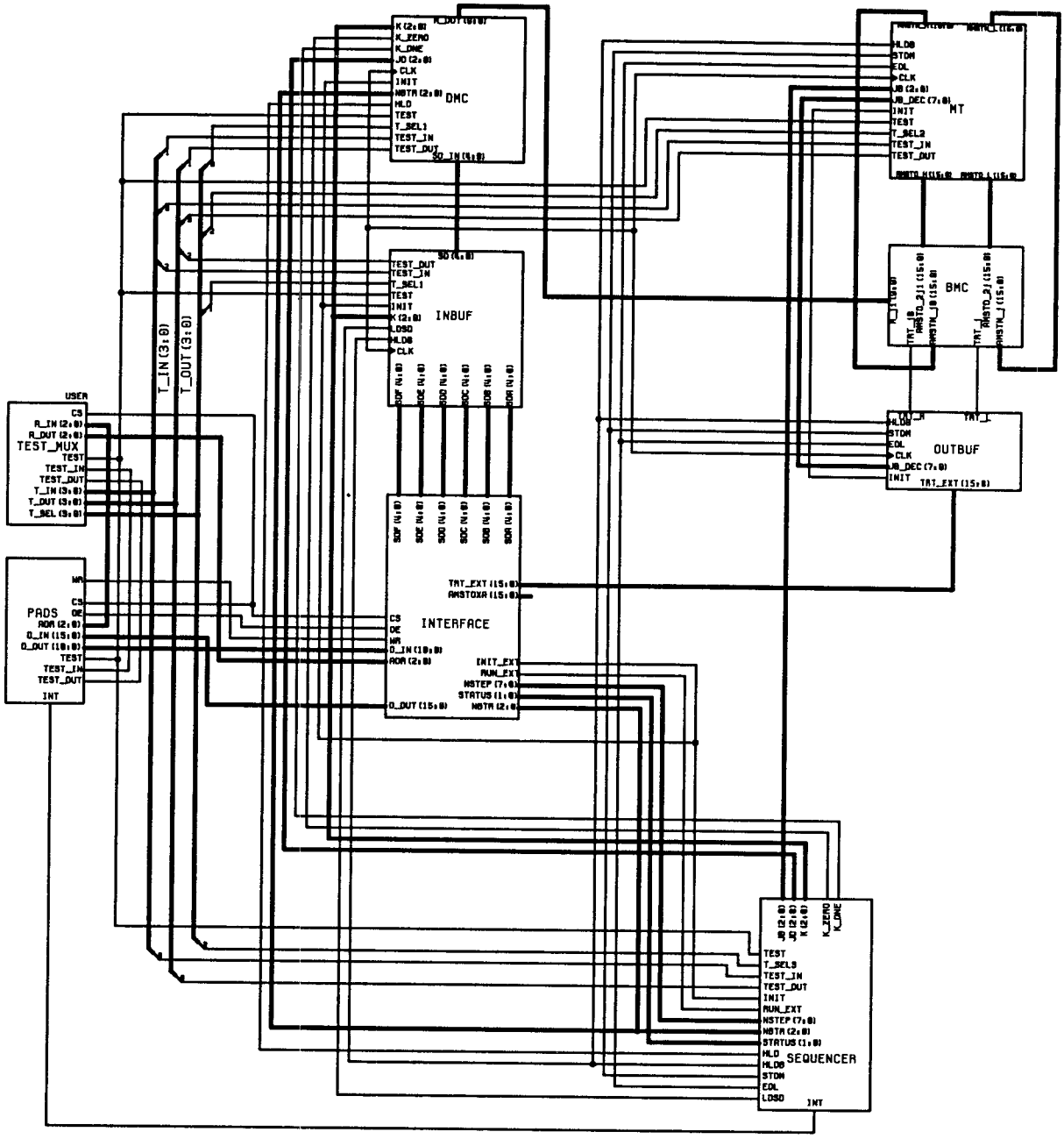


Figure 9 Block 'TRTC' (Gesamtschaltung)

4.2.2 Blockschaltbild des Addierwerkes DMC

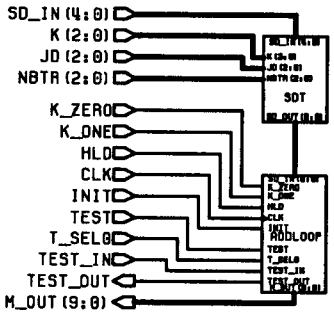


Figure 10 Block 'DMC'

Funktion: Aufaddition von binaercodierten Festkommazahlen nach vorheriger Gewichtung mit einem Vorzeichenfaktor, der in einem 11x16 ROM (SDT) abgelegt ist.
Aufbau: User-defined blocks

4.2.3 Blockschaltbild des Addierers 'ADDLOOP'

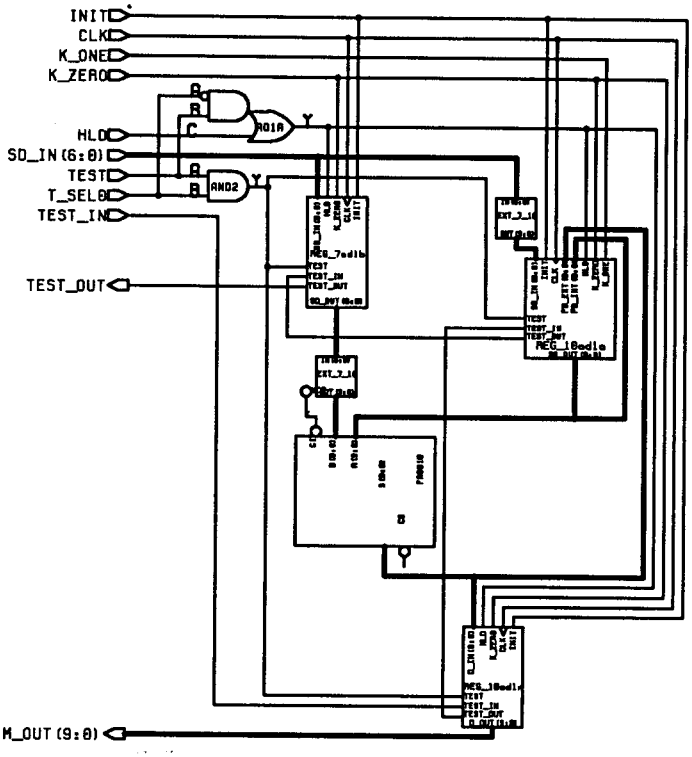


Figure 11 Block 'ADDLOOP'

Funktion: Aufaddition von K 7 Bit breiten Binaerwerten
Aufbau: User-defined Blocks, Softmacro FADD10, Hardmacros AND2 etc.

4.2.4 Blockschaltbild des Registerblockes mit Steuerlogik (REG_10adlm)

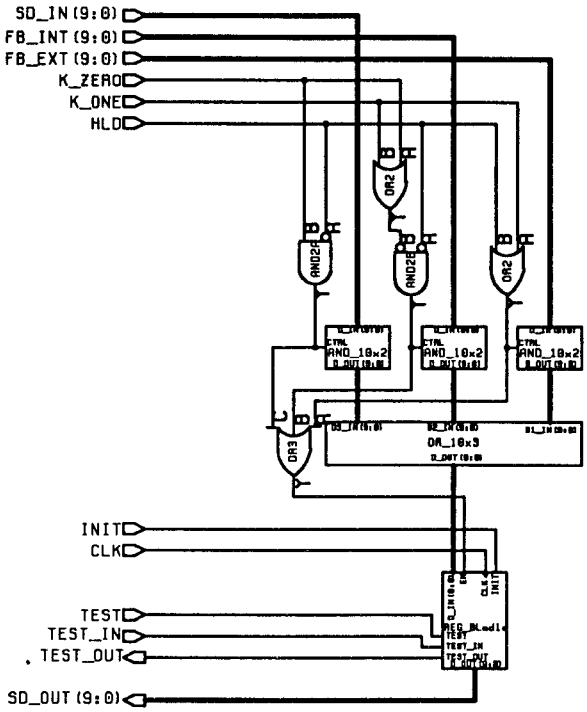


Figure 12 Block 'REG_10adla'

Funktion: Registerblock u. Steuerlogik als Buffer fuer vom Addierer gebildete Summen
Aufbau: User-defined blocks sowie Hardmacros 'AND2', 'OR2', 'OR3'

4.2.5 Blockschaltbild des Registerblockes

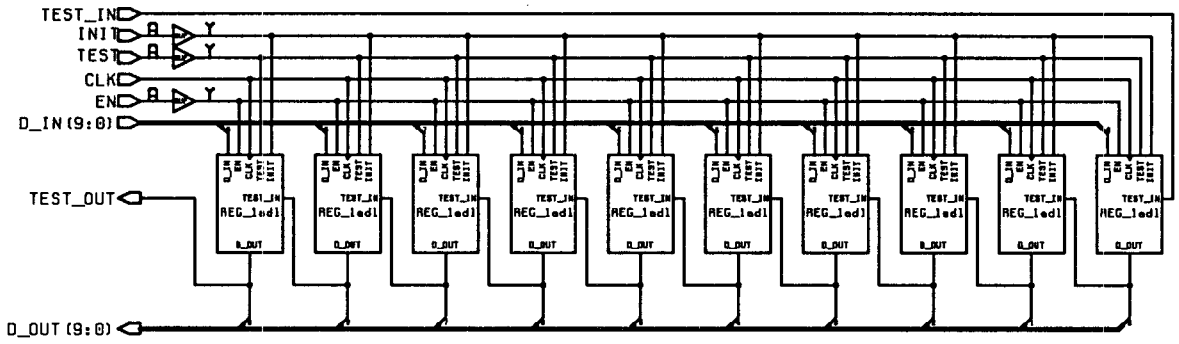


Figure 13 Block 'REG_bladla'

Funktion: 10fach-Registerblock als Eingangs-Zwischenspeicher fuer den Addierer
Aufbau: User-defined blocks, ACTEL-Hardmacros (buffer)
Testmoeglichkeit ueber Testpfad mit den Signalen 'TEST', 'TEST_IN', 'TEST_OUT'

4.2.6 Blockschaltbild eines einzelnen Registers

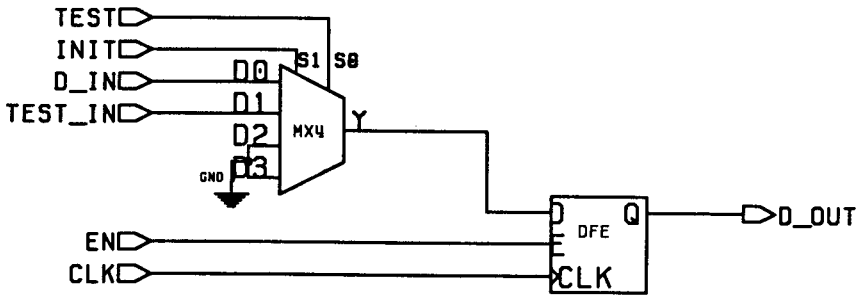


Figure 14 Block 'REG_1adla'

Funktion: Register mit vorgeschaltetem Multiplexer zur Konfiguration des Daten-Eingangs des Flip-Flops als Dateneingang, Testeingang oder Initialisierungseingang.
Aufbau: ACTEL-Hardmacros 'MX4' und 'DF1'

5. Das Testkonzept

Da sich die Gesamtfunktion als aeusserst komplex darstellt, ist es wichtig, schon beim Schaltungsentwurf geeignete Testmoeglichkeiten vorzusehen, um den Baustein nicht als 'Black Box' zu konzipieren, deren internes Verhalten im programmierten Zustand nicht mehr ueberprueft werden kann.

5.1 ACTEL-Testmoeglichkeiten

Ueber eine Actionprobe-Unit, die zwischen Bausteinfassung und Baustein eingesteckt wird, laesst sich der Baustein durch Aktivieren einer Steuerleitung in den Testmodus umschalten. Danach koennen ueber als Adresseingaenge geschaltete Pins beliebige Schaltungspunkte adressiert und die an ihnen anliegenden Signalzustaende an einen Ausgangspin des Bausteins gefuehrt werden. Da zwei dieser Actionprobe-Pins vorhanden sind, koennen auch beispielsweise Verzoegerungszeiten zwischen zwei internen Schaltungspunkten gemessen werden.

Von Nachteil ist allerdings der Umstand, dass nur Signalinformationen gelesen, nicht aber etwa Flip-Flops mit definierten Startwerten geladen werden koennen. Dies wird vor allem zum Laden von Zaehlern mit Anfangswerten notwendig, um etwa das logische Verhalten nach einem relevanten Zaehlerstandswechsel zu untersuchen und den Test somit abzukuerzen.

5.2 Verbesserte Testmoeglichkeiten durch Einfuehrung eines SCAN-Paths

Da, wie oben geschildert, die gegebenen Testmoeglichkeiten nicht als ausreichend erscheinen, ist hier der Anwender gefordert, selbst die notwendigen schaltungstechnischen Massnahmen fuer die Moeglichkeit eines (vollstaendigen) Tests zu ergreifen.

Fuer den Testbetrieb wurden am Baustein drei zusaetzliche Pins vorgesehen:

TEST	Aktivieren des Testmodus
TEST_IN	Serielles Einschreiben von Testvektoren mit dem Systemtakt
TEST_OUT	Serielles Auslesen von Testvektoren mit dem Systemtakt.

Hier soll kurz die entsprechende Eingangsbeschaltung von SCAN-Path- Registern aufgezeigt werden:

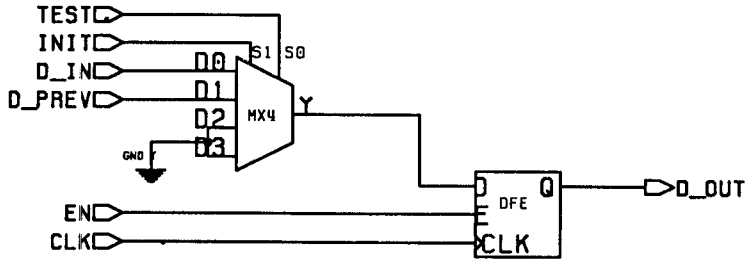


Figure 15 Schaltbild 'REG_1mto'

3 moegliche Betriebsarten:

	INIT	TEST
Normalbetrieb	0	0
Testbetrieb	0	1
synchrones Ruecksetzen	1	-

Im Normalbetrieb sind beide Steuerleitungen des Eingangsmultiplexers deaktiviert und das anliegende Datum 'D_IN' wird mit der aktiven Taktflanke von 'CLK' in das D-Flip-Flop geladen. Um den Baustein zu initialisieren, wird der Dateneingang durch Aktivieren von 'INIT' mit der naechsten aktiven Flanke auf Masse gelegt.
Im Testmodus jedoch wird das Datum des vorhergehenden Registers ueber den Eingang 'D_PREV' geladen und der eigene Registerinhalt ueber 'D_OUT' und den Multiplexer-Eingang 'D_IN' des Nachfolge-Flip-Flops demselben zur Verfuegung gestellt.

Im vorliegenden Fall wurde der SCAN-Path wegen der grossen Zahl von etwa 1000 Registern in mehrere Einzelpfade aufgeteilt, denn der Aufwand fuer die Generierung der Testvektoren und Ueberpruefung der Ergebnisvektoren laesst sich stark reduzieren, wenn zuerst Teilbloecke der Schaltung getestet werden, bevor die Gesamtschaltung dem Test unterworfen wird.
Deshalb kann im Testmodus ('TEST' = H) durch Anlegen einer Blocknummer an den Adressbus jeweils ein Registerblock ausgewaehlt werden, dessen Testsignale dann auf Pins geschaltet werden koennen. Auf diese Weise kann sukzessive Block fuer Block ueberprueft werden.

6. Ergebnisse der Diplomarbeit

Urspruengliche Ziele

Es war zuerst vorgesehen, den gesamten VITERBI-Decoder auf einem ACTEL-FPGA zu realisieren. Um bei Schwierigkeiten bei der Durchfuehrung der Diplomarbeit dennoch einen definierten Abschluss zu erreichen, wurden Meilensteine formuliert, die jeweils abgeschlossene Arbeitspakete als Teilziele enthielten.

Aufgetretene Probleme

Der Gedanke an die Integration der Gesamtschaltung auf einem Baustein alleine musste bald aufgegeben werden, da die Groesse der Schaltung selbst den groessten noch nicht einmal verfuegbaren ACTEL-Baustein um ein mehrfaches ueberfordert. Ausserdem war keine Vorausentwicklung von Schaltungsteilen auf dem A1280 moeglich, da die Entwicklungssoftware fuer diese Bausteinfamilie noch nicht freigegeben worden ist.
Die Arbeit unter dem Action Logic System von ACTEL wurde durch die mangelhafte Unterstuetzung der Mentor-Platform nicht gerade erleichtert. Hierbei waeren vor allem eine grafische Oberflaeche, Moeglichkeit zum Wechsel zwischen Applikationen per Tastenkombination sowie eine Mausunterstuetzung sehr hilfreichgewesen.
Ein schwerwiegender Fehler in der ACTEL-Macrobibliothek konnte nur nach unzaehligen Telefonaten und FAXen nach einigen Wochen endlich behoben werden.

Konsequenz

Wegen der oben genannten Probleme, wie der Komplexitaet der Schaltung und aus Ermangelung geeigneterr Bausteine beschraenkte ich mich in meiner Diplomarbeit darauf, einen Teil der Gesamtschaltung, das Steuerwerk, auf einem ACTEL-FPGA zu implementieren. Die Loesung dieser Aufgabe hat mir und dem ganzen Projektteam wichtige Einblicke in den guenstigsten Ablauf des Entwurfsprozesses unter Zuhilfenahme der ACTEL-Entwicklungssoftware verschafft. Die Realisierung der anderen Baugruppen wird sich nach diesem Schema durchfuehren lassen.

Bericht auf dem Workshop der MPC-Gruppe BW-FHn in Göppingen am 10.6.91

2. ERFAHRUNGEN MIT DEM ENTWURFSSYSTEM ES2 SOLO1400/PC

Kurt H. Schmidt, Markus Kreuz
Labor für IC-Entwurf der Fachhochschule Furtwangen

1. Einführung

Das Entwurfssystem SOLO1400 für den Entwurf anwendungsspezifischer integrierter Schaltungen stammt von European Silicon Structures (ES2) und kann auf einem Personal-Computer installiert werden. Eine Zusatzkarte gehört bei der Anschaffung mit dazu, die die PC-Hardware erweitert. Ein solches System steht im Labor für IC-Entwurf der Fachhochschule Furtwangen seit 1990 zur Verfügung und wurde im WS90/91 zum Entwurf eines ASICs mit Standardzellen der ES2 1,5µm CMOS Technologie verwendet.

2. Ziele

Der Entwurf war Bestandteil einer Diplomarbeit im Wintersemester 1990/91 mit folgenden Zielen:

- Erarbeitung der Nutzung SOLO1400,
- Erstellen einer eigenen Anleitung,
- Durchführung eines ASIC-Entwurfs im Umfang von etwa 2000 Gatteräquivalenten und frühzeitige Einreichung über EUROCHIP,
- damit auch unter Berücksichtigung des in Aussicht gestellten schnellen Turnarounds ein erstes Testen des ausgelieferten Chips noch während der Laufzeit der Diplomarbeit möglich wurde.

Die genannten Ziele, das sei vorab schon gesagt, konnten alle erreicht werden. Die nächsten Ziele sind somit möglich und bereits in Angriff genommen worden:

- Durchführung eines weiteren ASIC-Entwurfs mit einer Diplomarbeit im jetzt laufenden Sommersemester 1991
- Planung eines dritten ASICs für das kommende Wintersemester.

3. Gegebenheiten in SOL01400

Die Bedienung wird im wesentlichen durch den allgemein und hinlänglich bekannten Designablauf bestimmt. Zunächst müssen eine Reihe von Parametern mittels des Designmanagers behandelt werden. Die folgenden Abschnitte sind aus der angefertigten Kurzanleitung auszugsweise entnommen.

Parameterhandling mit dem Designmanager

Die Parameter, mit welchen ein Programm aufgerufen wird, können durch Eingabe des Programmnamen, gefolgt von einem Fragezeichen, angezeigt werden. Um einen Parameter zu ändern, wird "set", gefolgt vom Namen des entsprechenden Objekts und die Bezeichnung des Parameters mit dem gewünschten neuen Wert des Parameters eingegeben. Zum Beispiel kann der Designname mit dem Befehl

set design name NeuerName <RETURN>

abgeändert werden. "NeuerName" steht für die Bezeichnung des Designs. Über diese Namen können Teildesigns mit unterschiedlichen Namen in einem Directory gespeichert werden. Die Teildesigns können dann in weiteren Designs in diesem Directory verwendet werden.

Von dieser Oberfläche werden die einzelnen Programme aufgerufen, die zur Schaltplaneingabe, Simulation und Layoutcompilierung benötigt werden.

Erzeugen von Netzlisten

Um einen Chip zu entwerfen, wird von dem Programm eine Netzliste verwendet, die auf verschiedene Arten eingegeben werden kann:

- Die Eingabe eines Schaltplans mit dem Programm "DRAFT". Dies ist die einfachste Art ein Design zu erzeugen. Die Schaltungsteile werden als Zeichnung eingegeben, indem Schaltsymbole, die zuvor definiert wurden oder aus einer ES2 Bibliothek stammen, ausgewählt, plaziert und miteinander verbunden werden. DRAFT erzeugt dann einen File, der die Schaltung in der Syntax der "descrition language" "Model" beschreibt und für die Kompilierung und weitere Verwendung geeignet ist. Draft kann ebenso zur Erzeugung eines Plots der Schaltung genutzt werden.

- Ein Modelfile kann auch durch Umsetzen einer Funktionsbeschreibung mit den Programmen "SYNTHESISE" und "GENERATE" erzeugt werden.

- Die Eingabe des Modelfiles im ASCII Format mit einem Texteditor ermöglicht, die Schaltplanebene zu umgehen. Die Verwendung von MODEL erlaubt, Bauteile zu parametrisieren, was eine schnelle Erzeugung verschiedener Versionen desselben Grundelements gestattet. MODEL enthält Elemente von Hochsprachen, wie Schleifen und Bedingungen, welche es ermöglichen, eine Schaltung schnell und effektiv zu beschreiben.

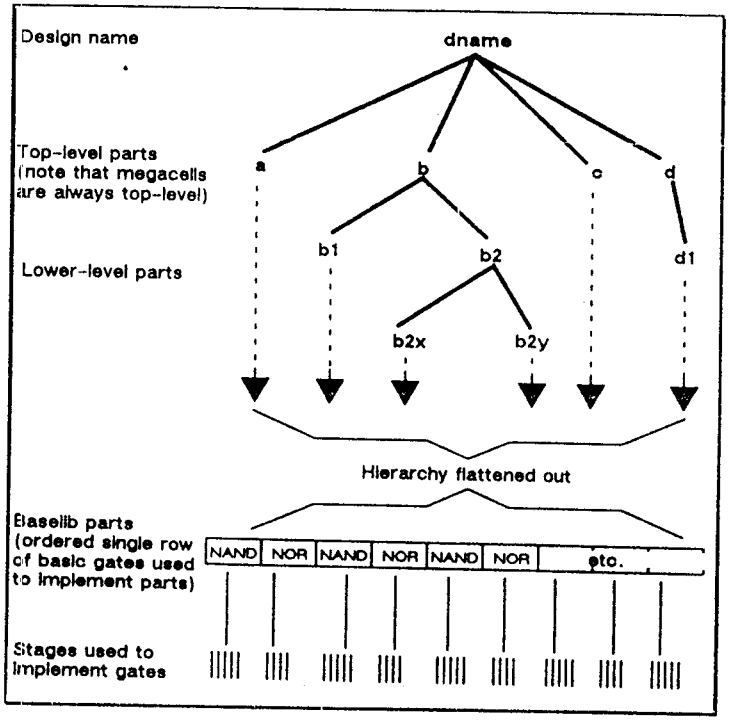
Um diesen im ASCII Format, mit DRAFT oder auf andere Weise, erstellten Beschreibungsfile (Name: design.MOD) für andere Programmteile des Designsystems lesbar zu machen, muß der File mit dem Programm MODEL compiliert werden, wenn das Design vollständig ist. MODEL führt eine Reihe von Tests während dem Compilelauf durch, was fatale Designfehler weitgehend ausschließt. Das Ergebnis dieser Übersetzung ist ein File in "Intermediate Design Language" (IDL), welcher in der Simulation des Designs verwendet wird.

Mit dem Programm "EXTRACT" wird aus der von MODEL erzeugten IDL-Netzliste die Information zur Positionierung der Anschlußpads und die Steuerfiles für die Simulation generiert. Da die Positionsinformation Grundlage für die Programme "PINOUT" und "TIS" sind, muß das Programm "EXTRACT" auf jeden Fall gestartet werden.

Das Programm "EXPAND" erzeugt aus dem IDL-File eine lesbare Netzliste, in welcher alle im Design enthaltenen Bauteile mit den daran angeschlossenen Signalen aufgeführt sind. Die Verwendung dieses Programms wird empfohlen, ist aber für den Programmablauf nicht unbedingt notwendig.

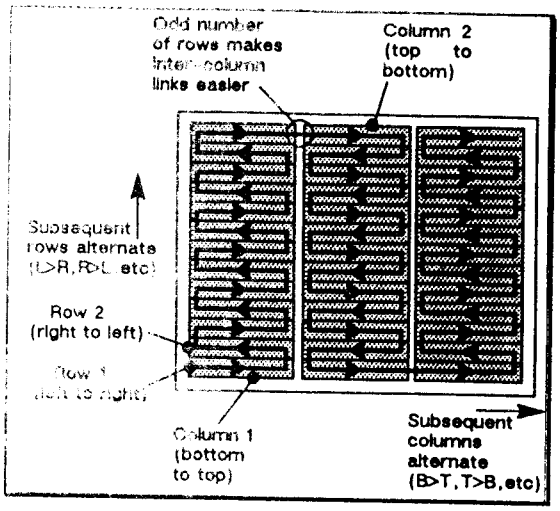
Simulation und Layout

Das so vorbereitete Design wird als Nächstes simuliert. Entsprechen die Ergebnisse der Simulation nicht den Erwartungen, kann der fehlerhafte Schaltungsteil verbessert werden. Funktioniert das Design korrekt, kann mit der Plazierung der Transistoren und den Pads begonnen werden. Das Programm "PLACE" berechnet anhand des IDL-Files die Anzahl der benötigten p-n-Transistorpaare (Stages). Diese Transistorpaare ordnet das Programm in Zeilen und Spalten an. Während dem Plazierungsvorgang werden die hierarchisch angeordneten Schaltungsteile in eine Reihe von Transistorpaaren umgesetzt, die dann über das Layout gelegt wird.



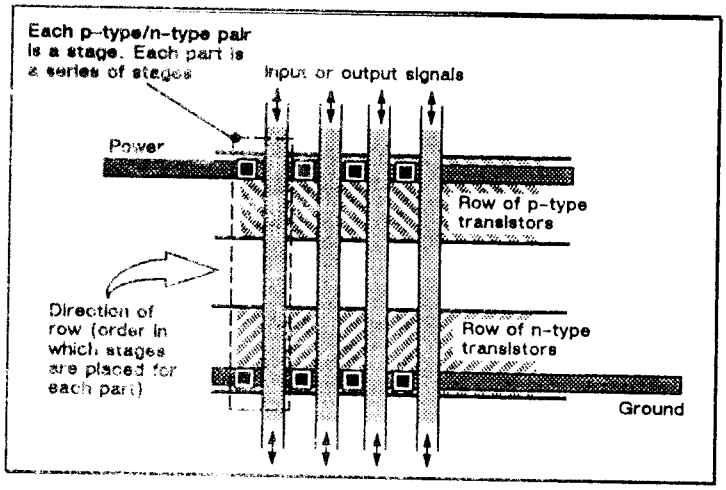
Darstellung der Umwandlung eines hierarchischen Designs in eine Kette von Transistorpaaren.

Die vom Programm PLACE erzeugte Stages Kette muß in eine etwa quadratische Chipfläche gelegt werden. Dies wird erreicht, indem die Kette wiederholt so gefaltet wird, das die einfache Reihe in mehrere Zeilen und Spalten aufteilt, wie in der Abbildung gezeigt.



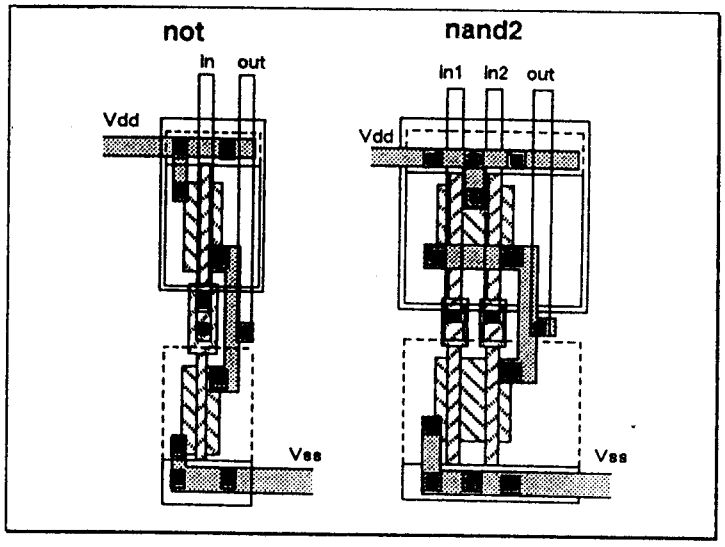
PLACE zeigt dann die Anzahl der verwendeten Stages und die Zeilen- und Spaltenzahl des vorgeschlagenen Layouts an. Die Werte können akzeptiert oder überschrieben werden,wenn dieses Layout nicht den eigenen Wünschen entspricht. Wenn das Design verändert wird, zeigt PLACE die Anzahl der daraus resultierenden ungenutzten Transistorpaare an. Auch die eingestellten Werte können so lange überschrieben werden, bis das Ergebnis befriedigt.

Anschließend werden die Reihen mit Transistoren besetzt. Die genaue Beschreibung der Transistorlayouts werden von dem Programm "GATE" in einen Layout-File geschrieben. Das Programm verwendet die Ergebnisse des Programms "PLACE" als Eingabe. Während des Programmlaufs wird kein Bild angezeigt. Die plazierten Transistorpaare sind nebeneinander angeordnet, wie im folgenden Bild gezeigt wird:



Typische Anordnung der Transistoren in einer Reihe.

Die Transistoren werden innerhalb der Grundelemente (Bibliothek-Zellen) entsprechend der Funktion der Zelle verdrahtet, mit Ein- und Ausgängen am oberen und unteren Zeilenrand. Das folgende Bild zeigt, wie zwei Standardbauteile verdrahtet sind.



Implementierung von zwei typischen Schaltungselementen.

Definition des Pinouts

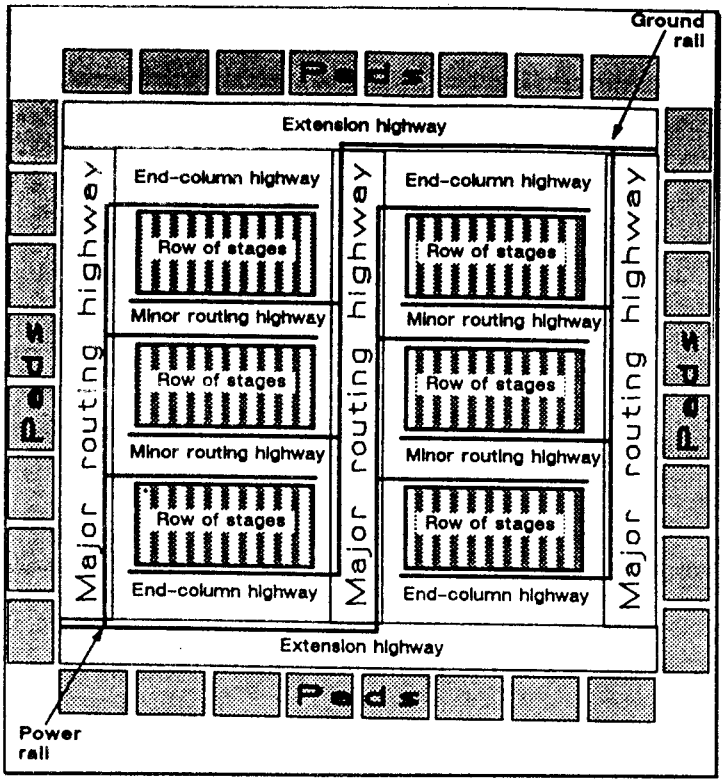
Um die Verdrahtung des Chips zu ermöglichen, müssen die Padzellen mit dem Programm "PINOUT" plaziert werden. Im SOLO1400/PC System wird in diesem Programm eine Liste erstellt, welche die Reihenfolge der Pads am Rand des Chips festlegt und die Zuordnung der Pads zu Signalleitungen im Chip und den Anschlußpins des Gehäuses beschreibt. Wenn die Anordnung der Signale an den Gehäusepins nicht dem gewünschten Anschlußbild entspricht, kann dieser File mit einem ASCII-Editor so verändert werden, daß die Signalnamen den richtigen Padpositionen und den richtigen Gehäusepins zugeordnet ist. Die Zuordnung von Padposition zu Anschlußpin sollte man möglichst nicht verändern, da die Bonddrähte bei der vorgegebenen Zuordnung ohne Überkreuzung und möglichst kurz ausgeführt werden. Werden in einem Design unterschiedlich breite Padzellen verwendet, kann eine Verschiebung der Padzellen um die Ecken manchmal notwendig werden. Sollte dies verlangt werden, kann es einfach dadurch gemacht werden, daß die Kantenbezeichnung (right, top,bottom,left) durch die entsprechend angrenzende Kante ersetzt wird.

Automatisches Plazieren und Verdrahten

Um einen schnellen Erfolg zu haben, kann die benötigte Breite aller Padzellen einer Seite mit den Daten aus der Bibliothek berechnet werden und so die annähernde quadratische Form des Chips als Hilfe dienen, die Pads auf der längsten Seite noch etwas zu verschieben. Die Position der Leads (Kontaktflächen des Gehäuses in der Cavity = Platz zum Einbau des Chips) ist bereits vom Gehäusehersteller festgelegt und beginnt im Allgemeinen in der Mitte der oberen Reihe mit dem Pin 1. Die Pins laufen dann in steigender Reihenfolge entgegen dem Uhrzeigersinn. Auf jeder Seite der Cavity sind etwa gleich viele Leads angeordnet.

Wenn die Anzahl der Stromversorgungspins nicht ausreicht, zeigt der Ergebnisfile des Programms "PADAUDIT" an, welche Padzelle zu weit von der Padzelle für Vdd und Vss entfernt ist. In der Nähe dieser Zellen sollte eine weitere Betriebsspannungszelle eingebaut werden, um die Versorgungsspannungsleitungen des Padrings zu entlasten. Das Programm "PADAUDIT" liefert am Ende des Ablaufs eine Meldung, welche zeigt, ob die Stromversorgung des Padrings ausreichend dimensioniert wurde. Wenn die Dimensionierung nicht ausreicht, kann der Problembereich in dem File design.PLF gesehen werden, indem der für ein bestimmtes Pad noch verbleibende Strom auf der Versorgungsleitung überprüft wird.

Die Verbindungen zwischen den einzelnen Standardzellen werden mit dem Programm "ROUTE" automatisch hergestellt. Es verwendet die Ausgabe von GATE und verbindet die Zellen untereinander und mit den Padzellen. ROUTE minimiert die Verbindungslängen der einzelnen Leitungen. Dadurch wird die Verteilung der Zellen in der in den inneren Teil (Core) des Chips gefalteten Transistorkette festgelegt. Durch die Verwendung von "routing highways" zwischen den Spalten und Zeilen kann die Verdrahtungslänge stark reduziert werden.



Routing highways in einem Design Layout

Die normale Funktion des Programms kann auf verschiedene Arten beeinflusst werden. Zeitkritische Signale kann man zum Beispiel im MODEL-Code als kritisch markieren und damit werden diese als Erste geroutet. Wenn notwendig, können einzelne Signale auch von Hand gelegt werden, indem man die "highways" definiert, denen die Leitung folgen soll, wenn sie geroutet wird. Am Ende des Routing-Vorgangs wird die Anzahl der Signalwege für jeden "highway" angezeigt.

Erzeugen eines CIF Files

Da in den bisher verwendeten Programmen noch kein Symbol tatsächlich auf eine feste Koordinate gelegt wurde und die Darstellung noch an Verdrahtungskanäle und Zellenlängen angepasst wurde, müssen jetzt die Flächen in Form eines CIF-Files gezeichnet werden. Dieser Zeichenvorgang entspricht ungefähr dem, wenn ein Zeichner eine Liste mit Plazierungsbeschreibungen in eine Zeichnung umsetzt. Das Programm "DRAW" verwendet hierzu die Ausgabe von ROUTE und übersetzt diese in Caltech Intermediate Format (CIF) als genormte Schnittstelle. Zusätzlich wird ein File erzeugt, welcher die Netzkapazitäten für alle Knoten des Designs enthält und somit eine Simulation mit EXERT im belasteten Zustand erlaubt. DRAW berechnet auch die Dimensionen des Chips und zeigt diese an.

Alle ES2 Standardzellen sind in einer Doppel-Metall-N-Well-CMOS Technologie implementiert, welche mit 10 Masken arbeitet:

Schritt Nr.	Beschreibung	entsprechende CIF-Layer Nr.
1	N-Wanne (N-Well)	1
2	Aktive Gebiete Diffusion (Activ Diffusion)	2
3	Poly (Polysilicon)	11
4	N+ Diffusion	12
5	P+ Diffusion	14
6	Kontaktlöcher (Contacts)	16
7	Metall 1	17
8	Via	18
9	Metall 2	19
10	Passivierung (Passivation)	20

Um das Layout bildlich zu zeigen und für Dokumentationen, kann der CIF-File mit dem Programm "ARTVIEW" in einen Plotter-File umgewandelt werden. Die grafische Anzeige auf dem Bildschirm ermöglicht, daß der Ausschnitt, welcher geplottet werden soll, vorher genau festgelegt werden kann. Der Benutzer kann dabei durch Zoom und ähnliche Werkzeuge den gewünschten Ausschnitt auf dem Bildschirm ansehen, bevor er die zeitraubende Ausgabe des Plotters wählt. Mit dem Programm können auch einzelne Layers aus- oder eingeblendet werden. Die hierarchisch aufgeteilte Darstellung wird auch hier unterstützt, womit sich Einzelheiten zusammenfassen lassen, was die Übersichtlichkeit der Zeichnungen erhöht. Der von ARTVIEW erzeugte HPGL-File kann mit einer einfachen Steuersequenz vor dem File an den Plotter geschickt werden.

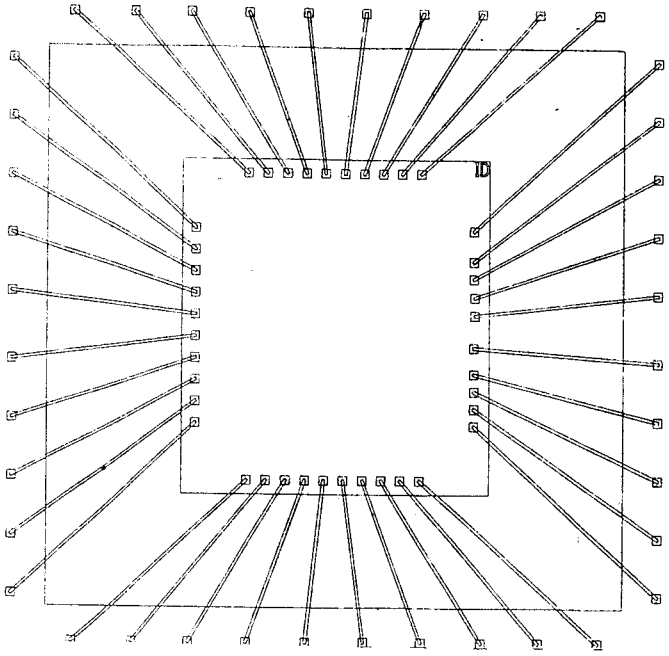
Im jetzigen Zustand des Designs empfiehlt sich, die Treiberleistung und Belastung einzelner Zellen nochmals zu überprüfen, da jetzt die Leitungskapazitäten bekannt sind. Dazu wird noch einmal das Programm AUDIT aufgerufen. Hier ergeben sich oft bei knapp dimensionierten Treibern noch Probleme, da jetzt die Leitungskapazität als zusätzliche Belastung an den Treiber angeschlossen ist.

Die Fehler sollten durch entsprechende Anpassung der Treiber behoben werden, was ein Redesign ab der Schaltplaneingabe erfordert. Für Designs mit 8000 Transistoren kommt so etwa eine Stunde zusätzlicher Arbeit auf den Designer zu, wenn er die Simulationen so weit wie möglich übergeht. Die stärkere Treiberleistung wird meist durch zusätzliche doppelte Inverter oder durch Ersetzen von Invertern durch solche mit größerer Treiberleistung erreicht, was nur bei zeitkritischen Signalen eine Simulation des Blockes erfordert. Da jetzt aber die Simulationsdaten schon vorliegen, reduziert sich die Arbeit auf ein Minimum.

Einbau des Chips in ein Gehäuse

Das Programm PACKAGE stellt ein Steuerfile für die Anschlußverdrahtung (Bonding) des Chips im Gehäuse her. In diesem File wird für jede Zuordnung von Chippad zu Gehäusepin die Koordinatenwerte des Bonddrahtende eingetragen. Mit der PC Version von SOLO1400 ist es leider nicht möglich, die im Handbuch beschriebene grafische Oberfläche zu nutzen, da diese nur auf der SUN und der VAX lauffähig ist. In der PC Version kann mit PACKAGE nur das Gehäuse gewählt werden und der Chip in 90 Grad Schritte gedreht werden, um die richtige Verdrahtung des Chips im Gehäuse zu ermöglichen. Die Zuordnung von Chippad zu Gehäusepin wird nach der Liste von PINOUT gemacht.

Mit dem Programm PADAUDIT wird nach dem das Programm PACKAGE gelaufen ist, noch einmal überprüft, ob die Chippads ordnungsgemäß verdrahtet sind. Um die Anordnung der geplanten Bonddrähte sichtbar zu machen, erzeugt PADAUDIT, wenn es nach PACKAGE gestartet wird, einen File mit der grafischen Darstellung der Cavity im CIF Format, welcher mit ARTVIEW auf dem Bildschirm angezeigt und auf einem Plotter ausgegeben werden kann. Damit können viele Fehler lokalisiert werden, die von PACKAGE ausgegeben werden. Ein fehlerfreies Design könnte zum Beispiel so aussehen:



Produktionsvorbereitung

Die eigentliche Designarbeit ist mit diesem Schritt abgeschlossen. Um jedoch die richtige Funktion des Designs sicher zu stellen, müssen an dieser Stelle im Designablauf noch einige Simulationen und Kontrollen durchgeführt werden, bevor das Design in die Produktion gehen kann.

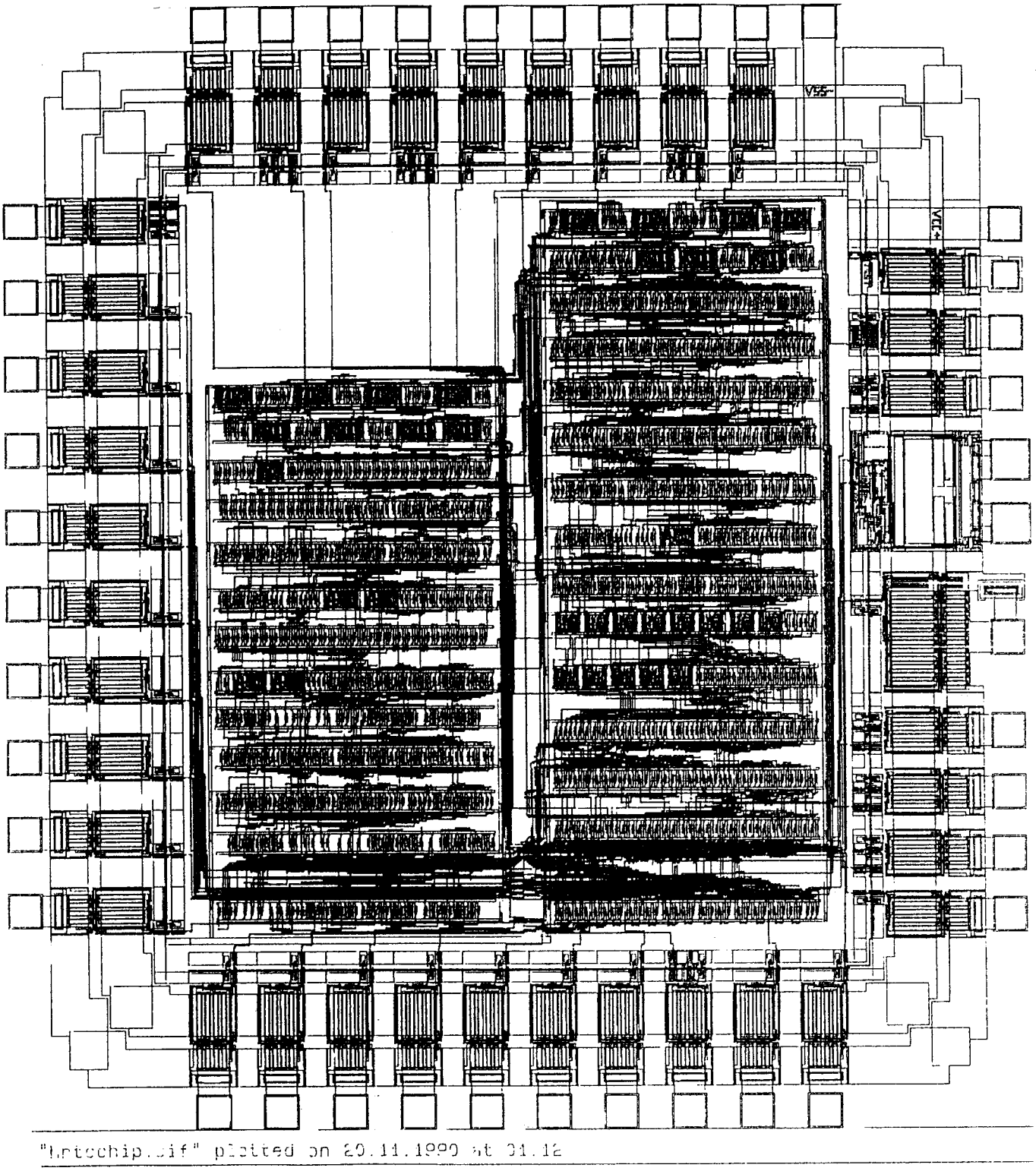
Als Grundlage für die Vergleiche werden noch drei Simulationsläufe mit den gleichen Stimuli Daten benötigt: Eine Simulation mit minimaler, eine mit durchschnittlicher und eine mit maximaler Verzögerungszeit. Die Ergebnisse von der minimal und der maximal verzögerten Simulation dürfen sich nicht unterscheiden. Um dieses zu überprüfen wird nach den Simulationen das Programm DIFFVEC aufgerufen. Die Ergebnisse der Simulation mit durchschnittlicher Verzögerung werden mit drei Programmen bearbeitet: TIS überprüft, ob die Testvektoren den Anforderungen von ES2 genügen. CHECKSKEW kontrolliert, ob der Zeitpunkt, zu welchem die Taktflanke im Verhältnis zu asynchronen Signalen steht, einen Einfluß auf die Ausgänge des Chips hat. Mit TOTLF werden die Simulationsergebnisse in einen von der Fabrik lesbaren File übersetzt.

Erst jetzt kann das Design mit dem Programm SHIPDES für die Produktion vorbereitet werden. SHIPDES überprüft, ob alle vorgeschriebenen Programmteile ordnungsgemäß ausgeführt wurden. Wurden alle oben beschriebenen Programme fehlerfrei ausgeführt, werden hier keine Probleme auftreten.

4. Arbeitsbeispiel

Die geplante Schaltung für das Chipbeispiel ist Teil eines Zeitprozessors. Die Vorgaben wurden in einem Pflichtenkatalog der Funktionen und Betriebsarten des Bausteins ausführlich niedergelegt. Darauf wird hier jedoch nicht näher eingegangen. Der Umfang der Schaltung liegt bei etwa 8000 Transistoren. Die Partitionierung der Schaltung erfolgte gemäß den Teilfunktionen. Sequentielle Schaltkreise wurden mit Scanpathflipflops erstellt, um die Testmethode mit Scanpath später anwenden zu können. Nach der Simulation der entworfenen Teilschaltungen konnte das endgültige Placement und Routing stattfinden. Unter Berücksichtigung der Padvorgaben für die Ein- und Ausgabesignale ergab sich ein Padrahmen, der im Inneren etwas mehr Platz erbrachte, als für die Schaltung benötigt wurde, s. die freie Fläche im Chip-Plot. Bei der Plazierung hätte man durchaus noch eine Umordnung zu einem Quadrat erzwingen können. Wie ersichtlich, entsteht beim automatischen Placement und Routing eine markante zweispaltige Anordnung von Zeilen, die aus Standardzellen gebildet werden. Der Chip hat Fläche von etwa 11,5 mm^2.

Für den EUROCHIP-Lauf von November 1990 erhielt der Entwurf die Bezeichnung hrtcchip. Der Chip wurde im März 1991 ausgeliefert, 10 Stück, gebondet im 40 poligen DIP-Gehäuse mit abnehmbarem Metalldeckplättchen. Das Foto zeigt einen geöffneten Baustein, so daß das Siliziumchip erkennbar ist. Das erste Testen des Bausteins erfolgte im April 1991. Alle Funktionen arbeiteten einwandfrei, somit konnte der Baustein sofort in einen Systemaufbau, für den er gedacht war, eingesetzt werden.



Chip-Plot "hrtcchip"

5. Zusammenfassung

Die mit dem System SOL01400 zum Entwurf integrierter Schaltungen gewonnenen Erfahrungen sind durchweg als positiv zu bewerten. Der vorliegende Bericht zeigt die wesentlichen Entwurfsschritte anhand einer eigenen Kurzanleitung auf und stellt ein mit SOL01400 erstelltes Entwurfsbeispiel im Umfang von 8000 Transistoren als Chip vor. Die Basierung auf PC hat bei noch größerem Schaltungsumfang wohl sehr bald ihre Grenzen. Dank dieses Entwurfssystems konnten an der FH Furtwangen im WS90/91 erzielt werden:

Premieren
1) Rascher Chipentwurf zusammen mit gutem Turnaround ermöglichten in der gleichen Diplomarbeit auch noch das Testen.
2) CMOS Standardzellenchip eigenen Entwurfs auf einer kommerziellen Technologielinie (ES2).

Dieser Erfolg, "das erste Silizium ist korrekt", zwar wie auch schon bei manchen Entwürfen früher, aber - und das spricht für das Tool - volle Simulation und das Layout waren in kurzer Zeit möglich. Der Designer weiß, was für ein "iterativer" Vorgang dahinter steckt, bis ein Entwurf in jeder Hinsicht fehlerfrei ist.

Danksagung: Es sei Herrn Gerhard Angst, Mitarbeiter im Labor für IC-Entwurf, gedankt, denn er hat sehr zum Gelingen des Projekts beigetragen.

Anhang: Angaben zum Entwurfssystem

Solo 1400 Software zum Design von "Compiled Silicon Technology" einschliesslich Layout mit den ES2 Bibliotheken, lauffähig auf einem IBM-AT 286 kompatiblen Rechner mit

- max. 10 MHz Systemtakt
- 640 KB RAM ohne Speichererweiterung
- EGA Graphik
- min 60 MB Harddisk
- Maus
- 8 MHz Clock-Frequenz

1.1 Software

- Schematic Entry "DRAFT"
- Hardware Description Language "MODEL"
- Logic-Simulator "EXERT" (Switch-Level) sowie "TURBO EXERT" (Gate Level)
- Layout Generation "PLACE", "GATE", "ROUTE", "DRAW" mit Parasitic Extraction für Post Layout Simulation
- DESIGN MANAGER
- Libraries:
  Baselib, Logiclib, 74lib, ES2Cell,
  Block Library,
  Analog Library,
  Padlib2, Padlib3
- GENERATOREN
  RAM (bis zu 8 KBit)
  ROM (bis zu 128 KBit)
  PLA (bis zu 128 Termen)
- 1 Satz Dokumentation
- maximal 8.000 Stages plus Macro-Zellen
- Evaluation-Kit für Analog-Zellen inkl. Dokumentation
- Training (eine Person, 5 Tage bei ES2 in München)

1.2 Hardware

- Coprocessorboard mit: NS 32032 Prozessor
  4 MB RAM
  Sonderpreis für Hochschulen

2 Installation

in Ihrem Hause

3 Wartung

während der Garantiezeit	vierteljährlich
nach der Garantiezeit	vierteljährlich

Garantie

- 6 Monate
- 12 Monate bei Abschluss eines Wartungsvertrages ab Installationsdatum

3. Ein VLSI-Chip zur schnellen Berechnung von Faltungsintegralen

W. Rülling

Labor für IC-Entwurf, FH Furtwangen

Zusammenfassung

Für viele meß- und regelungstechnischen Probleme ist die Berechnung von Faltungsintegralen von großer Bedeutung. Insbesondere benötigt man beim Einsatz von miniaturisierten intelligenten Sensoren eine schnelle Berechnungsmöglichkeit, die mit einem möglichst geringem Hardwareaufwand auskommt. Für diese Anwendung wird in der vorliegenden Arbeit ein neues Konzept eines 1-Chip Spezialprozessors vorgestellt. Im Gegensatz zu anderen bekannten Problemlösungen, erlaubt das neue Konzept auch die schnelle Berechnung von Faltungen großer Dimension. Da sowohl die Dimension als auch die Faltungskoeffizienten entsprechend der jeweiligen Anwendung wählbar sind, ist der Chip sehr vielseitig einsetzbar.

Neben der Arbeitsweise des Prozessors wird auch eine spezielle Layoutgenerierungsmethode vorgestellt. Mit ihr wird ein Layout mit extrem kurzen Datenleitungen erreicht, so daß der Prozessor auch mit hohen Taktraten betrieben werden kann.

1 Einleitung

In der vorliegenden Arbeit soll ein Entwurf für einen neuen VLSI-Chip zur Berechnung von Faltungsintegralen vorgestellt werden. Die Faltungsintegration spielt bei vielen regelungstechnischen Problemen eine große Rolle. Beispielsweise benutzt man sie bem Einsatz von Sensoren, um aus einer zeitlichen Folge gelieferter Meßwerte $y(t)$ $(t = 0, 1, 2, 3, \ldots)$ unter Berücksichtigung des Übertragungsverhaltens des Sensors die tatsächlich beobachteten Eingangsgrößen $x(t)$ zu ermitteln. Auf diese Weise beseitigt man die grundsätzlich durch den Meßvorgang bedingten Abweichungen und Meßwertverzerrungen.

In der Praxis kann die Berechnung von Faltungsintegralen beispielsweise mit Hilfe geeigneter Software auf einem Mikrocomputer oder einem schnelleren Signalprozessor durchgeführt werden. Außerdem lassen sich die Korrekturen natürlich auch durch speziell für den jeweiligen Sensor zu entwickelnde Analogschaltungen realisieren.

In der vorliegenden Arbeit soll ein universell für beliebige Sensortypen einsetzbarer Prozessor entwickelt werden, der Faltungsintegrale zu beliebigen vorgegebenen Funktionen y und g berechnen kann. Gegenüber den bisher verwendeten Lösungsansätzen

soll dieser Spezialprozessor den Einsatz von Sensoren mit einem Minimum an zusätzlicher Hardware ermöglichen und damit auch den gleichzeitigen Einsatz vieler Sensoren preisgünstig erlauben. Dazu soll der Prozessor auf einem einzigen Chip realisiert werden, der als universell einsetzbarer busfähiger Peripheriebaustein beispielsweise auf existierenden Controller-Karten verwendet werden kann.

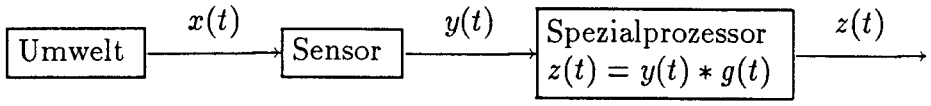


Abbildung 1: Prinzipielle Vorgehensweise bei der Meßwertkorrektur

2 Mathematische Grundlagen

Das Verhalten eines Sensors kann näherungsweise durch ein lineares Netzwerk beschrieben werden. Solche Netzwerke werden üblicherweise durch eine *Gewichtsfunktion* $h(t)$ charakterisiert, mit deren Hilfe die Antwortfunktion $y(t)$ auf jede beliebige Erregerfunktion $x(t)$ durch die folgende Faltung ermittelt werden kann.

$$y(t) = x(t) \cdot y(0) + \int_0^t x(\tau) \cdot h(t-\tau) \mathrm{d}\tau$$

Zur Vereinfachung der Darstellung wollen wir in der Arbeit $y(0) = 0$ voraussetzen. Für die interessierende Anwendung benötigen wir die Umkehrfunktion, die zu einer gegebenen Antwortfunktion y die verursachende Erregerfunktion ermittelt. Diese Umkehrung kann formal mit Hilfe von Laplace-Transformationen gebildet werden und läßt sich als Faltung der Funktion y mit einer geeigneten Gewichtsfunktion g darstellen. Deshalb muß der Spezialprozessor folgenden Ausdruck berechnen.

$$z(t) = y(t) * g(t) = \int_{\tau=0}^{t} y(\tau) \cdot g(t-\tau) \mathrm{d}\tau$$

In der Praxis sind die beteiligten Funktionen als Vektoren von Funktionswerten dargestellt und die Faltungsintegrale werden näherungsweise durch Skalarprodukte berechnet. Dies führt schließlich zu folgendem Ausdruck für die Meßwertkorrektur

$$z_t = (y * g)(t) = \sum_{i=0}^{t} y_t g_{t-i} \cdot \Delta t \ .$$

Die dabei benötigten Koeffizienten g_{t-i} lassen sich leicht aus der Gewichtsfunktion $h(t)$ des Sensors berechnen. In der Praxis wird die Qualität der Näherung $z(t)$ natürlich von der Rechengenauigkeit des Chips und von der Anzahl der benutzten Meßwerte, also von der Dimension T des Gewichtsvektors g bestimmt.

3 Realisierungskonzept der Faltungsintegration

Der prinzipielle Algorithmus zur Berechnung einer Folge $(z_t)_{(0 \leq t \leq T)}$ von korrigierten Meßwerten ist in Abbildung 2 wiedergegeben. Wir wollen nun Umsetzungen des Algorithmus in geeignete Hardwarestrukturen vorstellen.

```
for t := 0 to T − 1 do
  begin
  Einlesen von y_t;
  s := 0;
  for i := 0 to t do s := s + g_i · y_{t−i};
  z_t := s;
  end;
```

Abbildung 2: Algorithmus zur Berechnung von Faltungen

Eine grundsätzlich mögliche Vorgehensweise bei der Umsetzung von Algorithmen in Hardware besteht darin, die verwendeten Programmvariablen durch Register oder Speicher zu realisieren, die über einen Bus mit einer Recheneinheit verbunden sind. Diese Methode ist universell einsetzbar, hat aber einige gravierende Nachteile:

- Es wird relativ viel Zeit für den Transport von Daten über den Bus benötigt und sofern nur recht einfache Rechenoperationen durchzuführen sind, stellt der Datenaustausch den Engpaß für die erreichbare Arbeitsgeschwindigkeit dar.
- Bei umfangreichen Speichern belegt der Bus eine relativ große Chipfläche. Dies hat negative Auswirkungen auf die benötigte Gesamtfläche und auf die Arbeitsgeschwindigkeit, da es zeitaufwendig ist, lange Leitungen zu treiben.
- Man benötigt einen Kontrollmechanismus, mit dem festgelegt wird, welche Daten jeweils auf den Bus zu legen sind. Dies kann etwa durch Adreßangaben in Befehlen erfolgen, die von der Recheneinheit entsprechend interpretiert werden. Die damit verbundene Befehlsdekodierung erfordert einen zusätzlichen Zeit- und Hardware-Aufwand, der nicht durch die Anwendung gerechtfertigt ist.

Um diese Nachteile zu vermeiden, soll der Algorithmus aus Abbildung 2 so implementiert werden, daß keine langen Datenübertragungsleitungen benötigt werden. Dazu werden die g und y-Daten statt in RAM-Speichern in zwei zyklischen Schieberegistern abgelegt, in denen sie gleichmäßig über den Chip wandern und jeweils zu den richtigen Zeitpunkten zur Bearbeitung an den Rechenwerken ankommen.

Um die Steuerung des Algorithmus einfach zu gestalten, werden alle Ergebniswerte z_t für $0 \leq t < T$ einheitlich über ein Skalarprodukt mit T-dimensionalen Vektoren

ermittelt, so daß zur Berechnung jeder Komponente die gleiche Rechenzeit benötigt wird. Noch nicht eingelesene Komponenten des T-dimensionalen y-Vektors werden zu Beginn mit 0 initialisiert.

Die nach jeder Skalarproduktberechnung erforderliche Verschiebung der g-Daten um eine Einheit gegenüber den y-Daten läßt sich recht einfach erreichen. Dazu werden die g-Daten um eine Komponente mit Wert 0 ergänzt und in einem zyklischen Schieberegister der Länge $T+1$ dargestellt. Werden nun beide Register mit der gleichen Geschwindigkeit getaktet, erhalten wir nach jeweils T Takten die gewünschte Verschiebung.

Diese Vorgehensweise führt zu der in Abbildung 3 angegebenen neuen Formulierung des Algorithmus. Man beachte, daß diese Variante einfacher ist als die Darstellung in Abbildung 2, da keine Indizierungen der Größen g und y vorkommen und die Daten jeweils an festen Stellen g' und y' der Schieberegister abgegriffen werden können.

Initialisieren des Registers g der Länge $T+1$ mit den Werten $(g_0, g_1, ..., g_{T-1}, 0)$;
Initialisieren des Registers y der Länge T mit $(0, 0, ..., 0)$
for $t := 0$ **to** $T-1$ **do**
  begin
  Einlesen von y_t an der Stelle y' des Registers y;
  $s := 0$;
  for $i := 0$ **to** T **do**
    begin
    $s := s + y' \cdot g'$
    shiftleft(g); shiftleft(y);
    end
  $z_t := s$;
  end;

Abbildung 3: Variante des Algorithmus mit Verwendung von Schieberegistern

Die Arbeitsweise dieses Algorithmus wird in der Abbildung 4 am Beispiel der Dimension $T = 4$ verdeutlicht. Dieses Beispiel zeigt ebenfalls, wie sich die Steuerung des Rechenwerks implementieren läßt. Offensichtlich muß die Berechnung eines Skalarprodukts dann beginnen, wenn der Wert y_0 am Rechenwerk anliegt und die Berechnung kann abgeschlossen und das Ergebnis ausgegeben werden, nachdem der Wert g_0 benutzt wurde.

Deshalb kann man die Aktionen des Rechenwerks (Initialisierung, Rechnung und Ergebnisausgabe) leicht mit Hilfe einer zusätzlichen Kennung bei den Daten in den beiden Schieberegistern steuern. Dieser einfache Synchronisationsmechanismus hat außerdem zur Folge, daß das im wesentlichen aus einem Multiplizierer und einem Addierer bestehende Rechenwerk an beliebige Stellen der Schieberegister plaziert werden

kann. Es ist deshalb grundsätzlich auch möglich, mehrere Rechenwerke an verschiedenen Stellen parallel zu betreiben, um die Gesamtrechenzeit zu reduzieren. Wir werden später im Kapitel 5 auf diese Beschleunigungsmöglichkeit zurückkommen. Zunächst soll jedoch genauer auf die Realisierung der zyklischen Schieberegister eingegangen werden.

Aufgrund der geforderten Flexibilität beim Einsatz des Chips werden an die Schieberegister besondere Anforderungen gestellt. Da nämlich in der Praxis die Vektordimension T eine problemspezifische Größe ist, muß T nachträglich durch den Anwender wählbar sein. Deshalb benötigen wir für den Chip Schieberegister programmierbarer Länge.

Zeittakt	g'	y'	Berechnung	Ergebnis
0	g_0	y_0	$g_0 \cdot y_0$	z_0
1	0	y_1		
2	g_3	y_2		z_0
⋮	⋮	⋮		
8	g_2	y_0	$g_2 \cdot y_0$	
9	g_1	y_1	$g_1 \cdot y_1$	
10	g_0	y_2	$g_0 \cdot y_2$	z_2
11	0	y_3		
12	g_3	y_0	$g_3 \cdot y_0$	
13	g_2	y_1	$g_2 \cdot y_1$	
14	g_1	y_2	$g_1 \cdot y_2$	
15	g_0	y_3	$g_0 \cdot y_3$	z_3

Abbildung 4: Demonstration des Faltungsalgorithmus am Beispiel einer Faltung von $T = 4$-dimensionalen Vektoren

4 Schieberegister variabler Länge

In der Literatur findet man verschiedene Ansätze zur Realisierung von Schieberegistern variabler Länge. Beispielsweise kann man eine Auswahlschaltung benutzen (siehe [Muk86]) um zwischen mehreren Ausgängen eines Schieberegisters zu wählen. Dabei wächst der Hardwareaufwand linear mit der Anzahl wählbarer Registerlängen. Bei einer in [Dan83] vorgeschlagenen Methode kommt man zwar ohne eine Auswahlschaltung aus, benötigt aber kompliziertere Registerzellen und eine spezielle Steuerung, über die die Shiftrichtung beeinflußt werden kann.

Den geringsten Hardwareaufwand benötigt eine in [OKD79] vorgestellte Methode, bei der man mit Hilfe von nur $\log m$ Multiplexern zwischen m verschiedenen Registerlängen wählen kann. Allerdings werden im dort angegebenen Layout einige relativ

lange Datenleitungen benutzt, die sich negativ auf die verwendbare Taktrate des Schieberegister auswirken können. Wir stellen deshalb einen günstigeren Entwurf vor, der bei gleichem geringen Hardwareaufwand nur Datenleitungen zwischen benachbarten Registerzellen benötigt. Zur Vereinfachung der Darstellung beschränken wir uns zunächst auf Schieberegister der Wortbreite 1 Bit. Durch Parallelschaltung mehrerer solcher Register erhält man dann beliebige Wortbreiten.

Wie in Abbildung 5 dargestellt, besteht das Schieberegister programmierbarer Länge im wesentlichen aus einer Kette von Registerzellen, die so in Falten gelegt ist, daß Falten der Längen $1, 2, 4, \ldots, 2^k$ entstehen. Dies entspricht einer Gesamtlänge von $N = 1 + 2 + 4 + \ldots + 2^k = 2^{k+1} - 1$ Registerzellen.

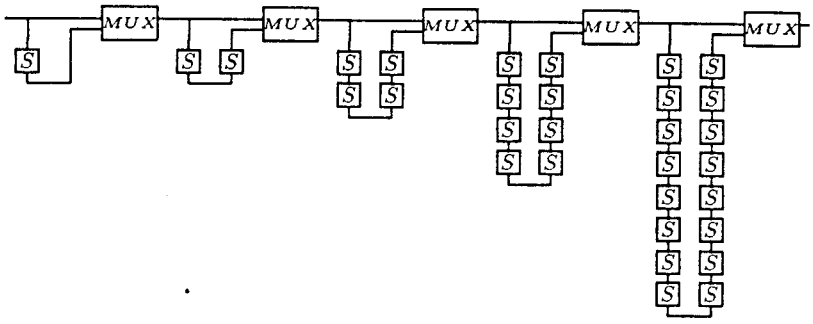


Abbildung 5: Schieberegister variabler Länge n mit $0 \leq n \leq 2^{k+1} - 1$.

Um die Registerlänge im Intervall von 0 bis $2^{k+1} - 1$ frei wählen zu können, wird für jede Falte ein 2:1-Multiplexer eingeführt, mit dem man steuern kann, ob die Falte zum Shiften benutzt werden soll, oder ob sie übersprungen werden soll. Die Binärdarstellung $(n_k, n_{k-1}, \ldots, n_0)$ der gewünschten Registerlänge n (mit $0 \leq n \leq N$) gibt dann an, durch welche Falten die Daten geshiftet werden müssen, damit sie exakt n Registerzellen durchlaufen.

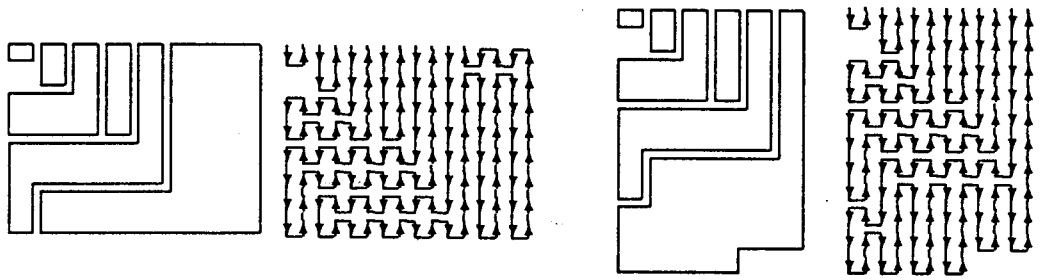
Da die Schieberegister einen wesentlichen Anteil an der Gesamtfläche des Spezialprozessors haben, muß ihr Layout besonders platzsparend realisiert werden. Wegen der besonders einfachen Schaltungsstruktur hat man dazu jedoch auch besonders viele Freiheitsgrade zur Verfügung. Sie sollen ausgenutzt werden, um die Register an die jeweils verfügbare Layoutfläche anzupassen, so daß etwa auch Lücken zwischen anderen Layoutteilen, wie etwa den Multiplizierern, ausgenutzt werden können.

Um diese Flexibilität zu erreichen wurde ein einfacher Layoutgenerator implementiert. Er ermittelt zunächst mit der in Abbildung 6 etwas vereinfacht dargestellten Prozedur *place* für jede benötigte Teilkette des Schieberegisters ausgehend von einer Layoutpostion (x, y) ein aus *length* Einheiten bestehendes freies Gebiet. Dabei werden bevorzugt Positionen mit kleiner x- und y-Koordinate verwendet, so daß insgesamt ein relativ kompaktes Layout entsteht. Das so ermittelte Gebiet wird anschließend mit vorgegebenen handdesignten Registerzellen gefüllt und gemäß einem einfachen Schema spaltenweise um die Verdrahtung zwischen den Zellen ergänzt.

In Abbildung 7a ist eine berechnete Gebietsaufteilung und die dazugehörige Verdrahtungsskizze am Beispiel eines Schieberegisters mit Falten der Längen 2, 4, 8, 16, 32 und 64 wiedergegeben. Abbildung 7b zeigt den entsprechenden Floorplan für das gleiche Beispiel bei Vorgabe einer geringeren Layoutbreite. Die Umsetzung der Skizzen in ein Layout ist recht einfach. Dazu ersetzt der Generator die Zellen und Verdrahtungsstücke der Skizze durch Plazierungen von Layoutteilen aus einer vorgegebenener Bibliothek. Schließlich ergänzt er dieses Layout um die Verdrahtung der Versorgungs- und Taktleitungen. Auf diese Weise erhält man je nach verwendeter Bibliothek wahlweise ein Layout in der Sea-of-Gates Technologie ([Beu89]) oder in einer CMOS Full-Custom Technologie. In den Abbildungen 8 und 9 sind Beispiele für so generierte Layouts wiedergegeben ([Leb90],[Loh91]).

```
procedure place(var length:integer; x,y:integer);
{Die Prozedur "place" versucht die Layoutfläche einer Kette der Länge "length"
beginnend bei der Position (x,y) zu plazieren. Die bereits benutzen Layout-
positionen sind in der globalen Matrix "layout" dargestellt.}
begin
if layout[x,y] ist frei
      then begin
            reserviere die Position (x,y);
            length := length - 1;
            {Versuche Layoutfläche für den Rest der Kette zu reservieren}
            if length > 0 then place(length, x - 1, y);
            if length > 0 then place(length, x, y - 1);
            if length > 0 then place(length, x, y + 1);
            if length > 0 then place(length, x + 1, y);
           end;
      end;
```

Abbildung 6: Prinzipieller Plazierungsalgorithmus für die Zellenblöcke einer Kette vorgegebener Länge.



a) b)

Abbildung 7: Berechnete Layout- und Verdrahtungsstrukturen für ein Schieberegister bei unterschiedlicher Vorgabe für die Layoutbreite.

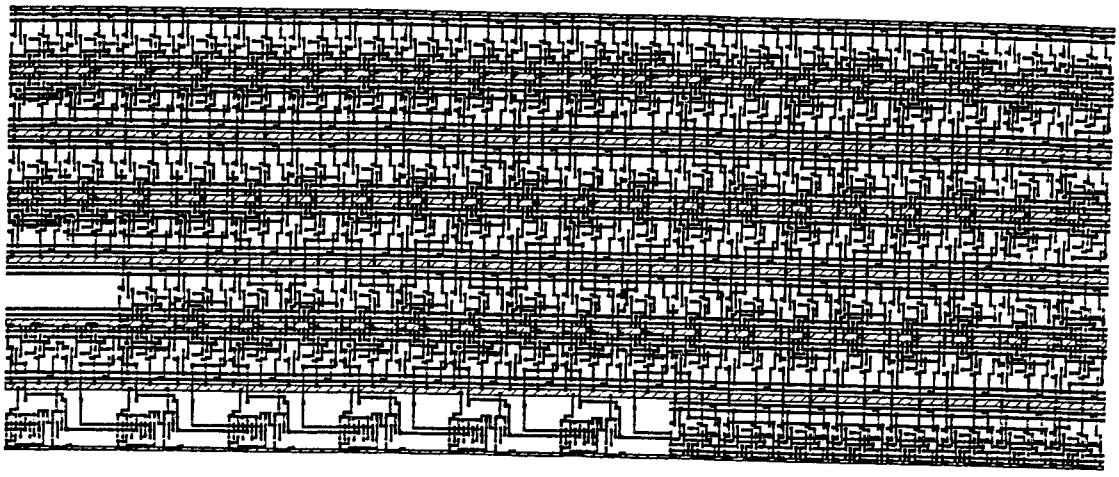


Abbildung 8: Generierte Verdrahtungsebenen eines Sea-of-Gates Layouts für den GATE-FOREST Master des IMS.

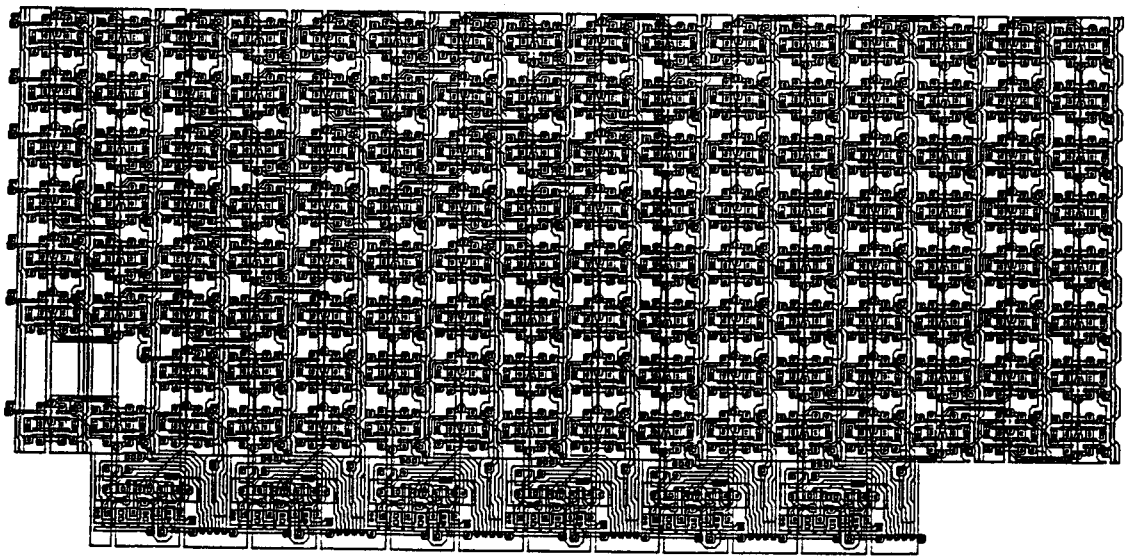


Abbildung 9: Generiertes Full-Custom Layout für einen 2μ CMOS Prozeß.

5 Realisierung der Multiplikation

Die wesentlichen arithmetischen Operationen des zu entwickelnden Chips sind die Multiplikation und das Aufsummieren von Partialprodukten. Da die Datenzuführung zum Rechenwerk mit Hilfe der gefalteten Schieberegister sehr schnell durchgeführt werden kann, sollen die Multiplizierer so betrieben werden, daß sie möglichst die volle Schieberegistergeschwindigkeit ausnutzen können und mit jedem Takt eine neue Multiplikation beginnen können. Dies bedeutet nicht, daß während eines Taktes eine komplette Multiplikation durchgeführt werden muß. Stattdessen ist es ausreichend, das Rechenwerk in der Pipelining-Technik zu betreiben, so daß mit einem Multiplizierer mehrere Multiplikationen gleichzeitig durchgeführt werden können. Ein solcher Multiplizierer mit hoher Durchsatzrate wird beispielsweise in [NSKE86] vorgestellt. Dort wird ausßerdem auch auf die geschickte Realisierung der Grundzellen eingegangen. Die Taktgeschwindigkeit des Multiplizierers kann dabei so hoch gewählt werden, daß die Taktlänge nur der Verzögerungszeit eines Volladdierers entspricht.

Für unsere Implementierung kann man zusätzlich ausnutzen, daß die Multiplikationsergebnisse lediglich Zwischenergebnisse von Faltungsberechnungen sind, und deshalb nicht in redundanzfreier Binärdarstellung dargestellt werden müssen. Die Normierung der Ergebnisdarstellung kann platzsparender und ohne Geschwindigkeitsverlust in einem nachgeschalteten sequentiellen Addierwerk erfolgen.

Eine weitere Erhöhung der Arbeitsgeschwindigkeit des Chips läßt sich durch den parallelen Einsatz mehrerer Multiplizierer erreichen. Ein solcher Ansatz wird beispielsweise in [Hor88] vorgestellt. Dort werden bei der Faltung von $(y_0, y_1, \ldots, y_{n-1})$ mit $(g_0, g_1, \ldots, g_{n-1})$ im i-ten Schritt gleichzeitig alle n Multiplikationen $y_i \cdot g_0$, $y_i \cdot g_1$,... $y_i \cdot g_{n-1}$ durchgeführt, was als Nebeneffekt dazu führt, daß der Vektor y auf dem Chip nicht gespeichert werden muß. Dabei muß sich die angegebene Implementierung jedoch auf eine feste Dimension von $n = 32$ beschränken.

Da unser Chip auch mit wesentlich größeren Dimensionen n arbeiten können soll, benutzen wir eine etwas andere Vorgehensweise. Bei ihr wird ein Koeffizient g_i aus einem zyklischen Schieberegister gleichzeitig mit m verschiedenen Eingabewerten y_j multipliziert und dies so oft wiederholt, bis alle n Produkte ermittelt sind. Dabei ist m die Gesamtzahl der verfügbaren Multiplizierer.

Wir erhalten dann nach jedem Zyklus der Länge von $(n+m)$ Takten m Ausgabewerte. Folglich beträgt die Datenein- und Ausgaberate $\nu \cdot \frac{m}{n+m}$, wobei ν die Taktrate des Chips darstellt. Die Gesamtlaufzeit zur Berechnung einer n-dimensionalen Faltung beträgt $(n+m) \cdot \lceil \frac{n}{m} \rceil$ Takte.

Die maximale Rechenleistung des Chips kann durch $\frac{m}{\nu}$ als die Anzahl der maximal pro Sekunde durchführbaren Multiplikationen bestimmt werden. Sinnvoller ist jedoch die Angabe der effektiven Rechenleistung des Chips, bei der lediglich die tatsächlich zur Faltungsberechnung benötigten Multiplikationen berücksichtigt werden. Für sie ergibt sich

$$\text{effektive Rechenleistung} = \begin{cases} \frac{\nu}{2} \cdot n & \text{falls } n \leq m \\ \frac{\nu}{2} \cdot \frac{n+1}{1+\frac{n}{m}} & \text{falls } n > m \end{cases}$$

Beispielsweise bedeutet das für eine Taktrate von 10 Mhz bei $m = 32$ Rechenwerken und einer Dimension von $n = 128$ eine effektive Rechenleistung von 129 MOPS (129 Millionen Operationen pro Sekunde). Die theoretische Rechenleistung beträgt dabei $32 \cdot 10 \cdot 10^6 = 320$ MOPS.

Man beachte, daß diese Rechenleistung mit einer relativ geringen Ein- Ausgaberate des Chips erreicht wird. Sie beträgt im angegebenen Beispiel lediglich $\nu \cdot \frac{m}{n+m} = 2$ Mhz. Das bedeutet, daß mit dem vorgestellten Konzept auch wesentlich höhere Taktraten mit entsprechend größerer Rechenleistung praktikabel sind.

Literaturverzeichnis

[Beu89] M. A. Beunder. *Design and Analysis of Semi-Custom Architectures.* Dissertation, Institut für Mikorelektronik, Stuttgart, 1989.

[Dan83] P. E. Danielsson. A Variable-Length Shift-Register. *IEEE Transactions on Computers*, C-32(11):1067–1069, November 1983.

[Hor88] G. Horvath. Digitale Daten in Echtzeit verarbeiten. *Elektronik*, 22(28.10.88):135–142, Oktober 1988.

[Leb90] P. Leber. *Entwurf und Realisierung eines dynamischen Schieberegisters variabler Länge in der Sea-of-Gates Technologie des IMS.* Diplomarbeit, Fachhochschule Furtwangen, 1990.

[Loh91] T. Lohscheidt. *Full-Custom Entwurf für ein Schieberegister variabler Länge.* Diplomarbeit, Fachhochschule Furtwangen, 1991.

[Muk86] A. Mukherjee. *Introduction to NMOS and CMOS VLSI Systems Design.* Prentice Hall International, 1986.

[NSKE86] T. G. Noll, D. Schmitt-Landsiegel, H. Klar, and G. Enders. A Pipelined 330-MHz Multiplier. *IEEE Journal of Solid-State Circuits*, SC-21(3):411–416, Juni 1986.

[OKD79] N. Ohwada, T. Kimura, and M. Doken. LSI's for Digital Signal Processing. *IEEE Journal of Solid-State Circuits*, SC-14(2):214–220, April 1979.

4. Entwicklung eines ASIC für einen GPS-Empfänger

Hans-Peter Behrens, Fachhochschule Offenburg

An der FH Offenburg arbeiten seit Ende 1989 in einem Team die Professoren Dr. Jansen, Dr. Schüssele, die wissenschaftlichen Mitarbeiter Bernd Reinke, Martin Jörger und die Diplomanden Hans Fiesel, Otmar Feißt an dem Entwurf eines Navigationsempfängers. Im Rahmen dieses Projekts, genannt GPS-Projekt (GPS = Global Positioning System), wurde im Herbst 1990 ein experimenteller Empfänger in Betrieb genommen. Nachdem die Testergebnisse gezeigt hatten, daß das Konzept der Anlage stimmte, ging es nun um die Miniaturisierung, Integration und Optimierung der Schaltung. Außerdem sollte der bisher verwendete PC durch einen auf der Platine befindlichen Mikroprozessor ersetzt werden. Im Zusammenhang mit dem GPS-Projekt wurden bisher im Offenburger ASIC-Labor eine Analogschaltung auf einem B500, drei LCA Designs und diverse GAL's entwickelt.

Zur Zeit arbeiten mehrere Diplomanden an der zweiten Generation des Empfängers.
Meine Aufgabe besteht darin, die dort noch in drei LCA's untergebrachte digitale Logik sowie einen Teil des bisherigen PC-Interface in einem IMS Gate Forrest zu integrieren. Außerdem muß die Logik von 8 Bit auf einen 16 Bit breiten Datenbus umgestellt und an die neue Peripherie des Mikroprozessors angepasst werden. Damit soll die jetzige Digital-Platine noch weiter verkleinert werden (Bild 1). Wesentlich ist dabei die Umsetzung der zahlreichen Zähler- und Registerstrukturen in einen Gate Forrest. Als Arbeitsmittel stehen Apollo Workstations mit Mentor Software zur Verfügung.

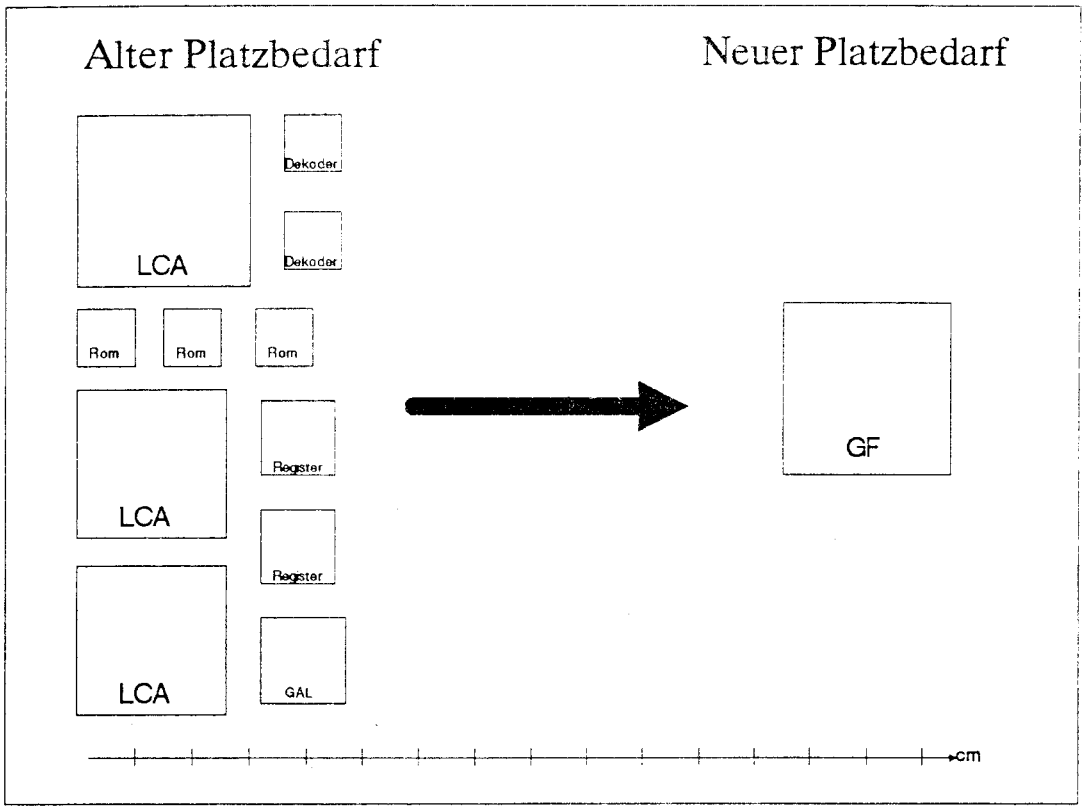


Bild 1: Platzvergleich Version 2 zu Version 3

Einführung:

Bei dem GPS-System handelt es sich um ein satellitengestütztes Navigationssystem, mit Spread Spectrum Modulation. Das System wurde ursprünglich für rein militärische Zwecke entwickelt, inzwischen aber zum Teil für zivile Zwecke freigegeben. Durch GPS wird es möglich, weltweit eine dreidimensionale Positionsbestimmung mit einer Genauigkeit von ca. 100m für zivile und 30m für militärische Anwendungen durchzuführen. Dies wird jedoch erst nach einer vollständigen Installation des Systems mit 18 Satelliten möglich sein. Diese werden die Erde in ca. 20200 km Höhe auf 6 verschiedenen Umlaufbahnen umkreisen. Drei weitere Satelliten sind als Reserve vorgesehen (Bild 2).

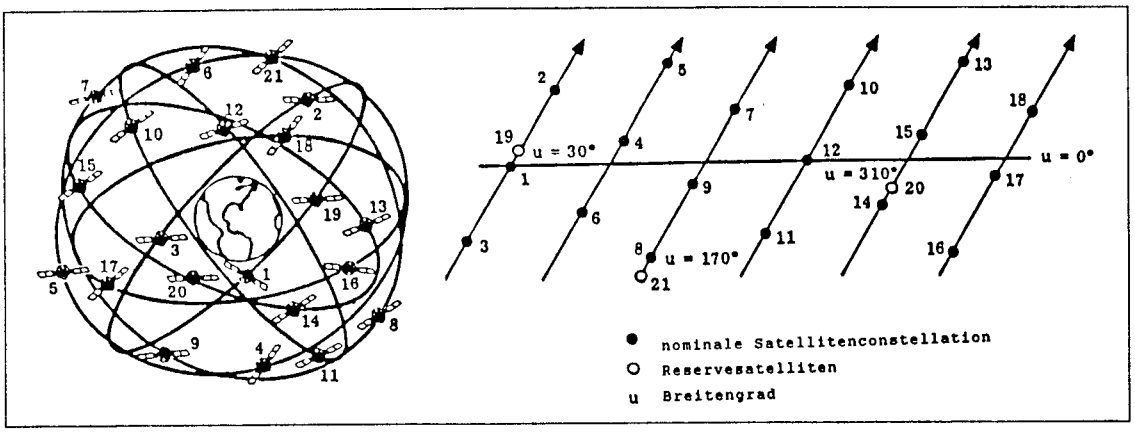


Bild 2: Satellitenlaufbahnen [2]

Um eine dreidimensionale Ortsbestimmung durchführen zu können, muß die Entfernung zu mindestens drei Satelliten sowie die genaue Position jedes dieser Satelliten bekannt sein. Zur Abstandsmessung wird die Signallaufzeit eines vom Satelliten ausgesendeten Datenstroms und die Ausbreitungsgeschwindigkeit elektromagnetischer Wellen verwendet. Da zwischen Empfänger- und Satellitenzeit keine Differenz auftreten darf, müßte auch der Empfänger genau wie die Satelliten mit einer Atomuhr ausgerüstet sein. Um diesen Aufwand zu umgehen, mißt man die Entfernung zu einem vierten Satelliten und erhält somit ein System von vier Gleichungen, aus denen Ort und Zeit bestimmt werden können.

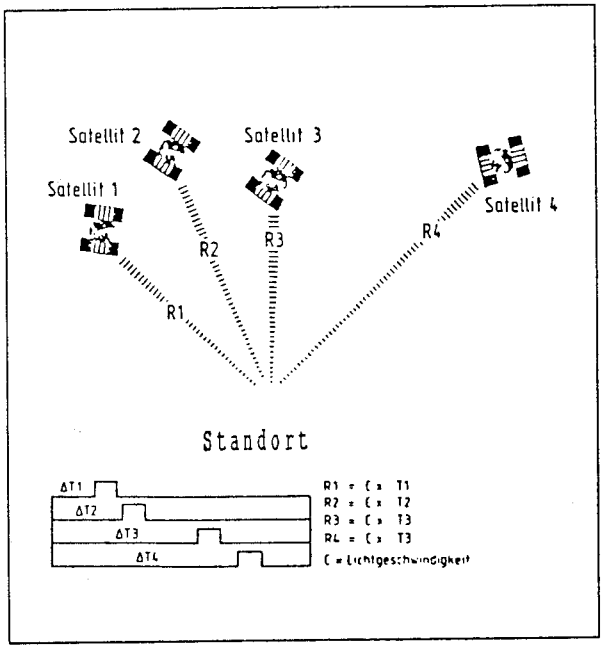


Bild 3: 3D-Ortsbestimmung [2]

Satellitensignale:

Alle Satelliten senden zwei Signale L1 und L2, zur Datenübertragung werden die Frequenzen f1 = 1575,42 MHz und f2 = 1227,6 MHz verwendet. Diese Frequenzen sind Vielfache des Systemgrundtaktes F0 = 10,23MHz. Für das Signal L1 wird der Träger f1 mit zwei Codefolgen, dem C/A(Coarse Aquissition) und P(Precision) -Code moduliert und zwar durch die Quadratur-Phasenmodulation. Die Modulation aus dem Träger f2 und dem P-Code bildet das Signal L2. Der P-Code und das Signal L2 sind jedoch der militärischen Nutzung vorbehalten. Als minimaler Empfangspegel wird für den C/A-Code -130dBm angegeben.

C/A-Code:

Der C/A Code ist ein Goldcode, der aus der Modulo-2-Addition (Exor) zweier 1023 Bit Pseudo-Random-Noise (PRN) Codes gewonnen wird. Die Folgen G1 und G2 werden von jeweils einem linear rückgekoppelten 10 Bit Schiebregister generiert. Getaktet werden die Schieberegister mit einem Zehntel des Grundtaktes, also 1,023 Mhz. Mit dem X1-Epoche Signal wird der C/A Code synchronisiert.

Jeder Satellit sendet einen anderen C/A Code. Die unterschiedlichen Codes werden dadurch erzeugt, daß die Folge G2 um eine ganzzahlige Anzahl von C/A Takten verzögert wird. Durch die Exorverknüpfung zweier Abgriffe des Schieberegisters erhält man solche verzögerten G2 Folgen.

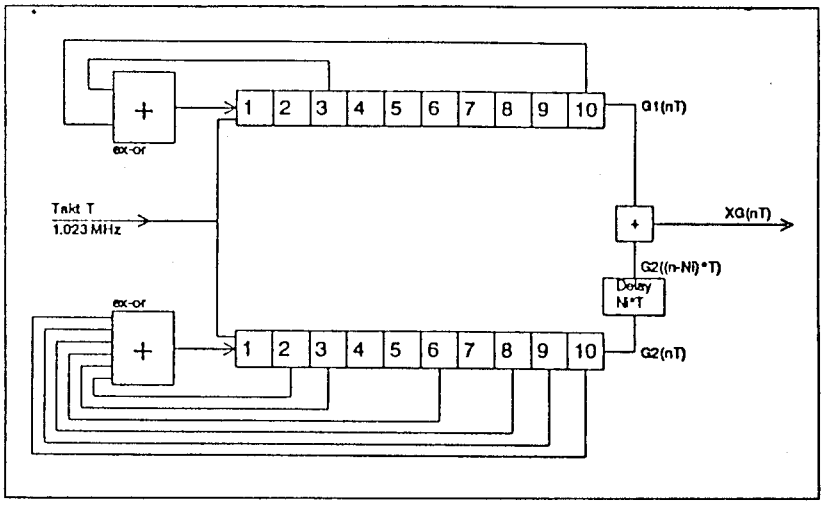


Bild 4: C/A-Code-Erzeugung [2]

Empfangskonzept der FH:

Zur Auswertung des Satellitensignals ist es notwendig, das Empfangssignal mit einem intern erzeugten C/A Code zu korrelieren. Dieser interne Code muß dem Satellitensignal entsprechen und mit ihm in Phase gebracht werden. Hierfür benötigt der GPS-Empfänger einen Codegenerator und eine Codesynchronisationseinheit. Bei der Verknüpfung beider Signale ergibt sich ein Signalanteil entsprechend der Verschiebung beider Codes zueinander. Zuerst wird die Empfängercodephase solange variiert, bis ein Korrelationssignal einen von Null verschiedenen Anteil besitzt. Der Korrelationspegel wird nach dem Pegeldetektor entsprechend der Codephase mit plus/minus 1 oder 0 bewertet. Ist der Mittelwert des Ausgangssignals dieser Bewertung gleich Null, dann sind die Codes in Phase (Prinzip des Tau-Dither-Loop).

Blockstruktur der gesamten Schaltung:

Bild 5 zeigt ein vereinfachtes Blockschaltbild des digitalen Empfängerteils. Außer dem Prozessor und dessen speziellen Peripheriebausteinen sowie einigen A/D-, D/A-Wandlern befinden sich alle digitalen Schaltungen in einem Chip.

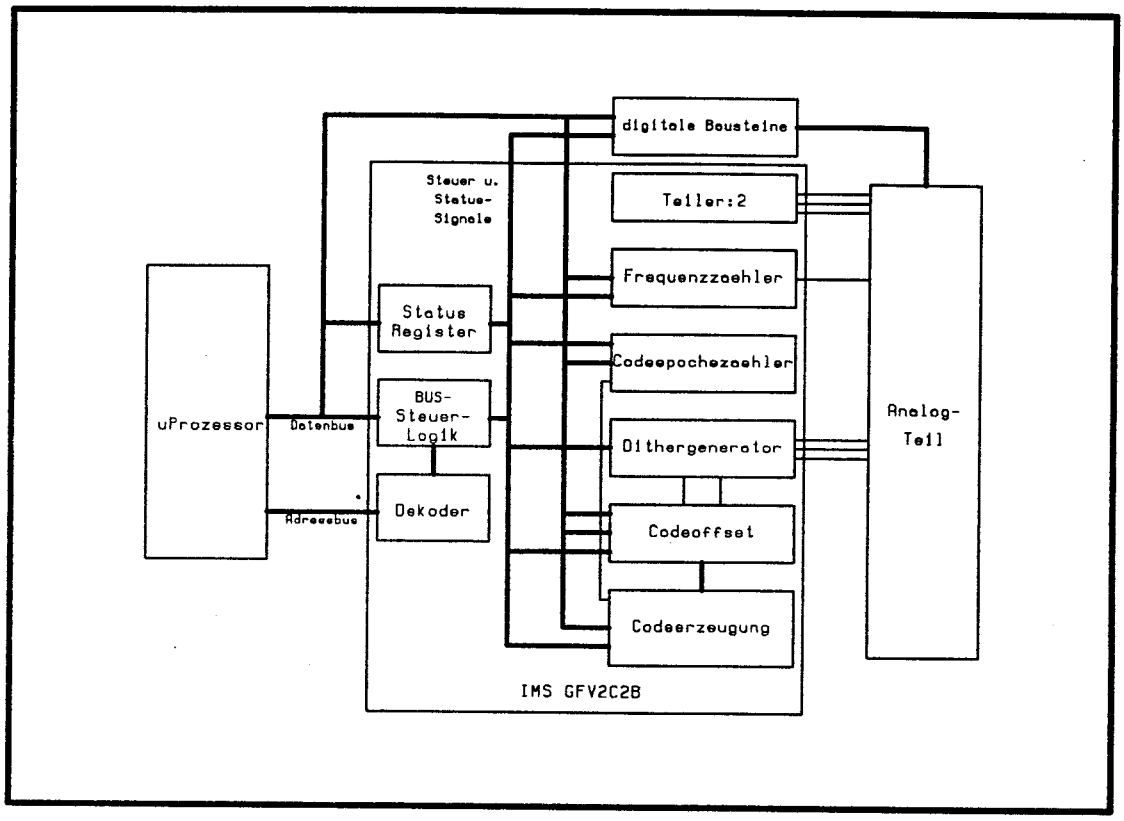


Bild 5: Blockschaltbild des ASIC

Kurze Beschreibungen der einzelnen Blöcke:
Codeerzeugung:

Mit zwei *10 Bit Schieberegistern* und jeweiligen EXOR-Matrizen wird der C/A-Code generiert. Um die je nach Satellitennummer unterschiedlichen C/A-Codes zu erhalten, wird der vom Prozessor gewünschte Parallelabgriff des *G2-Schieberegisters* im *G2-Codeselect-Register* abgespeichert, mit der *G2-Tab-Selektion* abgegriffen und EXOR-verknüpft. Der Takt für die Schieberegister wird in einem *Teiler* (Systemtakt:10) erzeugt und mit dem 1 ms Set-Enable-Signal (SE) synchronisiert.
Außerdem befinden sich im Codegenerator noch weitere Baugruppen, die zur Synchronisation des internen mit dem externen C/A-Code benötigt werden. Der *Systemtaktzähler 10,23MHz* erzeugt einen 14 Bit Vergleichswert zur Erzeugung des SE-Signals und liefert ein 1kHz Synchronsignal. Das 14-Bit-Signal wird mit dem vom *Codeoffset* gelieferten Verzögerungswert verglichen und erzeugt bei Gleichheit beider Werte das SE-Signal. Mit diesem Verzögerungswert kann der C/A-Code in 100ns-Schritten verschoben werden.

Codeoffset:

Diese Baugruppe liefert der *Codeerzeugung* einen statisch anstehenden 14 Bit Verzögerungswert. Dieser kann vom Prozessor vorgegeben und bei Dither Count Enable (DCE), in Abhängigkeit der Komperatoreingänge inkrementiert bzw. dekrementiert werden. Dabei steuert das *Stellglied* das Auf-/Abwärtszählen des Verzögerungswertzählers. Eine *Handshakeschaltung* regelt die Übernahme eines Wertes vom Prozessor in den Verzögerungwertzähler.

Dithergenerator:

Der C/A-Code durchläuft ein *11 Bit Schieberegister* mit drei Steuereingängen. Mit den im *Dithergenerator* erzeugten Steuersignalen kann der Code in 500ns-Schritten verschoben werden.

Codeepochezähler:

Der *Codeepochezähler* dient der Unterteilung der X1-Epochen in Millisekundenschritte, um eine eindeutige Zeitskalierung zu erhalten. Der Zähler durchläuft einen 0 bis 1499 Zyklus und wird mit jeder neuen Codeepoche durch das SE-Signal inkrementiert.

Frequenzzähler:

Er zählt die Frequenz des spannungsgesteuerten Quarzoszillators zur Messung der Dopplerverschiebung. Ein *Torsignalgenerator* zählt das 1ms SE-Signal und erzeugt damit alle 2 Sekunden einen Impuls. Mit diesem übernehmen die *Ausgangsregister* den Zustand des Frequenzzählers und der *Frequenzzähler* wird zurückgesetzt.

Teiler:

Dieser Block erzeugt Ansteuersignale für den Costas-Loop. Dabei handelt es sich um zwei 90^0 zueinander verschobene 5,115Mhz-Signale zur Versorgung der Regelschleife der Trägernachführung.

Dekoder:

Dekodierung der Adressen für die Bussteuerung.

Bussteuerlogik:

Die Bussteuerung regelt den Datenverkehr zwischen den einzelnen internen- bzw. externen Bausteinen und dem Controller.

Statusregister:

Für den Prozessor wichtige Statussignale werden hier zwischengespeichert.

Realisierung der Schaltung mit Mentor-Software:

Es bot sich an die bereits vorhandene Aufteilung in Blöcke beizubehalten und zu verfeinern, da es sich um eine sehr komplexe Schaltung mit einer großen Zahl an Registerstufen und Zählern handelt. Die IMS-Bibliothek bietet bisher keine Makros für Gate-Forrest an, deshalb gab ich häufig auftretende Strukturen einmalig ein, definierte sie als Block und verwendete sie wie ein Makro auf der untersten Hierachieebene der Schaltplaneingabe. Zum Beispiel definierte ich häufig benötigte Ein-/Ausgaberegisterstufen, Vier-Bit-Synchronzähler, sowie weitere Zähler- und Registereinheiten als "Makro-Block". Bei dem Aufbau dieser Blöcke wurde auf möglichst geringen Platzverbrauch geachtet. So wurden die Zählerstufen zwar den TTL Standardzählern nachempfunden, aber soweit möglich minimisiert und auf die platzsparenden negierten Standardbausteine der IMS-GF-Bibliothek umgestellt.
Die Funktion der einzelnen Blöcke wie auch der höherliegenden Schaltplanebenen wurde mit MENTOR-QUICKSIM überprüft und simuliert.

Bild 6 zeigt den *Frequenzzähler*. Dieser ist aus vier Zählermakros aufgebaut. Im Bild ist außerdem einer der 4-Bit-Synchronzähler-Blöcke sowie ein Ausschnitt aus der Quicksim-Simulation enthalten. Diese Zählerstrukturen konnten außer in dieser Baugruppe auch im *Codeepochezähler* bzw in der *Codeerzeugung* verwertet werden.

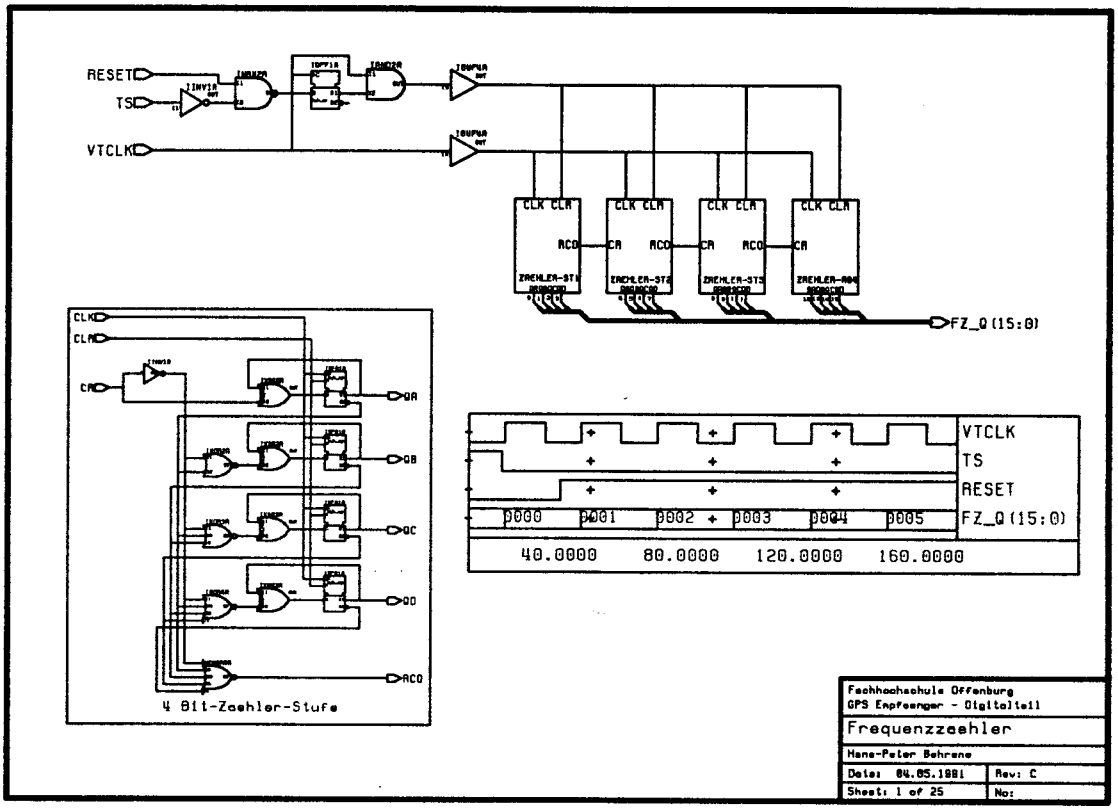


Bild 6: Frequenzzähler aufgebaut aus 4-Bit-Zählerstufen

Bild 7 zeigt die Ausgabeeinheit des *Frequenzzählers* auf den Datenbus. Eine Ausgabestufe besteht aus einem Multiplexer, einem D-Flip-Flop und einer Tristateausgangsstufe. Diese Einheit wurde von mir als Block definiert und ist für alle Ausgangsstufen auf den Datenbus benutzbar, so zum Beispiel im *Codeoffset* und *Codeepochezähler*.

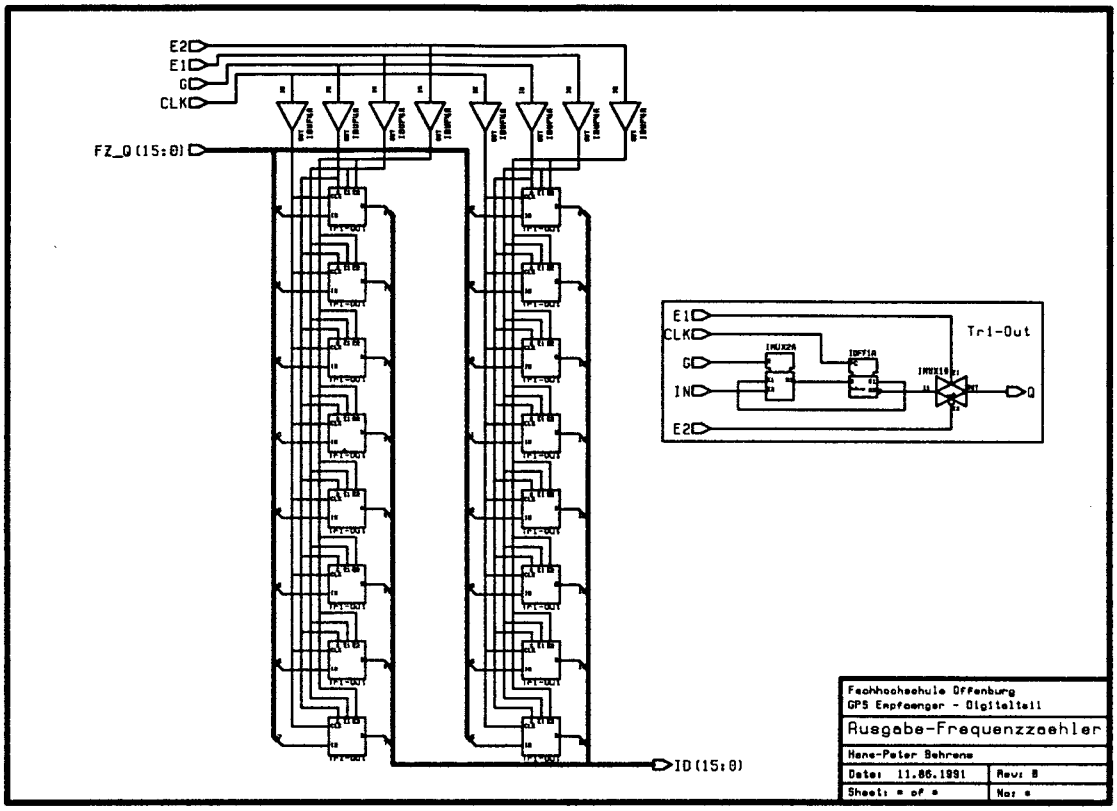


Bild 7: Ausgabeeinheit auf den Datenbaus

Berechnung der vorläufigen Chipkomplexität:

Folgende Standardzellen werden zum derzeitigen Stand der Schaltung benötigt:

Gatterbezeichnung	n * 17u Breite	Anzahl	Summe Sites
IINV1A	2	27	54
ICIN1B	5	55	275
IBUF1A	3	17	34
IBUF4A	7	55	385
INAN2A	3	19	57
INAN3A	4	11	44
INAN4A	6	14	84
INAN5A	7	5	35
INOR2A	3	32	96
INOR3A	5	17	85
INOR4A	6	14	84
INOR5A	7	11	77
IAND2A	5	35	175
IORO2A	5	23	115
IMUX2A	6	76	456
IXOR2A	5	103	515
IDFF1A	13	158	2054
IDFS1A	16	22	352
IDFR1A	17	75	1275
IPBF2A	15	11	165
ITPB2A	16	31	496
			6913

Würde man einen IMS-Gate-Forrest des Typs GFXX2 verwenden, so entspräche dies einem Ausnutzungsgrad von 47%. Hierbei ist jedoch der Platzbedarf für die Verdrahtungskanäle nicht miteinbezogen. Sollte die Integrierung der Schaltung auf dem GFXX2 nicht ohne Probleme möglich sein, wird erwogen den neuen GF-Baustein mit 1,2u Prozess zu benutzen.

Als Grundlage für den Entwurf der Schaltung stand mir die Diplomarbeit von H. Fiesel zur Verfügung. Außerdem möchte ich mich für die Unterstützung von Prof. Dr. Jansen und dem Laboringenieur B. Reinke bedanken.

Literaturverzeichnis:

1. Navstar GPS Joint Program Office, Los Angeles Air Force Station, US Air Force Space Division/CWNI, Indroduction to Navstar GPS User Equipment 1988

2. Spilker, J. J., Global Positioning System: Signal Structure and Performance Characteristics, Stanford Telecommunications Inc., 1979

3. Summ, Patrik, Entwicklung eines GPS-Empfängers, FHO Forschungsbericht 1/90

4. Fiesel, Hans, Digitalteil eines GPS-Empängers, Diplomarbeit SS90

Fachhochschule Aalen

Fachbereich Elektronik

5. *Einführung in die Hardwarebeschreibungssprache VHDL*

Verfasser: Bernd Mößner

Betreuer: Prof. Dipl.-Ing. R. Rudloff

Präsentation: MPC-Workshop an der FH Esslingen/Göppingen 10.06.1991

Inhaltsverzeichnis

Inhaltsverzeichnis *Fortsetzung*

Bildverzeichnis

Tabellenverzeichnis

1. Einleitung

Die zunehmend komplexer werdenden digitalen Schaltkreise, wie beispielsweise VLSI-IC's[1], erfordern zu ihrem Entwurf Entwurfsprozesse, die trotz der steigenden Komplexität überschaubar bleiben. Die Verwendung von hardwarebeschreibenden Sprachen ist hierfür eine der hilfreichsten Methoden. Diese Sprachen haben zwei Hauptanwendungsgebiete, zum einen unterstützen sie die Struktur eines Entwurfs und zum anderen beschreiben sie ihn.

Gefördert durch das amerikanischen Verteidigungsministeriums wurde die Hardwarebeschreibungssprache VHDL[2] ab 1983 im Rahmen des VHSIC[3]-Programms entwickelt. Das amerikanische Normungsgremium IEEE[4] veröffentlichte die Sprache im Dezember 1987 als Standard unter der Norm IEEE-STD-1076-1987. VHDL basiert auf der Programmiersprache ADA[5]. Für die schnelle Verbreitung von VHDL war vor allem das Verteidigungsministerium der USA verantwortlich, welches im MIL-STD 454 Standard vorschreibt, daß alle IC's, die mit Mitteln des Pentagons entwickelt werden, mit VHDL beschrieben werden müssen.

Ursprünglich wurde VHDL entwickelt, um als Kommunikationsmittel beim Entwurf eines Systems zwischen den verschiedenen Auftragnehmern des amerikanischen Verteidigungsministeriums zu dienen. Inzwischen sind von der Industrie und von einzelnen Neuentwicklern so viele Ergänzungen in VHDL eingeflossen, daß sie alle Merkmale aufweist, die eine hardwarebeschreibende Sprache haben sollte. VHDL kann in allen Phasen eines elektronischen Entwurfsprozesses zum Einsatz kommen. Da sie sowohl maschinenlesbar als auch vom Bearbeiter lesbar ist, unterstützt sie den Entwurf, die Überprüfung, die Synthese und das Testen einer Hardware genauso wie den Austausch von Hardware-Entwurfsdaten, die Wartung, die Änderung, die Fertigung und die Beschaffungsphase. Darüberhinaus lassen sich praktisch beliebige physikali-

1 Very Large Scale Integration-Integrated Circuits

2 VHSIC Hardware Description Language

3 Very High Speed Integrated Circuit

4 Institute of Electrical and Electronics Engineers

5 Höhere prozedurale Programmiersprache zur Echtzeitdatenverarbeitung

sche (z.B. mechanische oder thermodynamische) Systeme beschreiben.

Jeder Entwurf eines Systems beginnt mit dessen Spezifikation durch Festlegung der Systemfunktion (Logik) und den Systemeigenschaften (z.B. die Treiberstärken der Ausgänge). Diese Spezifikation wird in eine Hardwarebeschreibungssprache umgesetzt, wie beispielsweise VHDL, die von Simulationsprogrammen (z.B. Quicksim II) zur Simulation des zeitabhängigen Verhaltens des beschriebenen digitalen Systems unmittelbar benutzt werden kann. Daraus resultiert dann die erste sehr abstrakte Beschreibung des Systems, anhand derer mittels geeigneter Testvektoren die Spezifikationen des zu entwickelnden Systems überprüfbar wären. Im nächsten Schritt wird die Architektur entworfen, d.h das System wird in Funktionsblöcke wie beispielsweise Adressdekodierer, ALU[1], Speicher und Steuerwerk unterteilt, um einen Bezug zur Hardware festzulegen. Zur Findung der geeignetsten Lösung müssen hierbei mehrere Architekturansätze durchdacht und auf ihre Tauglichkeit hin überprüft werden. Die im ersten Schritt gewonnenen Testvektoren dienen hierbei wieder als Testgrundlage. Im übrigen kann es während des gesamten Entwurfsprozesses notwendig werden, zu dem jeweilig gewonnenen Architekturansatz Alternativen zu entwerfen. Im dritten und längsten Schritt entsteht aus der gewählten Architektur durch sukzessives Verfeinern der Strukturen das Gesamtsystem. Es ist dabei besonders wichtig, die Kontrolle über den Prozeß zu bewahren. Dies ist um so wichtiger, als im Laufe der Zeit die Anzahl der am Projekt arbeitenden Personen wächst. Jede dieser Persomen kann die Funktionsweise ihres Teilentwurfes am Gesamtentwurf simulieren.

Mit den heute existierenden CAE-Hilfsmitteln[2] können einzelne ASIC's so entwickelt werden, daß die ersten fertiggestellten IC's die dazugehörigen Funktionstests vollständig passieren. 67% all jener Chips jedoch arbeitet inkorrekt oder zumindest unvollständig im Systemverbund [12]. Mit Hilfe von VHDL kann der zu entwickelnde ASIC bereits nach der ersten Beschreibung seiner Funktionsweise im Systemverbund simuliert werden. Bei der traditionellen Methode, mittels der Schaltplaneingabe durch Neted, dagegen müssen die Gatterbeschaltungen festgelegt werden, bevor der Entwurf geprüft werden kann.

1 Arithmethical-Logical-Unit

2 Computer Aided Engineering

Ein geeignetes Verfahren zur Gewährleistung kontinuierlicher Kontrolle der Funktionsweise des Systems während der Entwicklungszeit ist die sogenannte Top-Down-Design-Methode. Verfährt man nach ihr, so wird aus der jeweiligen abstrakten Beschreibung der Schaltung mit VHDL die nächste Ebene mit verfeinerter Struktur abgeleitet. Die Mixed-Level-Simulation ist hierbei das Schlüsselinstrument. Mit der Mixed-Level-Simulation ist es möglich, die veschiedenen Entwicklungsschritte hierarchisch zu verifizieren. Darunter versteht man in diesem Zusammenhang eine Testphilosophie, die auf der hierarchischen Natur des Top-Down-Designs basiert. Das stellt sicher, daß jede neu entworfene untergeordnete Ebene mit verfeinerter Struktur immer wieder zusammen mit der übergeordneten Ebene simuliert werden kann. Man kann so leicht überprüfen, ob die gerade in der untergeordneten Ebene implementierte Unterfunktion noch mit dem Gesamtkonzept übereinstimmt. Es gibt also während der Entwicklung zu jeder untergeordneten Ebene der Schaltung einen Satz an Testvektoren, die direkt von den Eigenschaften des jeweiligen übergeordneten Ebene, die ja bereits simulierbar war, abgeleitet sind. Somit wird praktisch ausgeschlossen, daß die Entwicklung des Chips an der Spezifikation vorbeiführt. Am Ende des Top-Down-Designs steht die nochmalige Gesamtsimulation des Systems auf Gatterebene.
VHDL ist in der Lage, den Entwurf auf seinem gesamten Weg zu begleiten. Dies ist notwendig, da ein eventuelles Umsetzen eines Entwurfs von einer Datenbasis auf eine andere die Kontinuität unterbrechen würde. Eine solche Umsetzung würde eintreten, wenn der Entwurf auf Gatterebene mit Neted geschieht. Um eine Simulation auf Gatterebene durchführen zu können ist es somit notwendig, daß die Bauteilbibliotheken der ASIC Hersteller in VHDL beschrieben sind.

1.1 Sprachumfang

VHDL bietet dem Anwender 3 grundsätzliche Hardwarebeschreibungsformen. Mit Hilfe dieser 3 Beschreibungsformen eines Systems wird die Mixed-Level-Simulation, d.h. die gleichzeitige Simulation des Entwurfes auf verschiedenen Ebenen, ermöglicht.
Die erste und zugleich die abstrakteste, d.h. die hardwareentfernteste Beschreibungsform, ist das sogenannte Behavioral-VHDL oder Verhaltensorientierte-VHDL. Mit ihr lassen sich abstrakte Beschreibungen einer Schaltung auf der Ebene einer modernen Programmiersprache, vergleichbar mit Pascal oder C, aufbauen. Die in VHDL verwurzelten Sprachkonzepte erlauben hierbei

dem Anwender, die zu implementierende Funktion algorithmisch zu beschreiben. Ein ASIC läßt sich damit vollkommen technologieunabhängig beschreiben, dies eröffnet dem Anwender den Freiraum, der für die Funktionsentwicklungsphase notwendig ist. In Bild 1 ist ein Volladdierer als verhaltensorientiertes Modell abgebildet. Die VHDL-Anweisung *PROCESS* leitet jede Behavioral-VHDL-Beschreibung ein. Eine der wichtigsten Eigenschaften von verhaltensorientierten VHDL-Modellen ist die Sequentialität. Dies bedeutet, daß innerhalb eines Prozesses alle Programmschritte der Reihe nach, also Schritt für Schritt, abgearbeitet werden. Sind mehrere Prozesse aktiv, so werden diese parallel abgearbeitet.

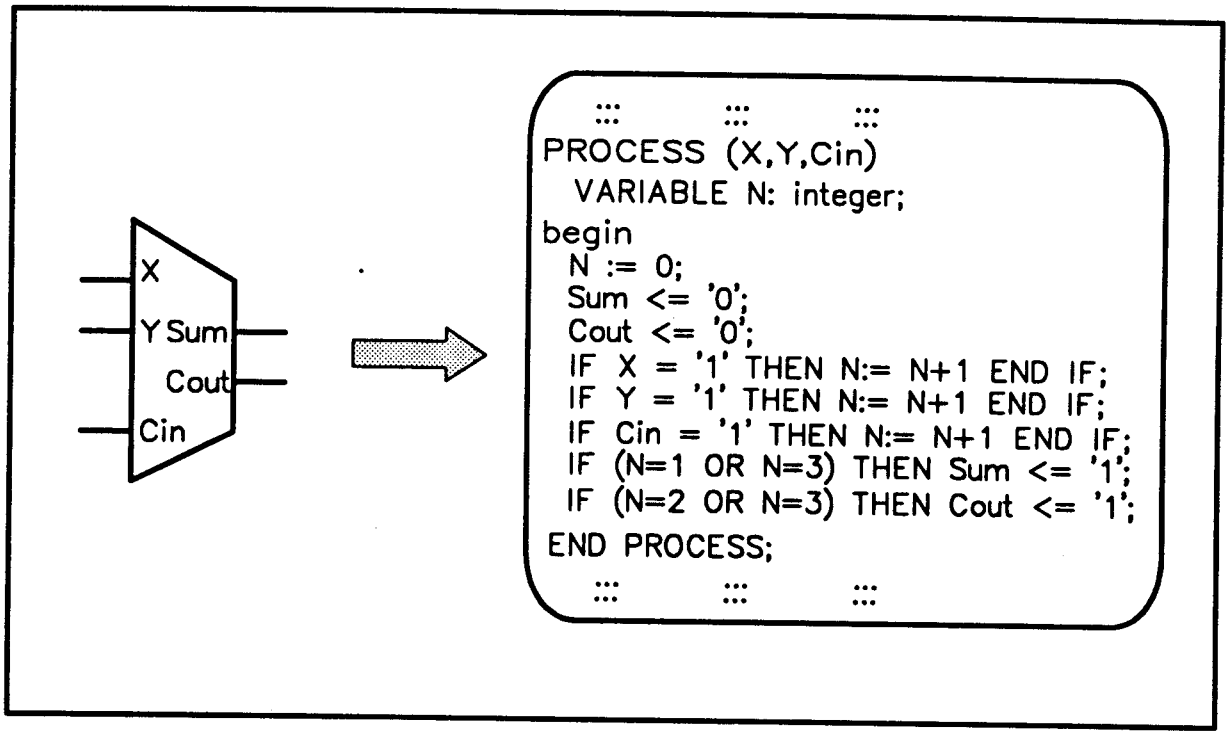


Bild 1: Der Volladdierer als Behavioral-Beschreibungs-Modell

Neben diesem Modellierungskonzept bietet VHDL noch die Möglichkeit, eine Schaltung als Datenfluß(Dataflow)- oder als Struktur(Structural)-Modell zu beschreiben. Eine datenfluß-orientierte Abbildung einer Schaltung wird oft auch als RTL_1-Modell bezeichnet, siehe Bild 2.

1 Register Transfer Level

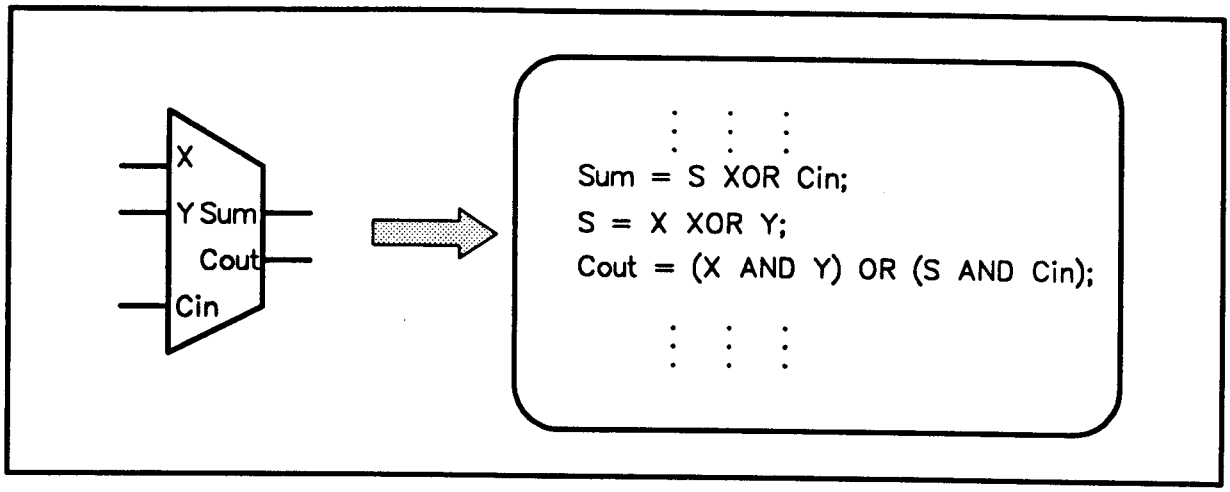


Bild 2: Der Volladdierer als Dataflow-Beschreibungs-Modell

Dataflow-VHDL erlaubt die Beschreibung einer Hardware durch die Beschreibung des Datenflusses mit Hilfe von Registerinhalten und deren Verknüpfung. In der Fähigkeit der abstrakten Modellbeschreibung liegt die datenflußorientierte Beschreibung zwischen dem Behavioral-VHDL und dem Structural-VHDL. Typische Vertreter sind zum Beispiel Beschreibungsmodelle für Zähler und allgemein für Systeme, für die bereits ein steuerungstechnisches Flußdiagramm[1] existiert.

Obwohl von einem in Dataflow-VHDL abgefaßten Entwurf schon viele Details bekannt und in die Beschreibung eingebracht sind, ist das Beschreibungsmodell dennoch so abstrakt, daß mit verhältnismäßig wenigen Zeilen im VHDL-Kode große Schaltungskomplexe darstellbar sind.

Die dritte Schaltungsbeschreibungsform von VHDL ist bekannt unter der Bezeichnung Structural-VHDL.

[1] auch als Statemachines bezeichnet

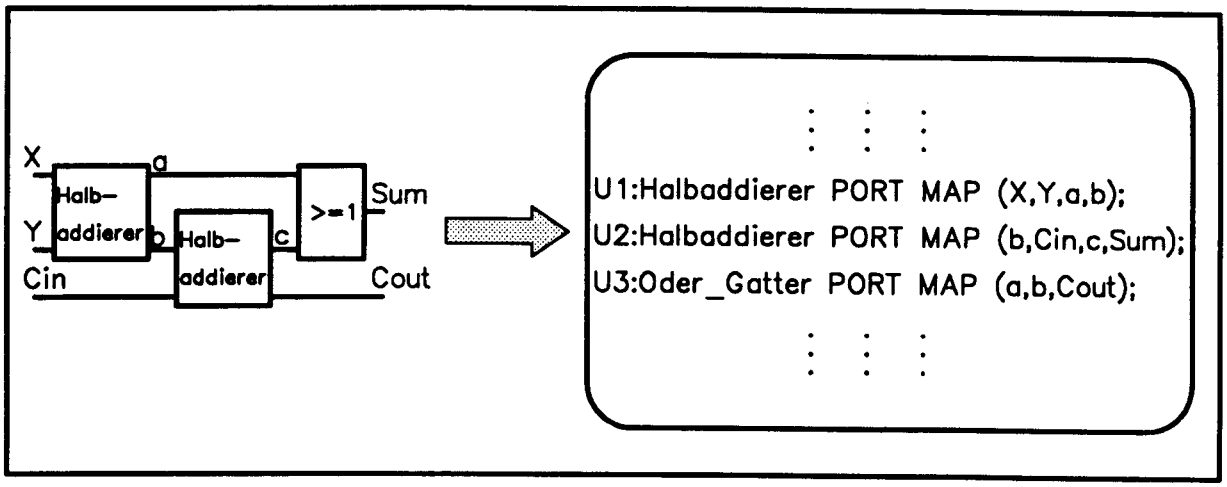


Bild 3: Der Volladdierer als Structural-Beschreibungs-Modell

Es ist eine hirarchisch organisierte Netzlistensprache (siehe Bild 3), die vergleichbar ist mit anderen gängigen Verbindungslisten (netlists) wie zum Beispiel die PSPICE Knotenliste. Ihr Einsatzgebiet innerhalb des Entwurfs umspannt zwei wesentliche Aufgabengebiete. Structural-VHDL dient als Bindeglied zwischen den unterschiedlichen Entwurfsblöcken. Hierbei ist es völlig gleichgültig, welche Beschreibungsform den jeweiligen Blöcken zugrundeliegt. Der Entwurf kann mit Structural-VHDL bis auf Gatterebene beschrieben werden. Structural-VHDL ist zudem eine leistungsfähige Verbindungslistendarstellung und kann somit das Bindeglied zum ASIC Hersteller sein.

Bild 4 zeigt nochmals bildlich den Anwendungsbereich der 3 Schaltungsbeschreibungsformen in VHDL beim Entwurf eines Systems. Weiterhin zeigt Bild 4 die Richtung, welche man bei der Top-Down-Designmethode geht.

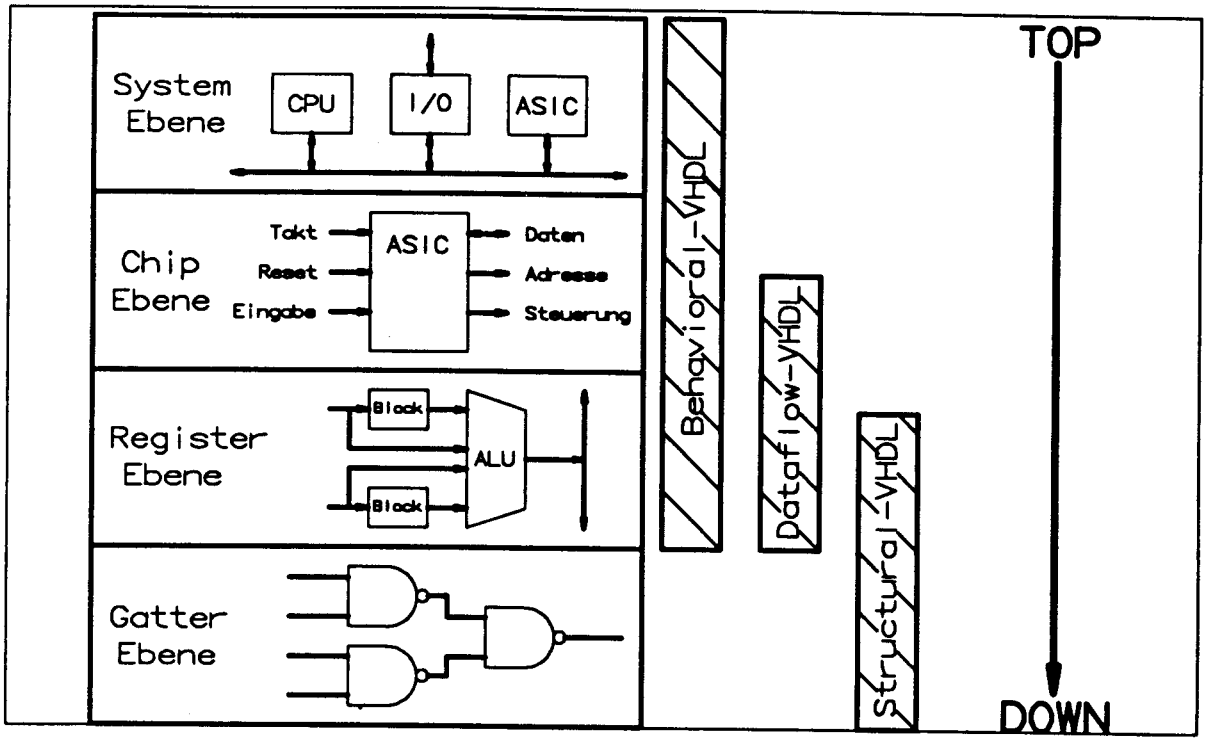


Bild 4: Schaltungsbeschreibungsformen in VHDL

Zur Gewährleistung einer klaren Struktur beim Entwurf nach der Top-Down-Design-Methode und zur Vermeidung von Sprachverwirrungen ist die Mischbarkeit der unterschiedlichen VHDL-Befehle durch wohldefinierte Gesetze geregelt. Es ist z.B. unzulässig, Befehle aus Dataflow-VHDL mit solchen aus Behavioral-VHDL zu mischen. Auch an anderer Stelle wurde VHDL für die Mixed-Level-Simulation vorbereitet. Im Laufe der Realisierung ist es immer wieder notwendig, einen Funktionsblock der einen Beschreibungsform durch einen entsprechenden Block einer anderen Beschreibungsform zu ersetzen. Das geschieht zum Beispiel immer dann, wenn zu einem Funktionsblock, welcher in Behavioral-VHDL beschrieben ist, ein detailliertere Architekturansatz gefunden wird. Mittels einer neuerlichen Mixed-Level-Simulation wird der Entwicklungsschritt sodann auf seine Übereinstimmung mit der Spezifikation getestet. In VHDL existieren hierfür spezielle Befehle, die eine Kontrolle der Beschreibungskonfiguration ermöglichen. Mit ihnen kann bestimmt werden, welche Beschreibungsform für den Funktionsblock zum Einsatz kommt.

1.2 Logiksynthese

Unter Logiksynthese versteht man die die automatische Umsetzung eines durch eine Hardwarebeschreibungssprache beschriebenen Entwurfs in eine Verbindungsliste. Die Umsetzung geschieht mit Hilfe eines Logiksyntheseprogramms. Das häufig benutzte Logiksyntheseprogramm LOG-IC der Firma ISDATA benutzt eine eigene Beschreibungssprache, welche jedoch im Gegensatz zu VHDL von Simulationsprogrammen zur Simulation der zeitabhängigen Verhaltens nicht benutzt werden kann. Künftige Logiksyntheseprogramme werden die Hardwarebeschreibungssprache VHDL, Behavioral- oder Dataflow-VHDL, als Eingabe verwenden. Zusammen mit den in VHDL beschriebenen Bauteilbibliotheken der ASIC Hersteller ist dann eine Logiksynthese, d.h die Generierung einer Verbindungsliste möglich, welche nach der Signallaufzeit bzw. nach der Anzahl der Gatter optimiert werden kann. Die generierte Verbindungsliste ist dann eine VHDL-Beschreibung des Entwurfs auf der Ebene von Structural-VHDL.
Das Logiksyntheseprogramm der Firma Mentor Graphics, AutoLogic, welches mit der Softwareversion 8.0 neben anderen Beschreibungsformen auch VHDL-Modelle unterstützt (siehe Bild 5), soll laut Mentor Graphics in der Lage sein, aus Entwürfen, welche in Behavioral-VHDL beschrieben sind, Structural-VHDL-Beschreibungen, d.h. Verbindungslisten, zu generieren. Falls die verwendete Bauteilbibliothek in VHDL beschrieben ist, bildet die Structural-VHDL-Beschreibung des Entwurfs das Bindeglied zum ASIC Hersteller. Für den Fall, daß die Bauteilbibliothek des ASIC Herstellers in der herkömmlichen Form vorliegt, kann mit Hilfe des Programms Schematic-Generator der Firma Mentor Graphics und der generierten Structural-VHDL-Beschreibung eine Verbindungsliste für das Programm Neted erzeugt werden.

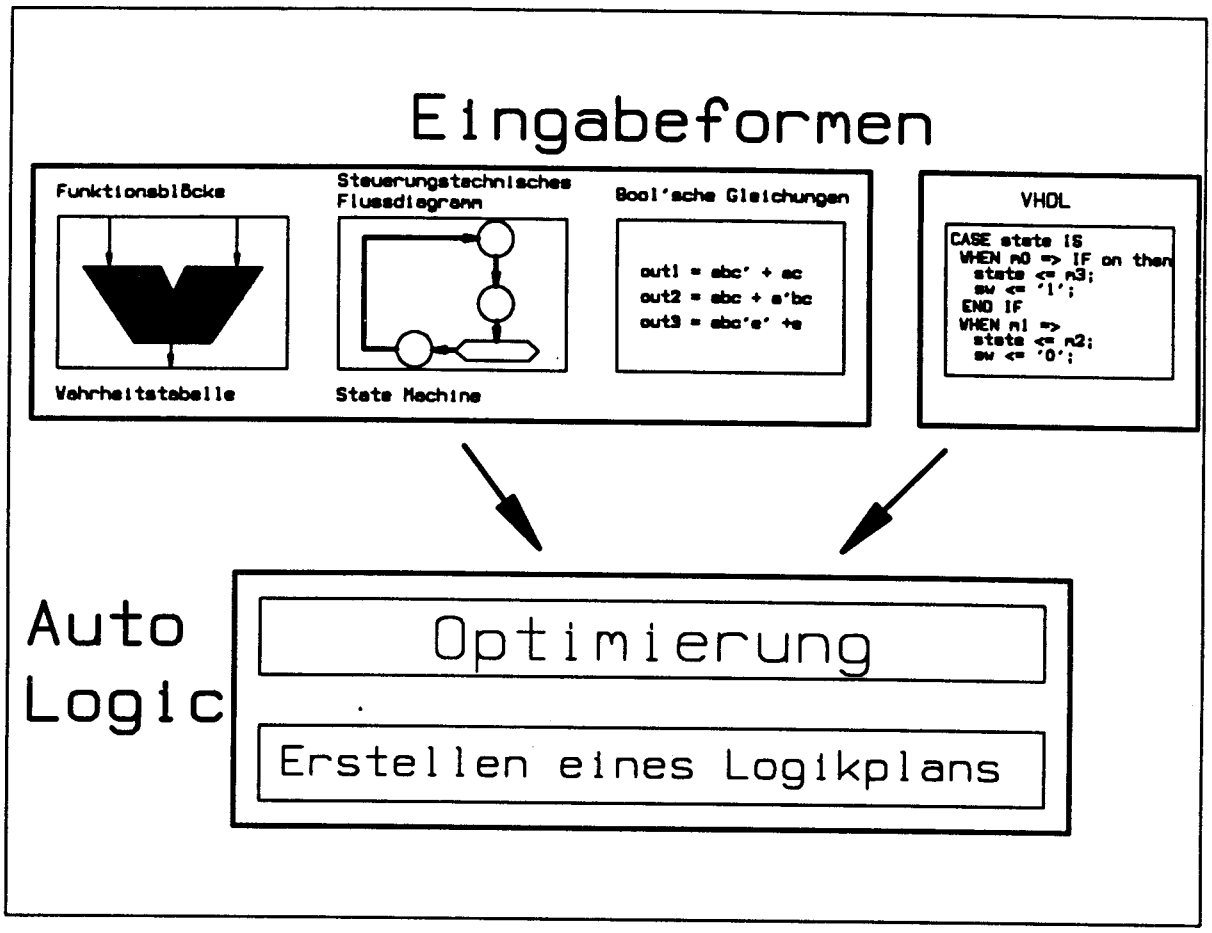


Bild 5: Beschreibungsformen in AutoLogic

2. Entwicklung eines VHDL-Modells

Im folgenden wird ein einfacher Entwurf mit VHDL beschrieben. Dabei werden die einzelnen Stationen des Ablaufplans, welcher in Bild 6 dargestellt ist, näher erläutert. Weiterhin wird der Entwurf in unterschiedlichen Ebenen entworfen.

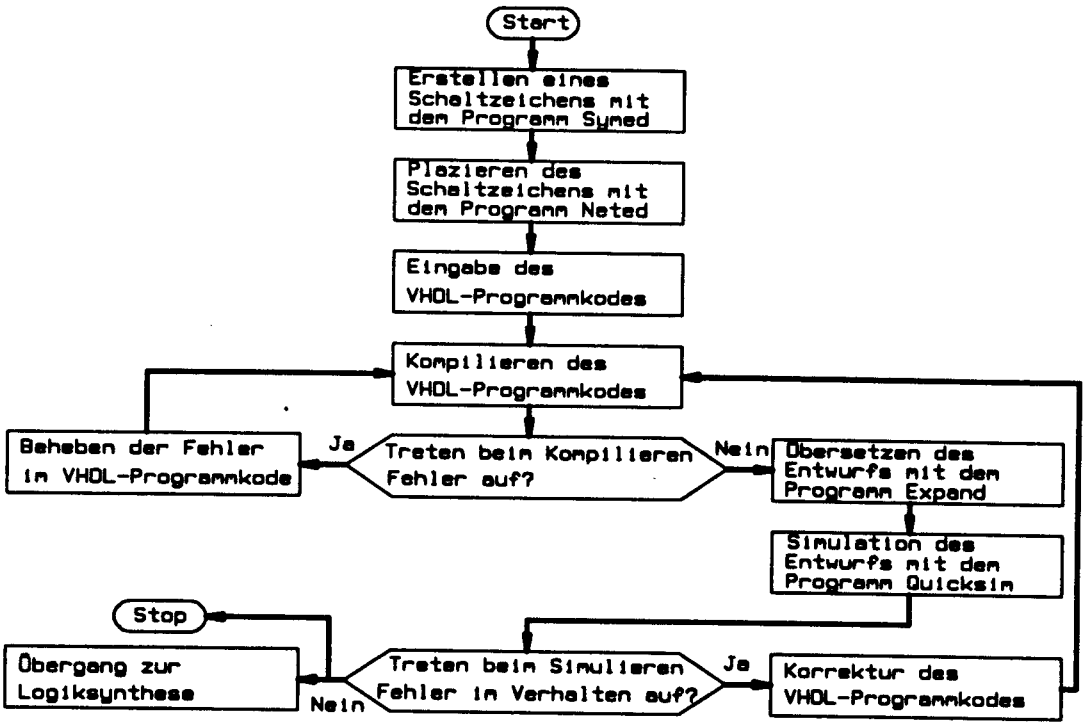


Bild 6: Ablaufplan zur Entwicklung eines VHDL-Modells

2.1 Funktionsweise des Entwurfs

Bei dem Entwurf handelt es sich um ein Schaltnetz, welches der Bool'schen Gleichung $E = \overline{A \cdot B + C \cdot D}$ bzw der Wahrheitstabelle in Bild 7 gehorcht.

D	C	B	A	E
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Bild 7: Wahrheitstabelle

Der Entwurf hat somit die 4 Eingangsvariablen A,B,C und D und die Ausgangsvariable E, welche kombinatorisch aus den Eingangsvariablen gebildet wird.

2.2 Erstellen des Schaltzeichens

Für jeden Entwurf, welcher in VHDL beschrieben werden soll, sollte mit Hilfe des Programms Symed der Firma Mentor Graphics ein Schaltzeichen erzeugt werden. Die Anwendung des Programms Symed findet man im Handbuch zum EDA-Seminar [15]. Der Name des Entwurfs sowie der Name des Schaltzeichens ist AOI, welches soviel wie And-Or-Inverter bedeutet. Der Aufruf des Programms Symed erfolgt von einer Shell-Ebene aus durch:

$ symed aoi <RETURN> .

Folgende Merkmale müssen dem Schaltzeichenkörper (body) zugeordnet werden:

- model

  Das Merkmal mit dem Namen model (model property) teilt dem Simulationsprogramm die Herkunft der Funktionsbeschreibung des Schaltzeichens mit. Für VHDL-Modelle wird hier $hdl angegeben, dadurch verwendet das Simulationsprogramm immer die zuletzt kompilierte Version der VHDL-Beschreibung des Schaltzeichens.

- comp

  Der Wert des Merkmals comp (comp property) entspricht dem Namen des Schaltzeichens und erlaubt dem Benutzer die Identifizierung der verschiedenen Schaltzeichen. Dieses Merkmal muß nicht hinzugefügt werden. In diesem Fall jedoch erhält der Schaltzeichenkörper das Merkmal comp mit dem Wert AOI.

Jedem Bauteilanschluß (pin) müssen folgende Merkmale zugeordnet werden:

- pin

  Der Wert des Merkmals mit dem Namen pin entspricht dem Anschlußnamen.
  In diesem Fall werden den Bauteilanschlüssen die Merkmale pin mit den Werten A bis E zugeordnet.

- pintype

  Das Wert des Merkmal pintype legt die Art des Bauteilanschlusses fest. Mögliche Werte des Merkmals mit dem Namen pintype sind:

  - in für Eingangsanschlüsse,
  - out für Ausgangsanschlüsse,
  - ixo für bidirektionale Anschlüsse.

  In diesem Fall wird den Bauteilanschlüssen A bis D das Merkmal pintype mit dem Wert in zugeordnet und dem Bauteilanschuß E das Merkmal pintype mit dem Wert out.

Bild 8 zeigt das entworfene Schaltzeichen im Programm Symed.

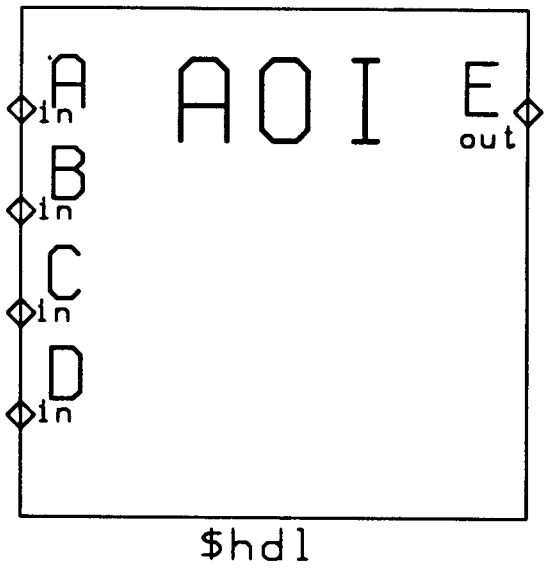


Bild 8: Schaltzeichen im Programm Symed

2.3 Plazieren des Schaltzeichens im Programm Neted

Das Schaltzeichen, welches den VHDL-Entwurf repräsentiert, wird anschliessend im Programm Neted plaziert. Die Anwendung des Programms Neted findet man im Handbuch zum EDA-Seminar [15]. Der Name des Stromlaufplans ist aoi_design.

Der Aufruf des Programms Neted erfolgt von einer Shell-Ebene aus durch:

$ neted aoi_design <RETURN> .

Um das zuvor im Programm Symed entworfene Schaltzeichen mit dem Namen aoi plazieren zu können, gibt man in der Kommandoeingabezeile folgendes ein:

NETED> activate component Verzeichnisname/aoi <RETURN> .

Signaleingänge (portin) und Signalausgänge (portout) in Stromlaufplänen müssen einen sog. Verbindungsnamen erhalten, damit die logischen Zustände an den Signalein- und ausgängen im Simulationsprogramm dargestellt und an den Signaleingängen verändert werden können. Die Verbindungsnamen sollten dieselben sein, wie die Anschlußnamen des Schaltzeichens. Bild 9 zeigt das plazierte Schaltzeichen mit den dazugehörigen Signalein- und ausgängen.

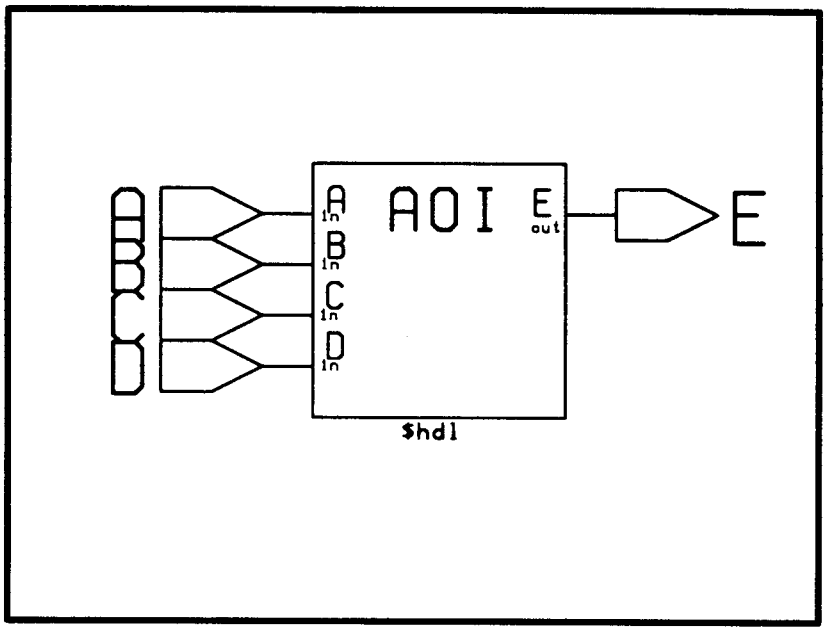


Bild 9: Plaziertes Schaltzeichen im Programm Neted

2.4 Eingabe des VHDL-Programmkodes

2.4.1 Verzeichnisstruktur

Beim Erstellen des Schaltzeichens mit dem Namen AOI, siehe Kapitel 2.2, sowie beim Erstellen des Stromlaufplans mit dem Namen aoi_design, wird sowohl von dem Programm Symed als auch von dem Programm Neted ein Verzeichnis angelegt. Bild 10 zeigt die für diesen Entwurf relevante Verzeichnisstruktur mit den in den Verzeichnissen abgelegten Dateien und Zeigern. Die Erläuterung der in Bild 10 verwendeten Symbole zeigt Bild 11.

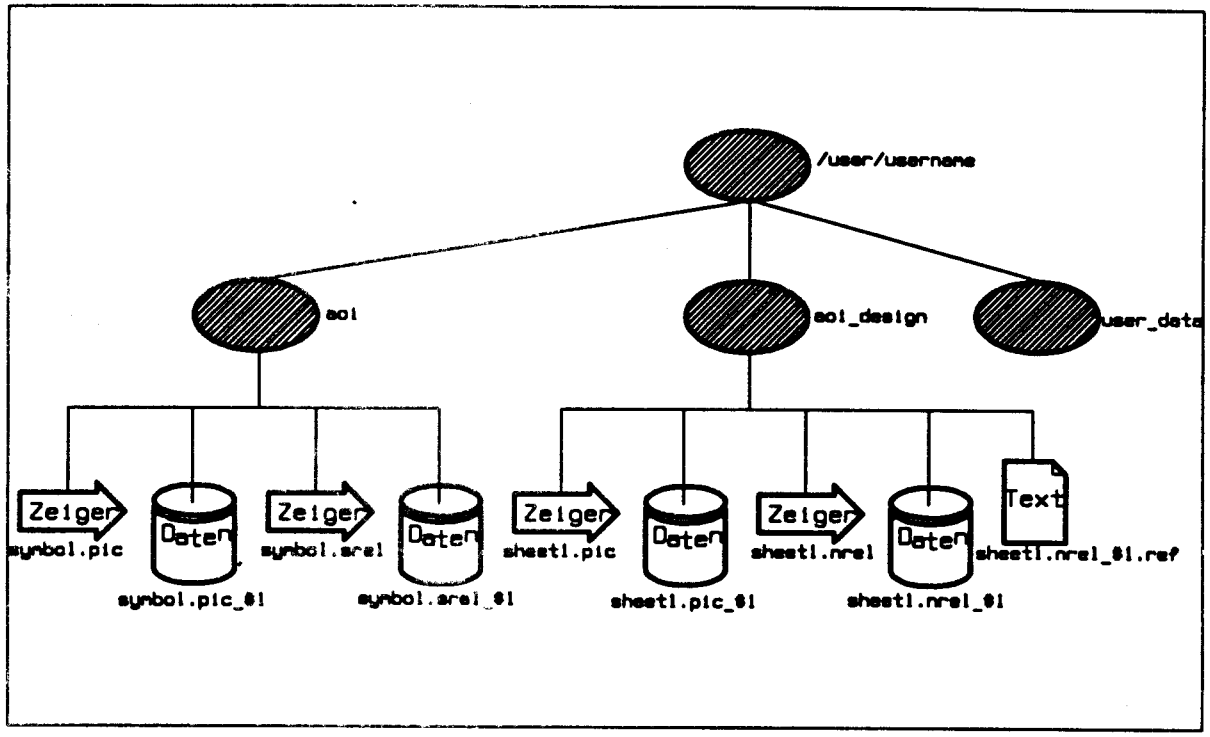


Bild 10: Momentane Verzeichnisstruktur

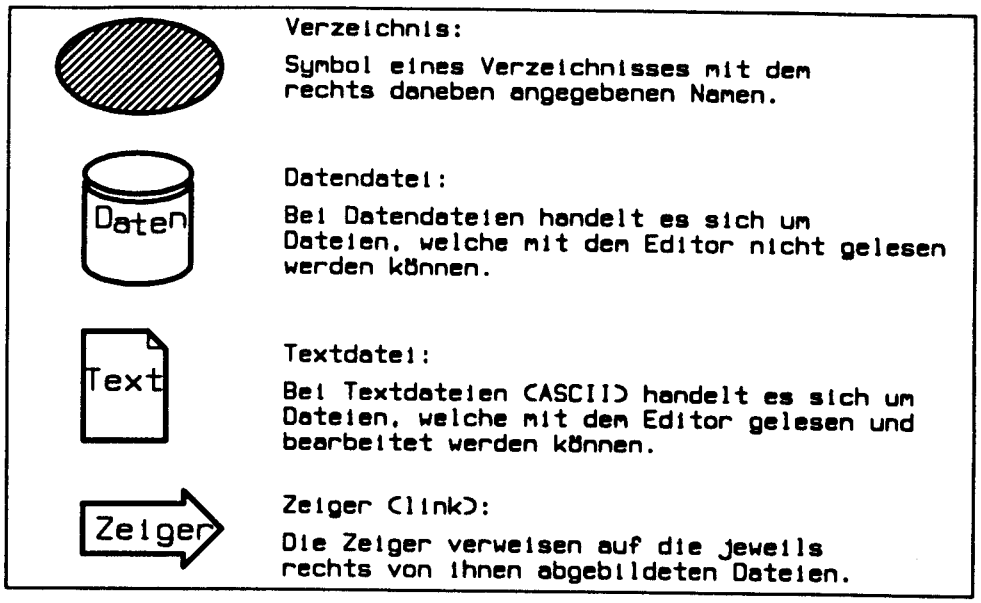


Bild 11: Erläuterung der verwendeten Symbole

2.4.2 Anlegen der Textdatei für der VHDL-Programmkode

Da das Verhalten des in Kapitel 2.2 entworfenen Schaltzeichens durch ein VHDL-Programm beschrieben werden soll, Bild 12 verdeutlicht dies, muß die Textdatei für den VHDL-Programmkode in dem Verzeichnis des Schaltzeichens, also im Verzeichnis aoi angelegt werden.

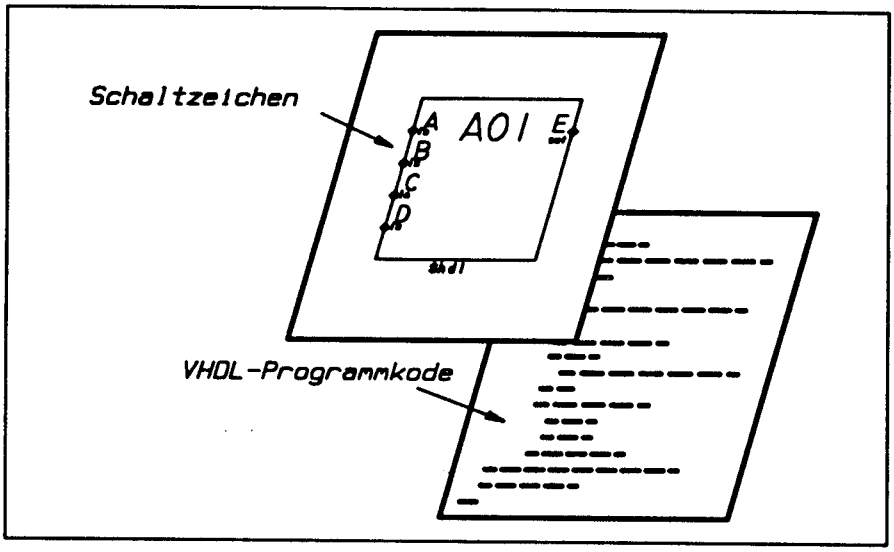


Bild 12: Das Verhalten des Schaltzeichens wird durch das Programm beschrieben

Um die Textdatei für den VHDL-Programmkode anzulegen, wechselt man in das vom Programm Symed bei der Erzeugung des Schaltzeichens angelegte Verzeichnis mit dem Namen aoi. Durch

<SAVE/EDIT>

innerhalb einer Shell-Ebene wird man aufgefordert, einen Namen für die zu erzeugende Textdatei einzugeben.

Für den Namen der Textdatei, welche den VHDL-Programmkode beinhalten soll, sollte der Name des Schaltzeichens mit dem Suffix .hdl angegeben werden. In diesem Fall erhält die Textdatei den Namen aoi.hdl.

Das Verzeichniss aoi enthält nun die in Bild 13 dargestellten Dateien und Zeiger. Die Textdatei aoi.hdl, welche später den VHDL-Programmkode enthalten soll, ist zu diesm Zeitpunkt eine leere Datei.

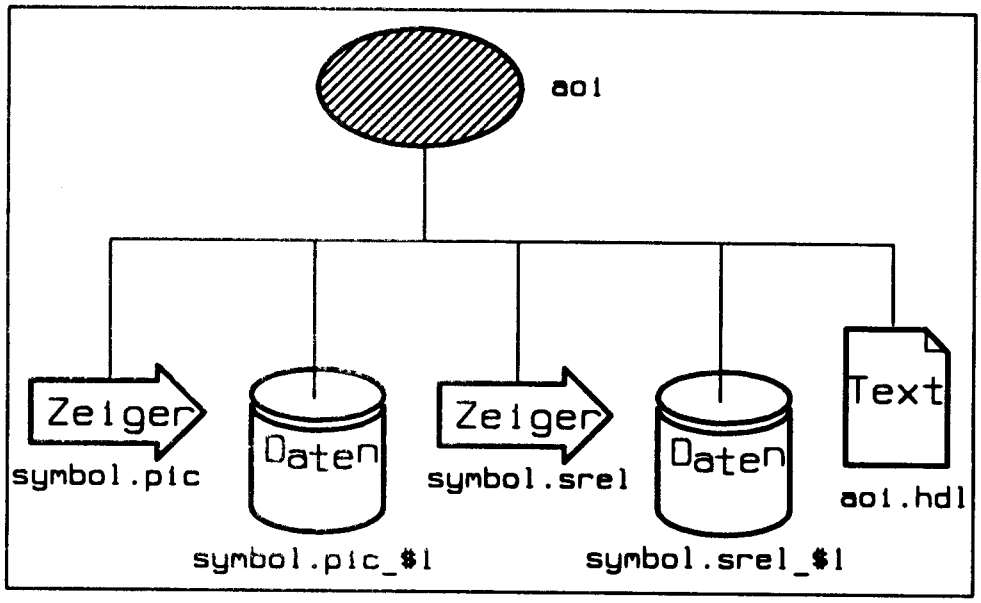


Bild 13: Inhalt des Verzeichnisses aoi

2.4.3 VHDL-Programmkode des Entwurfs

Der VHDL-Programmkode des Entwurfs AOI ist als Programmkode 1 nachstehend aufgeführt. Die in Klammern stehenden Zeilennummern am Anfang jeder Zeile dienen hier lediglich der Erläuterung der Programmschritte, sie dürfen bei der Eingabe des Programmkodes in die Textdatei aoi.hdl nicht angegeben werden. Weiterhin werden in diesem Beispiel als auch im folgenden sämtliche VHDL-spezifischen Anweisungen in Großbuchstaben geschrieben.
Der Programmkode 1 zeigt den Programmkode des Entwurfs AOI.

Programmkode 1: Programmkode des Entwurfs AOI

```
(1) USE std.mentor_base.ALL;                    -- Bibliotheksaufruf
(2)
(3) ENTITY aoi IS                                -- Beschreibung der Schnittstelle
(4)  PORT (A, B, C, D : IN  qsim_state;          -- zum Schaltzeichen
(5)                E :.OUT qsim_state);
(6) END aoi;
(7)
(8) ARCHITECTURE behav1 OF aoi IS                -- Beginn der Verhaltensbeschreibung
(9)
(10)  BEGIN
(11)
(12)   p1: PROCESS (A, B, C, D)                 --Prozess p1
(13)    BEGIN
(14)     E <= NOT ((A AND B) OR (C AND D) );
(15)   END PROCESS p1;                           --
(16)
(17) END behav1;
```

2.4.4 Basiszeichensatz

Der Basiszeichensatz von VHDL besteht aus Großbuchstaben, Kleinbuchstaben, Ziffern, den speziellen Zeichen

" # & ' ( ) * + , - . / : ; < = > _ ! $ % ? §

und dem Leerzeichen. Die Kleinbuchstaben sind gleichbedeutend mit den entsprechenden Großbuchstaben, mit Ausnahme von Zeichenketten und Zeichenliteralen. Im Programmkode 1 sowie im folgenden werden aufgrund der Übersichtlichkeit alle VHDL-Anweisungen in Großbuchsta-

ben und Bezeichner in Kleinbuchstaben geschrieben. Ausnahmen gibt es lediglich bei den Bezeichnern, welche den Anschlußnamen der Schaltzeichen entsprechen, und den Kommentaren. Bezeichner beginnen mit einem Buchstaben. Es können dann weitere Buchstaben, Ziffern und einzelne Unterstrichzeichen folgen. Bezeichner, die sich an der entsprechenden Position lediglich durch Groß- und Kleinbuchstaben unterscheiden, sind gleichbedeutend. Folgende Bezeichner sind dementsprechend gleichbedeutend:

aoi, AOI und Aoi .

Es gibt zwei Arten von numerischen Ausdrücken - Ganz- und Gleitpunktzahlen. Beispiele:

12 12.0 1.2E1 (oder 1.2e1) .

Bei Literalen können andere als dezimale Grundzahlen verwendet werden. Zum Beispiel kann 61.0 dezimal mit der Grundzahl 8 folgendermaßen geschrieben werden:

8#75.0# oder 8#7.5#E1.

Dabei ist die 1 im zweiten Beispiel der Exponent. Beide, die Grundzahl und der Exponent, sind immer in dezimaler Schreibweise dargestellt. Für Grundzahlen, die größer als 10 und kleiner als 16 sind, werden die Buchstaben A bis F in der Mantisse benutzt. Beispiel mit der Grundzahl 16:

16#FE85# .

Zwischen zusammengehörigen Ziffern können einzelne Unterstriche eingefügt werden, um sie leichter lesbar zu machen, wie z.B.:

12_000_000 oder 2#1010_1110_0011_0101# .

Ein Zeichenliteral wird von einfachen Anführungszeichen eingeschlossen. Zum Beispiel werden die Literale A, * und Leerzeichen folgendermaßen geschrieben:

'A' '*' ' ' .

Zeichenketten werden als eine Serie von null oder mehr druckbaren Zeichen innerhalb von doppelten Anführungszeichen geschrieben. Das doppelte Anführungszeichen " muß zweimal

geschrieben werden, um eine Zeichenkette einzuschließen. Verkettungen, dargestellt durch &, werden benutzt, um Zeichenketten darzustellen, die länger als eine Zeile sind oder um Zeichenketten zu verbinden. Beispiele für Zeichenketten sind:

" ", "", "A" "Eingangsvariable undefiniert",
"Eingangsvariable " & "A" & " undefiniert." .

Kommentare beginnen mit zwei unmittelbar aufeinanderfolgenden Gedankenstrichen an beliebiger Stelle in der Zeile und hören am Zeilenende auf:

```
(3) ENTITY aoi IS                              -- Beschreibung der Schnittstelle
(4)  PORT (A, B, C, D : IN  qsim_state;        -- zum Schaltzeichen
(5)                 E : OUT qsim_state);
(6) END aoi;
```

Alle VHDL-Anweisungen müssen mit einem Semikolon abgeschlossen werden. Beispiel:

```
(1) USE std.mentor_base.ALL;                -- Bibliotheksaufruf (12)   p1:
```

Zwischen Bezeichnern und VHDL-Anweisungen müssen mindestens ein oder mehrere Leerzeichen eingefügt werden. Bsp:

```
(3) ENTITY aoi IS
```

Am Zeilenanfang können beliebig viele Leerzeichen eingefügt werden. Das Einrücken von Programmblöcken dient der besseren Lesbarkeit des Programmkodes, siehe Programmkode 1.

2.4.5 Aufbau des VHDL-Programmkodes

Ein VHDL-Programmkode ist aufgeteilt in drei Hauptblöcke:

- PACKAGE-Block

  Im Package-Block wird festlegt, welche Unterprogrammbibliotheken im nachfolgenden Programm mitbenutzt werden sollen. In Unterprogrammbibliotheken sind Prozeduren, Funktion und Typendeklarierungen festgelegt. Im Programmkode 1 besteht der

Package-Block lediglich aus einer Zeile:

```
(1) USE std.mentor_base.ALL;                    -- Bibliotheksaufruf .
```

In dieser Zeile wird durch die VHDL-Anweisung *USE* festgelegt, daß die Unterprogrammbibliothek std.mentor_base vollständig (*.ALL*) mitbenutzt werden kann. In dieser Unterprogrammbibliothek sind Standardfunktionen festgelegt, welche der Kommunikation zwischen dem VHDL-Programm und dem Mentor-Simulator (Quicksim) dienen. Eine detaillierte Beschreibung dieser Unterprogrammbibliothek folgt später. Um eine VHDL-beschriebenes System mit Quicksim simulieren zu können, ist es notwendig, diese Programmzeile am Kopf des Programmkodes einzufügen. Weiteres zur Erstellung und Verwendung von Unterprogrammbibliotheken folgt später.

- ENTITY-Block

  Der Entity-Block bildet die Schnittstelle zwischen dem Programm und dem Schaltzeichen bzw. dem Simulator. Der unten aufgeführte Programmkode zeigt den Entity-Block des Programmkodes 1.

```
(3) ENTITY aoi IS                              -- Beschreibung der Schnittstelle
(4)  PORT (A, B, C, D : IN  qsim_state;        -- zum Schaltzeichen
(5)                E : OUT qsim_state);
(6) END aoi;
```

  Der Entity-Block ist unabhängig von der Hardwarebeschreibungsform, siehe Kapitel 1.1. Beginnend mit der VHDL-Anweisung *ENTITY* und dem danach folgenden Bezeichner (aoi), der immer der Name des Schaltzeichens ist, läßt sich der Entity-Block vollständig aus dem in Kapitel 2.2 entworfenen Schaltzeichen ableiten. Weiteres zum Entity-Block folgt in Kapitel 2.4.6 .

- ARCHITECTURE-Block

  Der Architecture-Block enthält die Verhaltensbeschreibung des Entwurfs. Er ist abhängig von der Hardwarebeschreibungsform aufgebaut, siehe Kapitel 1.1. Der Architecture-Block beginnt jedoch immer mit der VHDL-Anweisung *ARCHITECTURE* und ei-

nem Bezeichner. Abgeschlossen wird der Architecture-Block, genau wie der Entity-Block, mit der Anweisung *END* und dem Bezeichner. Der unten aufgeführte Programmkode zeigt den Architecture-Block des Programmkodes 1.

```
(8) ARCHITECTURE behav1 OF aoi IS     -- Beginn der Verhaltensbeschreibung
(9)
(10)  BEGIN
(11)
(12)   p1: PROCESS (A, B, C, D)              --Prozess p1
(13)    BEGIN
(14)     E <= NOT ((A AND B) OR (C AND D) );
(15)   END PROCESS p1;                        --
(16)
(17) END behav1;
```

Weiteres zum Architecture-Block des Programmkodes 1 zeigt Kapitel 2.4.7

2.4.6 Der Entity-Block

Der Programmkode 2 zeigt den Entity-Block des Entwurfs AOI.

Programmkode 2: Entity-Block des Entwurfs AOI

```
(3) ENTITY aoi IS                          -- Beschreibung der Schnittstelle
(4)  PORT (A, B, C, D : IN  qsim_state;     -- zum Schaltzeichen
(5)                E : OUT qsim_state);
(6) END aoi;
```

Der Entity-Block eines VHDL-Programmkodes beginnt mit der Anweisung *ENTITY* gefolgt von dem Bezeichner des Entity-Blocks, als Entity-Name bezeichnet, und der Anweisung *IS*. Abgeschlossen wird der Entity-Block durch die Anweisung *END*, dem Entity-Namen und einem Semikolon. Der Bezeichner des Entity-Blocks, also der Entity-Name, ist immer der Name des Schaltzeichens, siehe Kapitel 2.2, bzw der Name des Verzeichnisses, in welchem sich der VHDL-Programmkode befindet, siehe Bild 13. Die Zeilen 4 und 5 des Entity-Blocks, siehe Programmkode 2, bilden den sog. Entity-Körper. Der Entity-Körper legt die Schnittstelle zwischen dem Programmkode und dem Schaltzeichen des Entwurfs fest. Die VHDL-Anweisung PORT leitet die Schnittstellenfestlegung ein. Innerhalb von runden Klammern müssen

dann sämtliche Bezeichner, welche den Anschlußnamen der Schaltzeichen entsprechen und im folgenden als Anschlußsignale bezeichnet werden, zusammen mit der Übertragungsrichtung einem Datentyp (siehe Kapitel 3.1) zugeordnet werden. Diese Zuordnung bezeichnet man als Signaldeklarierung, siehe Kapitel 3.2.3.

Die Übertragungsrichtung eines Anschlußsignals entspricht der Art des Bauteilanschlusses, welches dem Anschluß des Schaltzeichens durch das Merkmal pintype, siehe Kapitel 2.2, zugeordnet wurde.
Gültige Übertragungsrichtungen sind:

- IN
  Der Wert des Anschlußsignals kann nur gelesen werden.

- OUT
  Der Wert des Anschlußsignals kann verändert aber nicht gelesen werden.

- INOUT
  Der Wert des Anschlußsignals kann gelesen und verändert werden.

Damit das Verhalten des Schaltzeichens mit Quicksim simuliert werden kann, werden sämtliche Anschlußsignale dem Datentyp qsim_state zugeordnet. Damit können die Anschlußsignale folgende Werte annehmen:

- '0' logisch 0 oder O,

- '1' logisch 1 oder L,

- 'X' unbestimmt,

- 'Z' hochohmig (high impedance).

Der Datentyp qsim_state ist in der Unterprogrammbibliothek std.mentor_base, welche im Package-Block (Zeile 1 des Programmkodes 1) dem Programmkode zur Anwendung freigegeben wurde, festgelegt. Weiteres zur Typendeklarierung siehe Kapitel 3.1 .

Falls mehrere Anschlußsignale dieselbe Übertragungsrichtung besitzen, so können sie, durch ein Komma getrennt, zu einer Bezeichnerliste zusammengefasst werden. So geschehen in Zeile 4 im Programmkode 2. Falls man bei der Programmeingabe auf eine solche Bezeichnerliste verzichtet, würde der Entity-Körper folgendermaßen aussehen:

```
PORT  (A  : IN  qsim_state;
       B  : IN  qsim_state;
       C  : IN  qsim_state;
       D  : IN  qsim_state;
       E  : OUT qsim_state);
```

2.4.7 Der Architecture-Block

Der Programmkode 3 zeigt den Architecture-Block des Entwurfs AOI.

Programmkode 3: Architecture-Block des Programmkodes AOI

```
(8) ARCHITECTURE behav1 OF aoi IS          -- Beginn der Verhaltensbeschreibung
(9)
(10)  BEGIN
(11)
(12)   p1: PROCESS (A, B, C, D)              --Prozess p1
(13)    BEGIN
(14)     E <= NOT ((A AND B) OR (C AND D) );
(15)   END PROCESS p1;                        --
(16)
(17) END behav1;
```

Der Architecture-Block eines VHDL-Programms beginnt mit der Anweisung *ARCHITECTURE* gefolgt von dem Bezeichner des Architecture-Blocks, als Architecture-Name bezeichnet, der Anweisung *OF*, dem Entity-Name (siehe Kapitel 2.4.6) und der Anweisung *IS*. Abgeschlossen wird der Architecture-Block durch die Anweisung *END*, dem Architecture-Name und einem Semikolon. Der Bezeichner des Architecture-Blocks, also der Architecture-Name, ist im Rahmen des Basiszeichensatzes, siehe Kapitel 2.4.4, frei

wählbar.

Der Architecture-Block ist folgendermaßen aufgebaut:

```
ARCHITECTURE Architecture-Name OF Entity-Name IS
   (Architecture-Kopf)
 BEGIN
   (Architecture-Körper)
END Architecture-Name;
```

Die Zeile 9 des Architecture-Blocks, siehe Programmkode 3, bildet den sog. Architecture-Kopf, die Zeilen 11 bis 16 den sog. Architecture-Körper.

Innerhalb des Architecture-Kopfes können Typen, Konstanten und Signale nicht aber Variablen zur Verwendung im Architecture-Körper deklariert (vereinbart) werden. Grundlegendes zur Typendeklarierung findet man in Kapitel 3.1, zur Konstantendeklarierung in Kapitel 3.2.1 und zur Signaldeklarierung in Kapitel 3.2.3.

Innerhalb dieses Beispiels werden keine Typen, Konstanten und Signale im Architecture-Körper benötigt, es ist deshalb nicht notwendig, solche innerhalb des Architecture-Kopfes zu deklarieren.

Getrennt vom Architecture-Kopf durch die Anweisung *BEGIN* in Zeile 10 des Programmkodes 3 beginnt bei Zeile 11 der Architecture-Körper, in welchem das Verhalten des Entwurfs beschrieben wird. Der Architecture-Körper ist der wichtigste Teil des VHDL-Programmkodes. Nur im Architectur-Körper kann erkannt werden, nach welcher Hardwarebeschreibungsform, siehe Kapitel 1.1, der Programmkode aufgebaut wurde. Die VHDL-Anweisung PROCESS (Zeile 12) leitet jede Behavioral-VHDL-Beschreibung ein, demnach handelt es sich bei diesem Beispiel um ein verhaltensorientiertes VHDL-Modell (siehe Kapitel 1.1).

2.4.7.1 Die VHDL-Anweisung PROCESS

Der Programmkode 4 zeigt den Process-Block des Entwurfs AOI.

Programmkode 4: Process-Block des Programmkodes AOI

```
(12)   p1: PROCESS (A, B, C, D)              --Prozess p1 mit Sensitivit
(13)    BEGIN
(14)     E <= NOT ((A AND B) OR (C AND D) );
(15)   END PROCESS p1;                       --
```

Die VHDL-Anweisung *PROCESS* legt einen Prozeß innerhalb einer verhaltensorientierten VHDL-Beschreibung (Behavioral-VHDL) fest. Eine der wichtigsten Eigenschaften von Behavioral-VHDL-Modellen ist die Sequentialität. Dies bedeutet, daß innerhalb eines Prozesses alle Programmschritte der Reihe nach, also Schritt für Schritt abgearbeitet werden. Sind mehrere Prozesse aktiv, so werden diese parallel abgearbeitet.

Ein Process-Block einer Behavioral-VHDL-Beschreibung beginnt mit einem Bezeichner, dem sog. Process-Namen, gefolgt von einem Doppelpunkt. Sowohl der Process-Name als auch der Doppelpunkt können weggelassen werden (vergleiche Bild 1). Die Angabe eines Process-Namen ist jedoch bei der Fehlersuche hilfreich. Gefolgt von dem Process-Namen und dem Doppelpunkt folgen die VHDL-Anweisung PROCESS und innerhalb von runden Klammern in einer Bezeichnerliste die Signalnamen, die den Prozeß aufrufen können, siehe Zeile 12. Die Bezeichnerliste, die die Signalnamen beinhaltet, welche den Prozeß aufrufen können, wird als Sensitivity-Liste bezeichnet.

Ein Prozeß wird aufgerufen, wenn sich der Wert eines oder mehrere Signale der Sensitivity-Liste ändert. In der Sensitivity-Liste können nur Signalnamen, siehe Kapitel 3.2.3 Signaldeklarierung, welche im Architecture-Kopf deklariert wurden oder Anschlußsignale mit der Übertragungsrichtung IN oder INOUT, siehe Kapitel 2.4.6 aufgeführt werden.
Abgeschlossen wird der Process-Block durch die Anweisung *END PROCESS*, den Process-Namen, falls dieser angegeben wurde, und ein Semikolon, siehe Zeile 15. Der Bezeichner des Process-Blocks, also der Process-Name, ist im Rahmen des Basiszeichensatzes, siehe Kapitel 2.4.4, frei wählbar.

Der Process-Block ist folgendermaßen aufgebaut:

```
Process-Name: PROCESS (Sensitivity-Liste)
   (Process-Kopf)
 BEGIN
   (Process-Körper)
END PROCESS Process-Name;
```

Zwischen den Zeilen 12 und 13 des Process-Blocks, siehe Programmkode 4, könnte der sog. Process-Kopf plaziert werden, die Zeile 14 bildet den sog. Process-Körper.

Innerhalb des Process-Kopfes können Typen, Konstanten und Variablen, nicht aber Signale zur Verwendung im Process-Körper, deklariert (vereinbart) werden. Grundlegendes zur Typendeklarierung findet man in Kapitel 3.1, zur Konstantendeklarierung in Kapitel 3.2.1 und zur Variablendeklarierung in Kapitel 3.2.2.

Innerhalb dieses Beispiels werden keine Typen, Konstanten und Variablen im Process-Körper benötigt, es ist deshalb nicht notwendig, solche innerhalb des Process-Kopfes zu deklarieren.

Die im Process-Kopf deklarierten Typen, Variablen und Konstanten sind nur innerhalb dieses Prozesses gültig.

Getrennt vom Process-Kopf durch die VHDL-Anweisung *BEGIN* in Zeile 13 des Programmkodes 4 beginnt bei Zeile 14 der Process-Körper, in welchem festgelegt wird, was geschieht, wenn der Prozeß aufgerufen wird. Falls der Prozeß aufgerufen wird, werden alle VHDL-Anweisungen innerhalb des Process-Körpers der Reihe nach, also Schritt für Schritt abgearbeitet. Anschließend wird der Prozeß beendet. In diesem Beispiel besteht der Process-Körper nur aus einer Zeile:

```
(14)   E <= NOT ((A AND B) OR (C AND D) );
```

In dieser Zeile wird dem Anschlußsignal mit dem Signalnamen E der durch die logische Verknüpfung, der Werte der Anschlußsignale mit den Signalnamen A bis D gebildete Wert zugeordnet. Mehr zur Signalzuweisung mit Hilfe der Zuweisungsanweisung <= findet man in Kapitel 3.3.2. Die logische Verknüpfung erfolgt mit Hilfe logischer Operatoren.
Diese Zeile entspricht also der Bool'schen Gleichung $E = \overline{A \cdot B + C \cdot D}$.

Der in Zeile 14 rechts der Zuordnungsanweisung stehende Teil der Zeile bezeichnet man als logischen Ausdruck. Ein logischer Ausdruck setzt sich aus logischen Operatoren und Operanden zusammen, wobei die Operanden in diesem Fall die Anschlußsignale sind. Bei der Berechnung eines logischen Ausdrucks werden die logischen Operatoren von links nach rechts abgearbeitet. Durch das Setzen von Klammern kann die Reihenfolge der Berechnung geändert werden.
Gültige logische Operatoren sind:

- *NOT*

  Der logische Operator *NOT* bewirkt eine Negation.

- *AND*

  Der logische Operator *AND* bewirkt eine UND-Verknüpfung.

- *OR*

  Der logische Operator *OR* bewirkt eine ODER-Verknüpfung.

- *NAND*

  Der logische Operator *NAND* bewirkt eine negierten UND-Verknüpfung.

- *NOR*

  Der logische Operator *NOR* bewirkt eine negierten ODER-Verknüpfung.

- *XOR*

  Der logische Operator *XOR* bewirkt eine Exklusiv-ODER-Verknüpfung.

2.5 Kompilieren des VHDL-Programmkodes

Des VHDL-Programmkode (Programmkode 1), welcher das Verhalten des Entwurfs beschreibt, muß mit dem VHDL-Kompiler kompiliert werden, damit das zeitabhängige Verhalten des Entwurfs mit dem Simulationsprogramm Quicksim simuliert werden kann.

Der Aufruf des VHDL-Kompilers erfolgt aus einer Shellebene durch:

$ hdl Verzeichnisname Dateiname <RETURN> .

Der Verzeichnisname ist der Name des Verzeichnisses, in welchem sich die Datei mit dem VHDL-Programmkode befindet (siehe Kapitel 2.4.2). Da man sich nach der Eingabe des Programmkodes bereits in diesem Verzeichnis befindet, kann man statt dem Verzeichnisnamen einen Punkt (.) setzen, welcher dem aktuellen Verzeichnis entspricht.

Der Dateiname entspricht dem Namen der Textdatei, welche den VHDL-Programmkode beinhaltet, in diesem Fall aoi.hdl (siehe Kapitel 2.4.2).

Der VHDL-Kompiler wird für diesen Fall, d.h. für die Beschreibung des Entwurfs AOI, folgendermaßen aufgerufen:

$ hdl . aoi.hdl <RETURN> .

Falls die Kompilierung erfolgreich war, angezeigt durch die Ausgabe 0 errors 0 warnings, erscheint nach dem Aufruf folgender Text am Bildschirm:

```
System-1076 compiler Version 7.0_0.30

Copyright (c) Mentor Graphics Corporation, 1989
                UNPUBLISHED, LICENSED SOFTWARE.
    CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
  PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.
```

```
0 errors.
0 warnings.
```

Falls die Kompilierung nicht erfolgreich war, erscheint nach dem Aufruf eine Fehlermeldung in der unten gezeigten Form am Bildschirm.

```
System-1076 compiler Version 7.0_0.30

Copyright (c) Mentor Graphics Corporation, 1989
                UNPUBLISHED, LICENSED SOFTWARE.
    CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
  PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.


[Line:   17] END behav1
----------------------^
Expecting one of  ';'.
# [Error 289] Semi_colon missing.

1 errors.
0 warnings.
```

In dem VHDL-Programmkode wurde das Semikolon am Ende der Zeile 17, siehe Programmkode 1 weggelassen. Der Kompiler zeigt daraufhin einen Fehler am Ende der Zeile 17 an, er schlägt weiterhin eine Verbesserung vor (`Expecting one of  ';'.`) und druckt zusätzlich eine Fehlernummer zusammen mit einer Kurzbeschreibung des Fehlers aus (`[Error 289] Semi_colon missing.`). Eine genauere Beschreibung des Fehlers findet man im System-1076 Error Message Manual [3], welches nach Fehlernummern geordnet ist.

Ist der Fehler im Programmkode behoben, so kann der Programmkode erneut kompiliert werden.

2.5.1 Verzeichnisstruktur nach dem Kompilieren

Nachdem der Programmkode erfolgreich, d.h. fehlerfrei, kompiliert wurde, enthält das Verzeichnis, welches die Daten des Entwurfs beinhaltet, die in Bild 14 gezeigten Einträge (vergleiche Bild 13). Die Erläuterung der in Bild 14 verwendeten Symbole zeigt Bild 11.

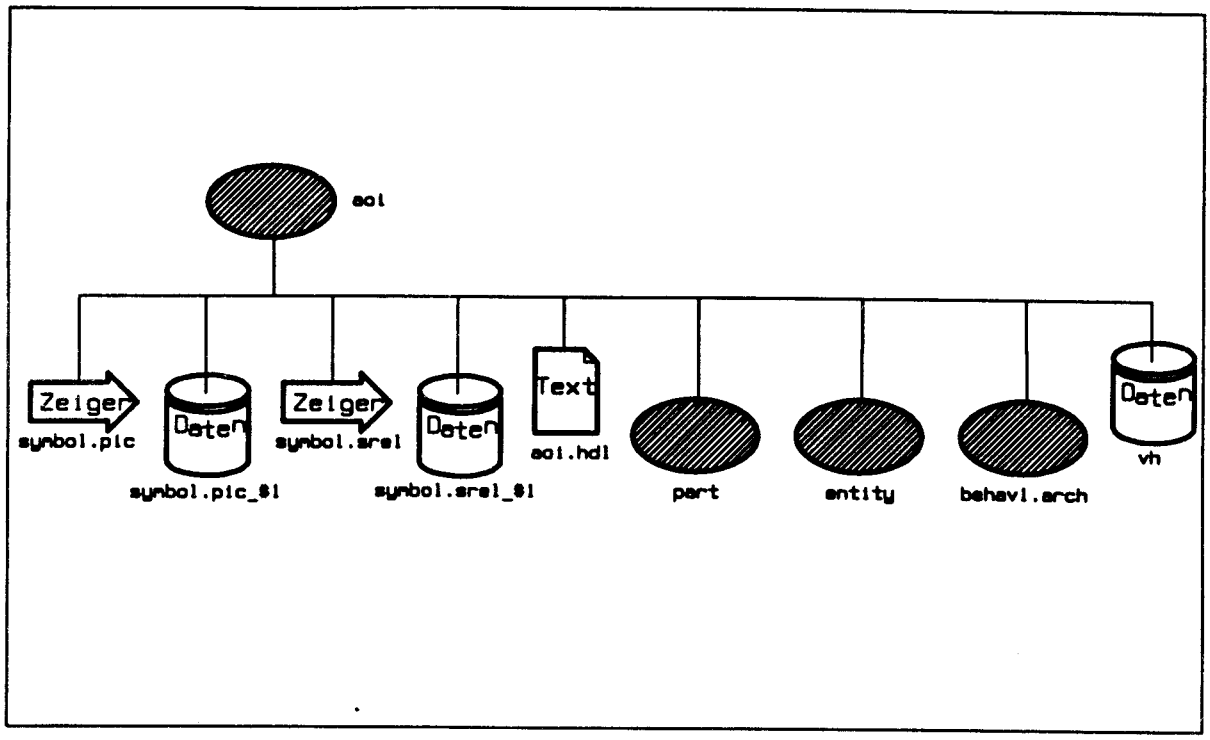


Bild 14: Inhalt des Verzeichnisses aoi nach dem Kompilieren

Beim kompilieren werden folgende Vezeichnisse und Dateien erzeugt:

- Ein Verzeichnis mit dem Namen part. Der Inhalt dieses Verzeichnisses ist nicht von Bedeutung.

- Ein Verzeichnis mit dem Namen entity. Dieses Verzeichnis enthält Dateien, welche die Schnittstelle zum Schaltzeichen, d.h. den Entity-Block des Programmkodes, siehe 2.4.6, beschreiben. Der Inhalt dieses Verzeichnisses ist nicht von Bedeutung.

- Ein Verzeichnis mit dem Namen behav1.arch. Der Name dieses Verzeichnisses setzt sich aus dem Architecture-Namen, siehe 2.4.7, und dem Suffix .arch zusammen. Das Verzeichnis behav1.arch enthält somit Dateien, welche das Verhalten des Entwurfs, d.h. den Architecture-Block des Programmkodes, siehe 2.4.7, beschreiben.

- Die Datendatei mit dem Namen vh.

2.6 Übersetzen des Entwurfs mit dem Programm Expand

Damit das zeitliche Verhalten des Entwurfs simuliert werden kann, muß der Entwurf, in diesem Fall der Stromlaufplan mit dem Namen aoi_design, in welchem das Schaltzeichen plaziert wurde (siehe Kapitel 2.3), mit dem Programm Expand übersetzt werden. Der Aufruf des Programms Expand erfolgt von eines Shell-Ebene aus durch:

$ expand_sim aoi_design <RETURN> .

Eine Einführung in die Handhabung des Programms Expand findet man im Handbuch zum EDA-Seminar [15].

2.7 Simulation des Entwurfs mit dem Programm Quicksim

Die Beschreibung der Handhabung des Programms Quicksim von Mentor Graphics würde den Rahmen dieser Diplomarbeit überschreiten. Im folgenden werden nur die Simulationsergebnisse gezeigt und erläutert.

Eine Einführung in die Handhabung des Programms Quicksim findet man im Idea Station Workbook [16].

Bild 15 zeigt das Simulationsergebnis als Zustandszeitdiagramm, wobei das logische Verhalten bei allen möglichen Eingangszustandskombinationen, gemäß der Wahrheitstabelle, welche Bild 7 zeigt, simuliert wurde.

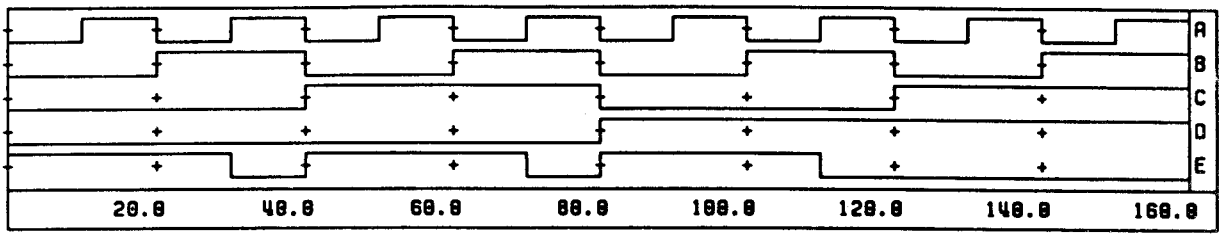


Bild 15: Zustandszeitdiagramme der Simulation

Nachstehend ist das Simulationsergebnis nochmals, in diesem Fall als Liste der Einangs- und Ausgangszustände mit einem Bezug zur Zeit, gemäß Bild 15, abgebildet. Dieses Ergebnis entspricht der Wahrheitstabelle aus Bild 7.

```
  0.0 0   0   0   0   1
 10.0 1   0   0   0   1
 20.0 0   1   0   0   1
 30.0 1   1   0   0   0
 40.0 0   0   1   0   1
 50.0 1   0   1   0   1
 60.0 0   1   1   0   1
 70.0 1   1   1   0   0
 80.0 0   0   0   1   1
 90.0 1   0   0   1   1
100.0 0   1   0   1   1
110.0 1   1   0   1   0
120.0 0   0   1   1   0
130.0 1   0   1   1   0
140.0 0   1   1   1   0
150.0 1   1   1   1   0

TIME   ^A ^B ^C ^D ^E
```

2.8 Beschreibung des Entwurfs AOI in unterschiedlichen Ebenen

In Kapitel 2.4.3 wurde der Programmkode des Entwurfs AOI (gezeigt im Programmkode 1) angegeben. Durch die VHDL-Anweisung *PROCESS* im Architecture-Block (siehe Kapitel 2.4.7) wird der VHDL-Programmkode als eine Behavioral-VHDL-Beschreibung mit einem Process-Block (siehe Kapitel 2.4.7.1) erkannt. Der Prozess-Körper des Programmkodes 1 besteht nur aus einer Zeile.

Programmkode 5: Process-Körper des Programmkodes 1

```
(14)   E <= NOT ((A AND B) OR (C AND D) );
```

Diese Zeile beschreibt das logische Verhalten des Entwurfs.

2.8.1 Entwurf einer detaillierteren Behavioral-VHDL-Beschreibung

Um den Entwurfs AOI detaillierter, d.h. hardwarenäher zu beschreiben, muß nur der Architecture-Block (siehe Kapitel 2.4.7) des VHDL-Programmkodes geändert werden. Der Entity-Block einer VHDL-Beschreibung (siehe Kapitel 2.4.6) bleibt identisch, da dieser von der Hardwarebeschreibungsform unabhängig ist.

Um den Entwurf, welcher durch den in Kapitel 2.4.3 gezeigten Programmkode (Programmkode 1) beschrieben ist, detaillierter zu beschreiben, kann der Programmkode folgendermaßen geändert werden.

Programmkode 6: Programmkode zur detaillierteren Beschreibung des Entwurfs AOI

```
(1)   USE std.mentor_base.all;
(2)
(3)   ENTITY aoi IS
(4)       PORT (A, B, C, D : IN  qsim_state ;
(5)                          E : OUT qsim_state);
(6)   END aoi;
(7)
(8)   ARCHITECTURE behav2 OF aoi IS -- Festlegung des Architecture-Namens
(9)
(10)   BEGIN
(11)
(12)    p1 :  PROCESS (A, B, C, D)
(13)
(14)       VARIABLE O1, O2, O3 : qsim_state ; -- Variablendeklarierung
(15)                                          -- im Process-Kopf
(16)     BEGIN
(17)
(18)       O1 := A AND B;    -- Zeilen werden nacheinander abgearbeitet
(19)       O2 := C AND D;
(20)       O3 := O1 OR O2;
(21)       E  <= NOT O3;
```

```
(22)
(23)     END PROCESS p1;
(24)
(25)   END behav2;
```

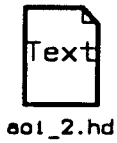
Durch die Einführung der 3 Variablen O1, O2 und O3, welche im Process-Kopf (Zeile 14) zur Verwendung im Process-Körper deklariert wurden (siehe Kapitel 2.4.7.1 und 3.2.2) kann das Verhalten des Entwurfes dadurch detaillierter beschrieben werden, daß der Process-Körper des Prozesses p1 (Zeile 17 bis 22) im Gegensatz zum Process-Körper des Programmkodes 1, siehe Programmkode 5, detaillierter beschrieben wird.

Innerhalb des Process-Körpers des Prozesses p1 im Programmkode 6 werden alle VHDL-Anweisungen sequentiell ausgeführt. Dies bedeutet, daß, sobald sich der Wert eines oder mehrerer Signale der Sensitivity-Liste (siehe Kapitel 2.4.7.1) ändert (die Sensitivity-Liste wurde in Zeile 12 festgelegt), die Zeilen 17 bis 22, welche den Process-Körper bilden, der Reihe nach abgearbeitet werden.

In den Zeilen 18 und 19 des Programmkodes 6 wird den Variablen mit den Namen O1 und O2 der durch die logische Verknüpfung der Werte der Anschlußsignale mit den Signalnamen A bis D gebildete Wert durch eine Variablenzuweisung zugeordnet. Durch die logische Verknüpfung der so gebildeten Werte der Variablen O1 und O2 in Zeile 20 wird dann der Wert der Variable O3 gebildet. In der Zeile 21 wird dann der Wert des Anschlußsignals E durch eine Signalzuweisung gebildet.

2.8.1.1 Simulation der detaillierteren Beschreibung des Entwurfs AOI

Um das korrekte Verhalten dieser Beschreibung überprüfen zu können, ist es nun sinnvoll, das zeitabhängige Verhalten dieser detaillierteren Beschreibung zu simulieren, ohne die Daten der vorhergegangenen Beschreibung (Programmkode 1) zu verlieren.



Dies ist möglich, indem für den Programmkode der detaillierteren Beschreibung, in diesem Fall für den Programmkode 6, eine neue Textdatei angelegt wird (vergleiche Kapitel 2.4.2). Für den Namen der Textdatei, welche den detaillierteren VHDL-Programmkode beinhalten soll nun der Name aoi_2 mit dem Suffix .hdl verwendet werden.

Das Verzeichnis aoi, dargestellt in Bild 16, enthält nun zusätzlich zu der in Bild 14 dargestellten Verzeichnisstruktur die Textdatei aoi_2.hdl.

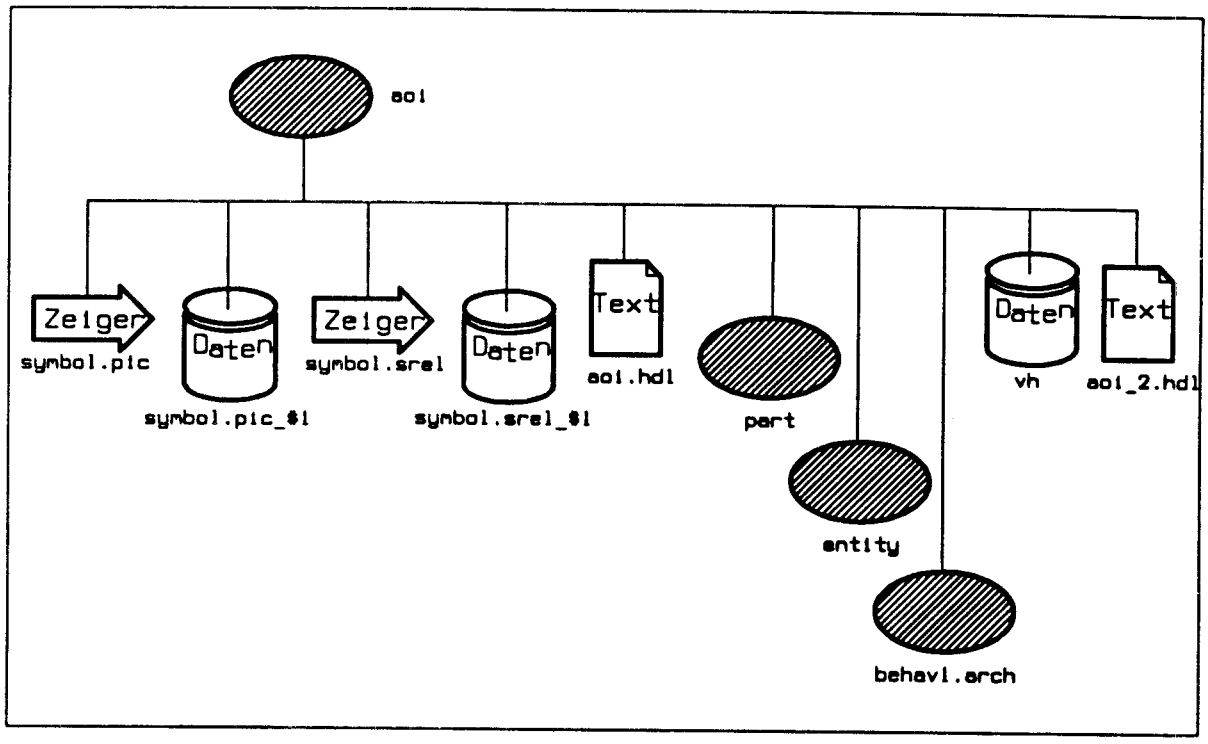


Bild 16: Inhalt des Verzeichnisses aoi nach dem Einfügen der Datei aoi_2.hdl

Nach dem fehlerfreien Kompilieren des VHDL-Programmkodes in der in Kapitel 2.5 gezeigten Vorgehensweise, enthält das Verzeichnis aoi die in Bild 17 gezeigten Dateien, Zeiger und Verzeichnisse. Im Gegensatz zu der in Bild 14 gezeigten Verzeichnisstruktur wurde bei diesem Kompilierungsvorgang nur das Verzeichnis behav2.arch erzeugt. Der Name dieses Verzeichnisses setzt sich aus dem Architecture-Namen (siehe Kapitel 2.4.7), welcher in Zeile 8 des Programmkodes 6 festgelegt wurde und dem Suffix .arch zusammen. Wie bereits in Kapitel 2.5.1 erläutert, enthält das Verzeichnis behav2.arch ebenso wie das Verzeichnis behav1.arch,

welches beim ersten Kompilierungsvorgang erzeugt wurde, Dateien, welche das Verhalten des Entwurfs, d.h. den Architecture-Block des Programmkodes, siehe Kapitel 2.4.7, beschreiben. Es gibt nun also zwei Verzeichnisse, deren Dateien das Verhalten des Entwurfs in einer für den Simulator Quicksim verständlichen Weise beschreiben. Wichtig ist deshalb, daß für unterschiedliche Beschreibungen des Entwurfs auch unterschiedliche Architecture-Namen vergeben werden.

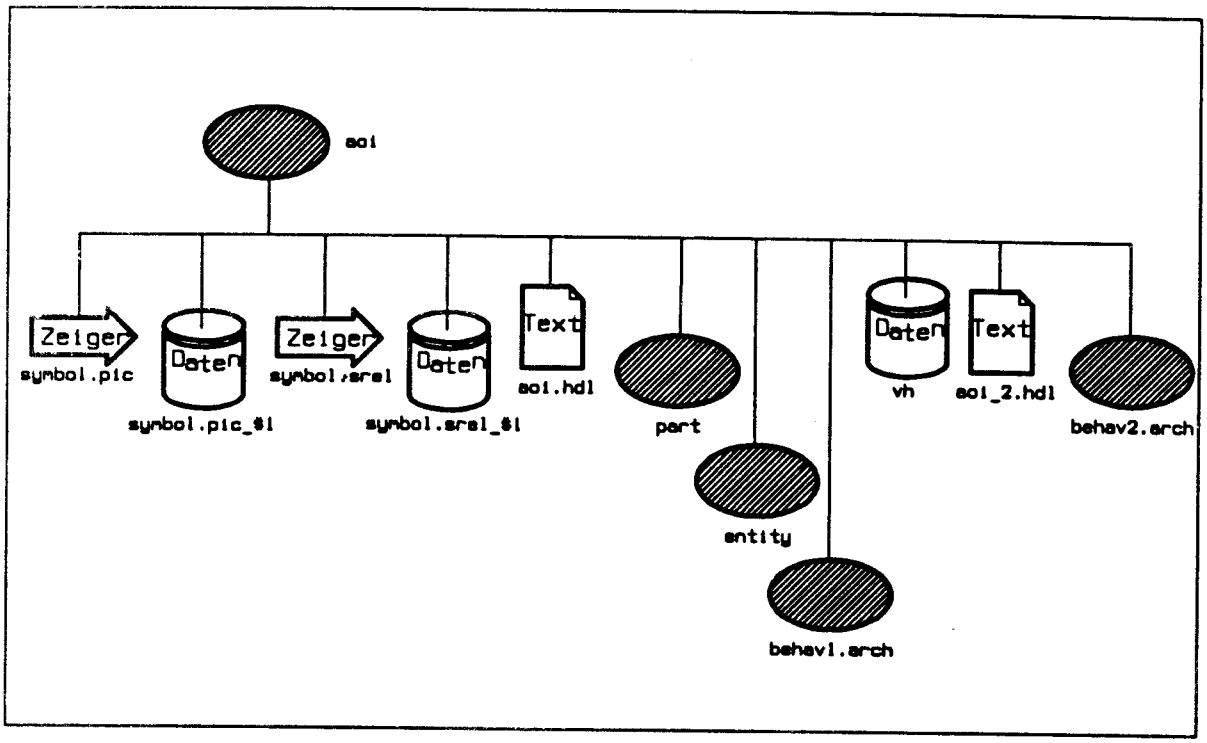


Bild 17: Inhalt des Verzeichnisses aoi nach dem Kompilieren der detaillierteren VHDL-Beschreibung

In Kapitel 2.2 wurde dem Schaltzeichenkörper das Merkmal model (model property) mit dem Wert $hdl zugeordnet. Durch diese Zuordnung wird dem Simulationsprogramm mitgeteilt, daß die zuletzt kompilierte Verhaltensbeschreibung des Schaltzeichens bei der Simulation des zeitabhängigen Verhaltens benutzt werden soll. Bei einer erneuten Simulation des zeitabhängigen Verhaltens des Entwurfs würde nun die detailliertere VHDL-Beschreibung zum Einsatz kommen. Um dem Schaltzeichen fest vorzuschreiben, welche VHDL-Beschreibung verwendet wird, kann nun statt dem Wert $hdl des Merkmals model der Architecture-Name verwendet

werden. Das Merkmal model (model property) kann sowohl im Programm Neted als auch im Programm Symed geändert werden. Bild 18 zeigt die möglichen Werte des Merkmals model (siehe auch Bild 8).

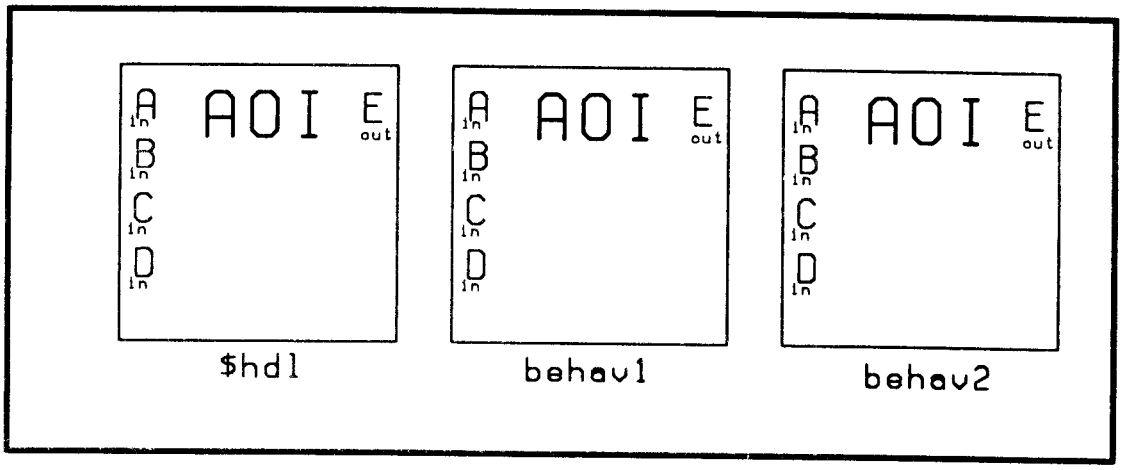


Bild 18: Schaltzeichen mit den möglichen Werten des Merkmals model

Nach dem Übersetzen des Entwurfs mit dem Programm Expand, siehe Kapitel 2.6, können nun das zeitabhängige Verhalten des Entwurfs mit dem Programm Quicksim simuliert und anschließend die Simulationsergebnisse mit denen aus Kapitel 2.7 verglichen werden.

2.8.2 Beschreibung eines Architekturansatzes

Durch eine detailliertere Behavioral-VHDL-Beschreibung kann ein Architekturansatz entworfen werden. Jeder Prozeß entspricht hierbei einem Teilschaltwerk in der Schaltungstechnik. Signale übernehmen Aufgaben, die den elektrischen Verbindungen zwischen den Teilschaltwerken in der Schaltungstechnik entsprechen.

Bild 19 zeigt den zu beschreibenden Architekturansatz. Dieser Architekturansatz teilt den Entwurf in 4 Teilschaltwerke mit den Namen p1 bis p4 auf. Die elektrischen Verbindungen zwischen den Teilschaltwerken haben die Namen O1 bis O3.

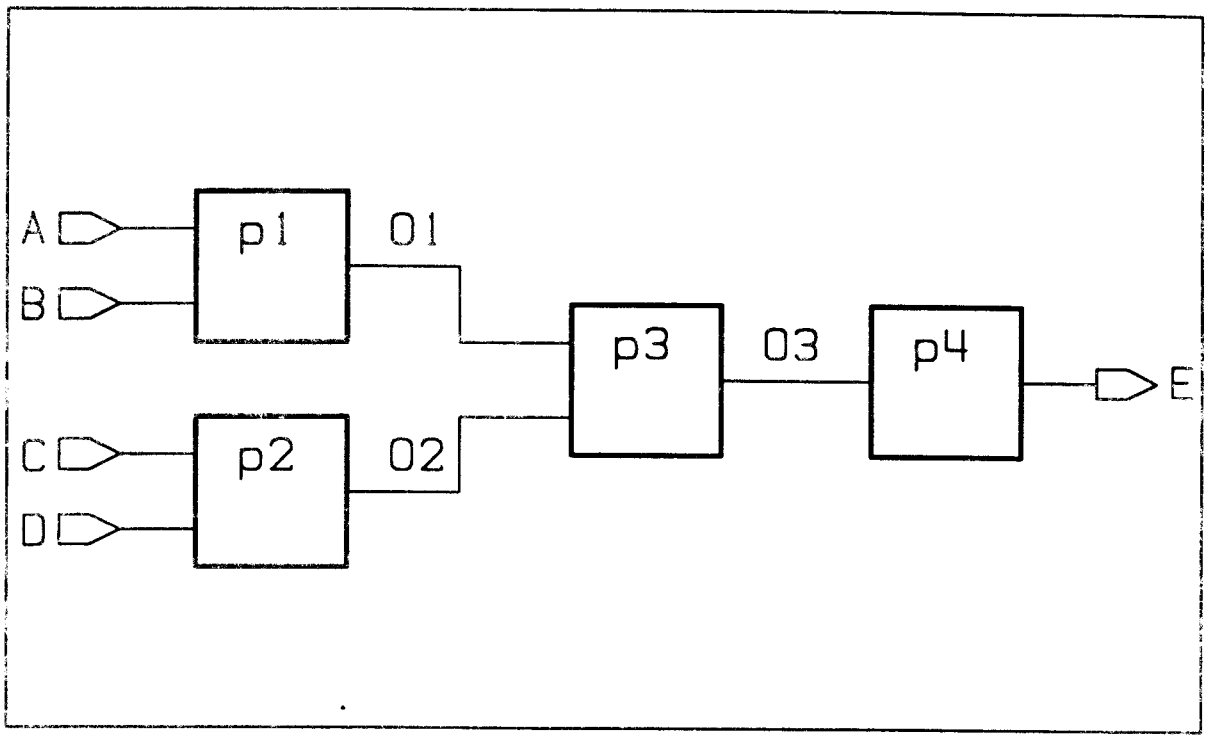


Bild 19: Architekturansatz zur detaillierteren Beschreibung des Entwurfs AOI

Der VHDL-Programmkode der Entwurfs AOI, welcher den Architekturansatz, dargestellt in Bild 19, beschreibt, ist nachstehend als Programmkode 7 aufgeführt.

Programmkode 7: Programmkode des Entwurfs AOI, welcher den Architekturansatz beschreibt

```
(1) USE std.mentor_base.ALL;                    -- Bibliotheksaufruf
(2)
(3) ENTITY aoi IS                               -- Beschreibung der Schnittstelle
(4)  PORT (A, B, C, D : IN  qsim_state;         -- zum Schaltzeichen
(5)                E : OUT qsim_state);
(6) END aoi;
(7)
(8)  ARCHITECTURE behav3 OF aoi IS          -- Beginn der Verhaltensbeschreibung
(9)
(10)     SIGNAL  O1: qsim_state ;               -- Signaldeklarierung
(11)     SIGNAL  O2: qsim_state ;               -- im Architecture-Kopf
(12)     SIGNAL  O3: qsim_state ;
(13)
(14)  BEGIN
(15)
```

```
(16)     p1: PROCESS (A, B)                    --Prozess p1
(17)          BEGIN                            --zur Bildung von o1
(18)           O1 <= A AND B;
(19)         END PROCESS p1;
(20)
(21)     p2: PROCESS (C, D)                    --Prozess p2
(22)          BEGIN                            --zur Bildung von o2
(23)           O2 <= C AND D;
(24)         END PROCESS p2;
(25)
(26)     p3: PROCESS (O1, O2)                  --Prozess p3
(27)          BEGIN                            --zur Bildung von o3
(28)           O3 <= O1 OR O2;
(29)         END PROCESS p3;
(30)
(31)     p4: PROCESS (O3)                      --Prozess p4
(32)          BEGIN                            --zur Bildung von E
(33)           E <= NOT O3;
(34)         END PROCESS p4;
(35)
(36)  END behav3;
```

Die Behavioral-VHDL-Beschreibung, vorstehend gezeigt als Programmkode 7, ist analog zu den Teilschaltwerken des Architekturansatzes, gemäß Bild 19, in 4 Prozesse aufgeteilt. Dabei wurde für die Process-Namen (p1 bis p4) die gleichen Namen wie für die Teilschaltwerke verwendet. Weiterhin haben in der VHDL-Beschreibung die im Architekture-Kopf deklarierten Signale die gleichen Namen (O1 bis O3) wie die elektrischen Verbindungen zwischen den Teilschaltwerken in Bild 19.

Wie bereits in Kapitel 2.4.7.1 erwähnt, werden innerhalb eines Prozesses alle Programmschritte der Reihe nach abgearbeitet, sind jedoch mehrere Prozesse aktiv, so werden diese parallel abgearbeitet. Ein Prozeß wird aufgerufen, wenn sich der Wert eines oder mehrerer Signale der Sensitivity-Liste ändert. Die Signalnamen, welche in den Sensitivity-Listen der VHDL-Beschreibung aufgeführt sind, entsprechen den Namen der elektrischen Verbindungen, welche mit den Eingängen der Teilschaltwerke, dargestellt in Bild 19, verbunden sind.

Ändert sich beispielsweise der Wert des Anschlußsignals A, so wird dadurch der Prozeß p1 aufgerufen, da sich das Anschlußsignal A in der Sensitivity-Liste des Prozesses p1 befindet

(siehe Kapitel 2.4.7.1), und der Process-Körper des Prozesses p1 (Zeile 18 im Programmkode 7) wird abgearbeitet. Ändert sich dadurch der Wert des Signals O1, so wird dadurch wiederum der Prozeß p3 aufgerufen. Falls im Process-Körper des Prozesses p3 der Wert des Signals O3 geändert wird, wird daraufhin der Prozess p4 abgearbeitet und der Wert des Anschlußsignals E geändert.

Aufgrund dieser gegenseitigen Aktivierung der Prozesse durch Signale ist es gleichgültig, an welcher Stelle sich die Process-Blöcke, siehe Kapitel 2.4.7.1, im Architecture-Körper der VHDL-Beschreibung befinden. Es ist deshalb nicht notwendig, daß der Prozeß p4 an letzter Stelle im Programmkode 7 aufgeführt wurde.

Falls sich die Werte von zwei Signalen zeitgleich ändern, wie beispielsweise die Werte der Anschlußsignale A und C, so werden dadurch die Prozesse p1 und p2 zur Bildung der Signale O1 und O2 gleichzeitig aufgerufen und die Process-Körper der Prozesse p1 und p2 (Zeile 18 und 23 im Programmkode 7) abgearbeitet.

2.8.2.1 Simulation der Beschreibung des Architekturansatzes

Die Simulation des zeitabhängigen Verhaltens des Entwurfs, dessen Verhalten durch den Programmkode 7 beschrieben wurde, zeigt Bild 20. Bei der Simulation wurde davon Gebrauch gemacht, daß zeitabhängige Werteverläufe von Signalen, in diesem Fall die Werteverläufe der Signale O1, O2 und O3, im Zustandszeitdiagramm sichtbar gemacht werden können (siehe Kapitel 3.2). Das logische Verhalten wurde bei allen möglichen Eingangszustandskombinationen, gemäß der Wahrheitstabelle in Bild 7 bzw. der Simulation in Kapitel 2.7, simuliert. Das Ergebnis kann nun mit dem Simulationsergebnis der Beschreibung in Kapitel 2.7, welches in Bild 15 gezeigt wurde verglichen werden.

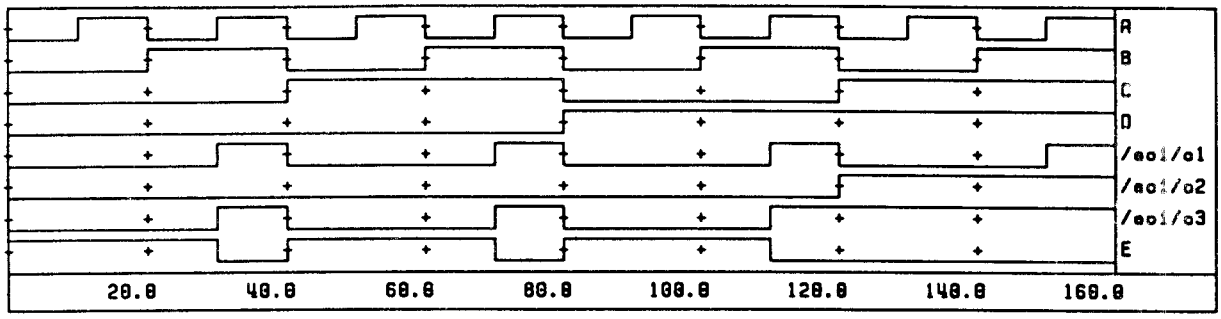


Bild 20: Zustandszeitdiagramme der Simulation

Nachstehend ist das Simulationsergebnis nochmals, in diesem Fall als Liste der Eingangs- und Ausgangszustände, sowie der Werte der Signale O1 bis O3, mit einem Bezug zur Zeit abgebildet. Dieses Ergebnis entspricht der Wahrheitstabelle aus Bild 7 und kann mit der Liste, welche in Kapitel 2.7 gezeigt wurde, auf Übereinstimmung der Werte der Anschlußsignale verglichen werden.

```
  0.0 0  0  0  0  0   0   0   1
 10.0 1  0  0  0  0   0   0   1
 20.0 0  1  0  0  0   0   0   1
 30.0 1  1  0  0  1   0   1   0
 40.0 0  0  1  0  0   0   0   1
 50.0 1  0  1  0  0   0   0   1
 60.0 0  1  1  0  0   0   0   1
 70.0 1  1  1  0  1   0   1   0
 80.0 0  0  0  1  0   0   0   1
 90.0 1  0  0  1  0   0   0   1
100.0 0  1  0  1  0   0   0   1
110.0 1  1  0  1  1   0   1   0
120.0 0  0  1  1  0   1   1   0
130.0 1  0  1  1  0   1   1   0
140.0 0  1  1  1  0   1   1   0
150.0 1  1  1  1  1   1   1   0

TIME  ^A ^B ^C ^D ^o1 ^o2 ^o3  ^E
```

3. Typen und Objekte

3.1 Typendeklarierung

In jedem Programm ist man mit verschiedenen Objekten, also Konstanten, Variablen und Signalen (siehe Kapitel 3.2), unterschiedlicher Art befaßt. Ein fundamentales Prinzip in VHDL ist nun, daß jedes Objekt von einem bestimmten Typ sein muß, der die möglichen, annehmbaren Werte des Objektes bestimmt. Zum Beispiel kann ein Anschlußsignal, siehe Kapitel 2.4.6, die Werte 'X' '0' '1' und 'Z' annehmen, oder ein Byte die Werte 0 bis 255.

Ebenso wie in anderen typenstrengen Programmiersprachen, wie z.B. Pascal, muß in VHDL für alle in einem VHDL-Programm vorkommenden Objekte ein Typ deklariert (vereinbart) werden, damit die Verwendung der Objekte geprüft werden kann. VHDL enthält einige vordefinierte Typen, wie z.B. Integer oder Real, aber die meisten Typen im Programm werden durch den Programmierer eingeführt, um dem speziellen Anwendungsfall gerecht zu werden. Ein Typ gibt dann den möglichen Wertebereich für ein Objekt an.

Es geht bei der Typendeklarierung also nicht so sehr um einen augenblicklich aktuellen Wert, sondern vielmehr um die Menge der erlaubterweise annehmbaren Werte. Dies ist von großer Bedeutung beim Testen eines Programms sowie bei der Logiksynsthese, welche effektiver ist, wenn der Wertebereich eines Objektes eingegrenzt ist. Bei einem zu entwerfenden Zähler, dessen maximaler Zählerstand 10, ist sollte die in dem ihm beschreibenden Programm verwendete Zählvariable den Wertebereich 0 bis 10 besitzen, um bei der nachfolgenden Logiksynthese ein optimales Ergebnis zu erhalten.

3.1.1 Skalare Typen

Als skalare Datentypen bezeichnet man Zahlen (Ganz- und Gleitpunktzahlen) , Zeichen, Wahrheitswerte (false und true), physikalische Werte (wie z.B. die Zeit) und Aufzählungsdatentypen, in denen alle zugehörigen Werte explizit aufzulisten sind.

Ein Beispiel für einen Aufzählungsdatentyp gibt die folgende Typendeklarierung:

```
TYPE innerer_zustand IS (m0, m1, m2, m3, m4, m5, m6);
```

oder

```
TYPE schalterstellung IS (ein, aus);
```

Damit werden für Objekte vom Typ schalterstellung die möglichen Werte festgelegt, nämlich entweder ein oder aus.

Die VHDL-Anweisung *TYPE* leitet die Typendeklarierung ein, gefolgt von einem Bezeichner, welcher den Namen des Typs darstellt, der Anweisung *IS* und einer Liste in der oben gezeigten Form oder einer Bereichswahl mit Hilfe der VHDL-Anweisungen *RANGE* und *TO*. Beispiel einer Typendeklarierung mit der Anweisung *RANGE*:

```
TYPE zaehlerstand IS RANGE 0 TO 10;
```

Damit werden für Objekte des Typs zaehlerstand die möglichen Werte festgelegt, nämlich eine der Ganzzahlen im Bereich zwischen 0 und 10. Falls statt Ganzahlen Gleitpunktzahlen gewünscht werden, muß bei der Bereichsangabe mindestens eine Nachkommanstelle mit angegeben werden. Bsp:

```
TYPE spannung IS RANGE 0.0 TO 5.0;
```

Unter Physikalischen Typen versteht man Typen, bei denen eine Einheit mit angegeben werden kann. Als Beispiel wird ein Typ mit dem Namen widerstandswert deklariert werden. Dies geschieht folgendermaßen:

```
TYPE widerstandswert IS RANGE 0 TO 1E9
  UNITS
   ohm;                          --Basiseinheit Ohm
   kohm = 1000 ohm;              --Einheit  KiloOhm
```

```
  mohm = 1000 kohm;                --Einheit  MegaOhm
END UNITS;
```

Mit der Anweisung *TYPE* wird für Objekte des Typs widerstand die möglichen Werte festgelegt. Mit der Anweisung *UNITS* werden anschließend die Einheiten festgelegt. Abgeschlossen wird die Deklarierung eines physikalischen Typs mit der Anweisung *END UNITS* und einem Semikolon.

3.1.1.1 Vordefinierte skalare Typen

Vordefinierte skalare Typen sind:

- integer

  Der Typ integer umfaßt Ganzzahlen im Bereich von -2147483648 bis 2147483647

- real

  Der Typ real umfaßt Gleitpunktzahlen im Bereich -1.79769E308 bis 1.79769E308.

- bit

  Der Typ bit beinhaltet die Zeichenliterale '0' und '1'.

- boolean

  Der Typ boolean umfaßt die Wahrheitswerte false und true.

- character

  Der Typ character umfaßt sämtliche druckbaren Zeichenliterale wie 'A', 'a', '1', '!', '#' und ' '.

- time

  Der physikalische Typ time umfaßt die Zeit im Bereich -1.569E57 bis 1.569E57 Femtosekunden. Dabei sind die Einheiten

  fs (Femtosekunden),

ps (Picosekunden),
ns (Nanosekunden),
us (Microsekunden),
ms (Millisekunden),
sec (Sekunden),
min (Minuten) und
hr (Stunden) festgelegt.

- qsim_state
  Der Typ qsim_state ist in der Unterprogrammbibliothek std.mentor_base festgelegt und umfaßt die Zeichenliterale 'X', '0', '1' und 'Z'. Siehe Kapitel 2.4.6 .

3.1.2 Deklarierung von Untertypen (Subtypes)

In vielen Fällen nimmt ein Objekt nur Werte aus einer Teilmenge der möglichen Werte eines bestimmten Datentyps an. Es ist daher naheliegender, diesen Fall als Einschränkung des zugrundeliegenden Datentyps zu betrachten, als einen vollständig neuen Typ zu deklarieren.

Wenn der Typ speicherbereich folgendermaßen deklariert ist:

```
TYPE speicherbereich IS RANGE 0 TO 65535; -- 64 kByte
```

kann der Typ datenwort als Untertyp des Typs speicherbereich folgendermaßen deklariert werden:

```
SUBTYPE datenwort IS speicherbereich RANGE 0 TO 255;
```

Die VHDL-Anweisung *SUBTYPE* leitet die Deklarierung eines Untertyps ein, gefolgt von einem Bezeichner, welcher den Namen des Untertyps darstellt, der Anweisung *IS*, einem weiteren Bezeichner, welcher den Namen des zugrundeliegenden Datentyps trägt und der Anweisung *RANGE*, welche zusammen mit der Anweisung *TO* den Bereich des Untertyps festlegt.

3.1.2.1 Vordefinierte Untertypen

Vordefinierte Untertypen sind:

- natural

  Der Untertyp natural ist ein Untertyp des Typs integer und umfaßt Ganzzahlen im Bereich von 0 bis 2147483647.

- positive

  Der Untertyp positive ist ein Untertyp des Typs integer und umfaßt Ganzzahlen im Bereich von 1 bis 2147483647.

3.1.3 Feldtypen (Array Types)

Häufig bestehen Datenwerte aus einer Anzahl verschiedener Komponenten, die aber alle vom gleichen Typ sind. Die einzelnen Datenwerte werden durch einen Index angesprochen, wobei der Typ des Index entweder vom Typ integer oder ein vom Typ integer abgeleiteter Typ sein muß.

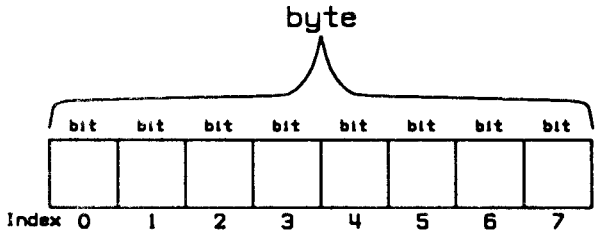


Bild 21: Form des Types byte

Um beispielsweise 8 Objekte vom Typ bit, siehe Kapitel 3.1.1.1, zu einem neuen Typ mit dem Namen byte zusammenzufassen, siehe Bild 21, nimmt man folgende Typendeklarierung vor:

```
TYPE byte IS ARRAY (integer RANGE 0 TO 7) OF bit;
```

Die VHDL-Anweisung *TYPE* leitet die Deklarierung eines Feldtypes ein, gefolgt von einem Bezeichner, welcher den Namen des Feldtyps darstellt, der Anweisung *IS ARRAY*, einer runden Klammer, des Typs des Index, der Anweisung *RANGE*, welche zusammen mit der Anweisung *TO* den Bereich des Index festlegt, einer geschlossenen Klammer, der Anweisung *OF*, einem weiteren Bezeichner, welcher den Namen des Typs der Kopmponenten trägt und einem Semikolon.

Jedes Objekt vom Typ byte setzt sich aus 8 Komponenten zusammen, die alle vom Typ bit sind. Ein solchen solchen Feldtyp bezeichnet man auch als eindimensionalen Feldtyp.

Indem die in Klammern stehenden Bezeichner und Anweisungen durch ein Komma getrennt mehrfach aufgeführt werden, lassen sich auch mehrdimensionale Feldtypen deklarieren.
Beispiel eines zweidimensionalen Feldtyps:

```
TYPE schachbrett IS ARRAY (positive RANGE 1 TO 8,
                           positive RANGE 1 TO 8) OF figuren;
```

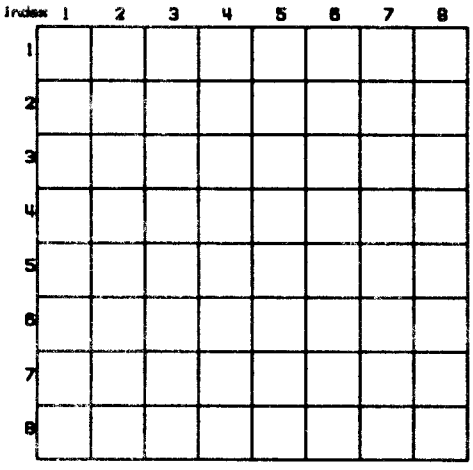


Bild 22: Form des Types schachbrett

Hierbei handelt es sich um ein Feldyp welcher aus 8 Zeilen und 8 Spalten besteht, siehe Bild 22, also insgesamt 64 Komponenten beinhaltet, welche alle vom Typ figuren sind. Als

Index dient hierbei der vordefinierte Untertyp positive, siehe Kapitel 3.1.2.1, welcher vom Typ integer abgeleitet ist.

Der Typ figuren könnte beispielsweise ein folgendermaßen deklarierter Aufzählungsdatentyp, siehe Kapitel 3.1.1, sein:

```
TYPE figuren IS (leeres_feld,weisser_stein, schwarzer_stein);
```

oder

```
TYPE figuren IS (leeres_feld,
                 w_bauer, w_turm, w_pferd, w_laeufer,
                 w_dame,w_könig,
                 s_bauer, s_turm, s_pferd, s_laeufer,
                 s_dame,s_könig);
```

Ein Objekt vom Typ byte, siehe Objektdeklarierung Kapitel 3.2, nennt man auch eindimensionales Feld. Ein Objekt vom Typ schachbrett wird auch als zweidimensionales Feld bezeichnet.

3.2 Objektdeklarierung

In VHDL werden drei Arten von Objekten unterschieden:

- Konstanten:
  Der Wert einer Konstante wird bei der Deklarierung festgelegt und kann danach nicht wieder verändert werden.

- Variable:
  Der Wert einer Variable kann durch die Variablenzuweisung, siehe Kapitel 3.3.1 geändert werden.

- Signale:

  Der Wert eines Signals kann durch die Signalzuweisung, siehe Kapitel 3.3.2 geändert werden. Im Gegensatz zu Variablen haben Signale einen Bezug zur Zeit, so kann z.B. der Wert eines Signals zeitlich verzögert geändert und im Simulator in sog. Zustandszeitdiagrammen sichtbar gemacht werden. Signale übernehmen Aufgaben die den elektrischen Verbindungen in der Schaltungstechnik entsprechen, wobei Variable keine direkte Analogie in der Schaltungstechnik haben. Aus diesem Grund ist die Verwendung von Variablen nur in der verhaltensorientierten Hardwarebeschreibungsform, d.h. in Behavioral-VHDL, zulässig.

VHDL ist eine Sprache mit strenger Typbindung, d.h. jedes Objekt, ob Konstante, Variable oder Signal, wird bei seiner Einführung einem bestimmten Typ zugeordnet. Für einen bestimmten Programmteil, z.B. für einen Prozeß, bzw. für das gesamte Programm wird ein Objekt mittels einer Objektdeklarierung eingeführt. Innerhalb dieses Programmteils können dann dem Objekt, vorausgesetzt es handelt sich nicht um eine Konstante, nur Werte des zugeordneten Typs zugewiesen werden, siehe Kapitel 3.3.

3.2.1 Konstantendeklarierung

In der Konstantendeklarierung werden der Bezeichner, der Datentyp und ein Initialwert angegeben.
Beispiel einer Konstantendeklarierung:

```
CONSTANT p5 : integer := 5;
```

Die Deklarierung von Konstanten wird durch die Anweisung *CONSTANT* eingeleitet, gefolgt von einem Bezeichner, welcher den Namen der zu deklarierenden Konstante darstellt, einem Doppelpunkt, dem Datentyp, der Zuweisungsanweisung :=, einem Initialwert und einem Semikolon.

Bei der Deklarierung wird der Wert der Konstanten p5 durch den Initialwert festgelegt. Dieser Initialwert muß vom zugeordneten Typ, in diesem Fall vom Typ integer, sein. Der Wert einer Konstante wird bei der Deklarierung festgelegt und kann danach nicht wieder verändert werden.

Falls mehrere Konstanten vom gleichen Typ mit demselben Initialwert festgelegt werden sollen, so kann eine Bezeichnerliste angelegt werden. Beispiel:

```
CONSTANT p5, vcc, high, logisch_1 : integer := 5;
```

Weitere Beispiele der Konstantendeklarierung:

```
CONSTANT verzoegerung : time := 20 ns;
```

Der physikalische Typ time ist vordefiniert, siehe Kapitel 3.1.1.1.

```
CONSTANT kontrollbyte : byte := ('1','0','1','0',
                                 '1','1','0','0');
```

Der Feldtyp byte wurde in Kapitel 3.1.3 als Beispiel deklariert.

3.2.2 Variablendeklarierung

Die Verwendung von Variablen, d.h. auch die Deklarierung von Variablen, ist nur in der verhaltensorientierten Hardwarebeschreibungsform, d.h. in Behavioral-VHDL, zulässig.

In der Variablendeklarierung werden der Bezeichner, der Datentyp und wahlweise ein Initialwert angegeben.
Beispiel einer Variablendeklarierung:

```
VARIABLE ergebnis : integer := 0;
```

Die Deklarierung von Variablen wird durch die Anweisung *VARIABLE* eingeleitet, gefolgt von einem Bezeichner, welcher den Namen der zu deklarierenden Variable darstellt, einem Doppelpunkt, dem Datentyp, wahlweise der Zuweisungsanweisung := zusammen mit einem Initialwert und einem Semikolon.

Bei der Deklarierung wird der Wert der Variablen ergebnis durch den Initialwert festgelegt. Dieser Initialwert muß vom zugeordneten Typ, in diesem Fall vom Typ integer, sein. Der Wert der Variable ergebnis wird bei der Deklarierung festgelegt, er kann jedoch durch eine Variablenzuweisung, siehe Kapitel 3.3.1, geändert werden.

Wird bei der Variablendeklarierung auf die Angabe eines Initialwerts verzichtet, so ist der Wert der Variable vor der ersten Variablenzuweisung unbekannt.
Beispiel einer Variablendekalrierung ohne die Angabe eines Initialwertes:

```
VARIABLE ergebnis : integer; -- Deklarierung ohne Initialwert
```

Falls mehrere Variablen vom gleichen Typ und wahlweise mit dem gleichen Initialwert festgelegt werden sollen, so kann eine Bezeichnerliste angelegt werden.
Beispiel einer Variablendeklarierung mit Hilfe einer Bezeichnerliste:

```
VARIABLE schalter1,schalter2,schalter3: schalterstellung := aus;
```

Der Typ schalterstellung wurde in Kapitel 3.1.1 als Beispiel für die Typendeklarierung deklariert.

3.2.3 Signaldeklarierung

Die Verwendung von Signalen, d.h. auch die Deklarierung von Signalen, ist in allen drei Hardwarebeschreibungsformen, d.h. in Behavioral-, Dataflow- und Structural-VHDL, zulässig.

In der Signaldeklarierung werden der Bezeichner, der Datentyp und wahlweise ein Initialwert angegeben.
Beispiel einer Signaldeklarierung:

```
SIGNAL takt : qsim_state := 'X';
```

Die Deklarierung von Signalen wird durch die Anweisung *SIGNAL* eingeleitet, gefolgt von einem Bezeichner, welcher den Namen des zu deklarierenden Signal darstellt, einem Doppelpunkt, dem Datentyp, wahlweise der Zuweisungsanweisung := zusammen mit einem Initialwert und einem Semikolon.

Bei der Deklarierung wird der Wert der Signals takt durch den Initialwert festgelegt. Dieser Initialwert muß vom zugeordneten Typ, in diesem Fall vom Typ qsim_state, sein. Der Wert der Signal takt wird bei der Deklarierung festgelegt, er kann jedoch durch eine Signalzuweisung, siehe Kapitel 3.3.2, geändert werden.

Wird bei der Signaldeklarierung auf die Angabe eines Initialwerts verzichtet, so ist der Wert des Signal vor der ersten Signalzuweisung unbekannt.
Beispiel einer Signaldekalrierung ohne die Angabe eines Initialwertes:

```
SIGNAL takt : bit; -- Deklarierung ohne Initialwert
```

Falls mehrere Signal vom selben Typ und wahlweise mit demselben Initialwert festgelegt werden sollen, so kann eine Bezeichnerliste angelegt werden.
Beispiel einer Signaldeklarierung mit Hilfe einer Bezeichnerliste:

```
SIGNAL ad0, ad1, ad2 ,ad3 : byte := ('0','0','0','0',
                                     '0','0','0','0');
```

Der Feldtyp byte wurde in Kapitel 3.1.3 als Beispiel deklariert.

3.3 Zuweisungen

Variablen und Signale sind Objekte (siehe Kapitel 3.2), deren Werte sich ändern können. Mittels einer Zuweisungsanweisung kann einer Variablen oder einem Signal ein neuer Wert gegeben werden. Der vorhergehende Wert, der in die Berechnung des neuen Wertes eingehen kann, ist dann nicht mehr vorhanden. Eine Variable oder ein Signal kann von jedem beliebigen Datentyp, siehe Kapitel 3.1, sein. Wird einer Variablen oder einem Signal ein neuer Wert zugewiesen, so muß dieser vom selben Typ sein.

3.3.1 Variablenzuweisungen

Die Wertzuweisung einer Variablen hat folgendes Format:

```
V := E;
```

Dabei stellt V einen Variablennamen und E einen Ausdruck dar. Dadurch wird der Wert von E der Variablen mit dem Namen V durch die Zuweisungsanweisung := zugewiesen. E und V müssen demselben Typ angehören.
Beispiele zur Variablenzuweisung:

```
schalter1 := ein;
ergebnis  := ergebnis + 1;
o1        := a AND b;
```

Die Variablen schalter1 und ergebnis wurden in Kapitel 3.2.2 deklariert. Die Variablen o1, a und b sollen jeweils vom Typ bit sein.

Variablenzuweisungen sind für alle Typen definiert; zum Beispiel können ganzen Feldern Werte direkt zugewiesen werden.
Beispiel:

```
in0 := ('0','0','0','1','0','0','0','1');
in1 := ('1','0','0','0','1','0','0','0');
out := in0 OR in1;      --('1','0','0','1','1','0','0','1')
```

Die Variablen in0, in1 und out sollen jeweils vom Feldtyp byte, welcher in Kapitel 3.1.3 deklariert wurde, sein.

3.3.2 Signalzuweisungen

Die Wertzuweisung eines Signals hat folgendes Format:

```
S <= E AFTER T;
```

Dabei stellt S einen Signalnamen, E einen Ausdruck und T eine Zeit dar. Dadurch wird der Wert von E dem Signal mit dem Namen S durch die Zuweisungsanweisung <= nach der Zeit T zugewiesen. E und S müssen demselben Typ angehören. T muß vom Typ time, siehe Kapitel 3.1.1.1 sein. Falls dem Signal S der Wert von E sofort, d.h. zur momentanen Simulationszeit zugewiesen werden soll, so kann die Anweisung AFTER und die Zeit T weggelassen werden. Beispiele zur Signalzuweisung:

```
takt <= '1';
takt <= '1' AFTER 10 ns;
takt <= '1' AFTER verzoegerung;
takt <= sig1 AND sig2 AFTER verzoegerung;
```

Das Signal takt wurde in Kapitel 3.2.3 als Signal vom Typ qsim_state deklariert. Die Konstante verzoegerung wurde in Kapitel 3.2.1 deklariert. Die Signale sig1 und sig2 sollen jeweils vom Typ qsim_state, siehe Kapitel 3.1.1.1 sein.

Indem der Ausdruck, die Anweisung AFTER und die Zeitangabe durch ein Komma getrennt mehrfach aufgeführt werden, lassen sich auch mehrere zeitlich versetzte Zuweisungen in einer

Zeile ausführen.
Beispiel:

takt <= '1' AFTER 10 ns, '0' AFTER verzoegerung;

Signalzuweisungen sind für alle Typen definiert; zum Beispiel können ganzen Feldern Werte direkt zugewiesen werden. Weiterhin können die Werte von Variablen, siehe Kapitel 3.3.1, auch Signalen zugewiesen werden.
Beispiel:

```
in0 := ('0','0','0','1','0','0','0','1');
in1 := ('1','0','0','0','1','0','0','0');
ad0 <= in0 OR in1 AFTER 100 ns;
```

Die Variablen in0 und in1 sollen jeweils vom Feldtyp byte, welcher in Kapitel 3.1.3 deklariert wurde, sein. Das Signal ad0 wurde in Kapitel 3.2.3 als Signal vom Feldtyp byte deklariert.

A. Literaturverzeichnis

[1] Mentor Graphics: Getting Startet with System-1076 Alpha Release

[2] Mentor Graphics: System-1076 User's manual Alpha Release

[3] Mentor Graphics: System-1076 Error Message Manual Alpha Release

[4] Mentor Graphics: System-1076 Quicksim Family Reference Manual Supplement

[5] Mentor Graphics: System-1076 Reference Manual Version 8.0 Beta Draft

[6] IEEE: IEEE Standard VHDL Language Reference Manual

[7] Lipsett, Schaefer, Ussery: VHDL Hardware Description and Design

[8] David R. Coelho: The VHDL Handbook

[9] Augustin, Luckham, Gennart, Huh, Stanculescu: Hardware Design and Simulation in VAL/VHDL

[10] CADS, Jahrgang 3/Heft 7 vom 31. Oktober 1990

[11] CADS, Jahrgang 3/Heft 6 vom 12. Oktober 1990

[12] CADS, Jahrgang 3/Heft 5 vom 3. September 1990

[13] CADS, Jahrgang 3/Heft 1 vom 12. März 1990

[14] CADS, Jahrgang 2/Heft 5 vom 1. August 1989

[15] Handbuch zum EDA-Seminar, FH Aalen

[16] Mentor Graphics: Idea Station Workbook

[17] I. C. Pyle: Die Programmiersprache ADA

[18] A. Schwald: ADA; Eine Einführung

FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

6. Entwurf , Simulation und Messung von temperaturoptimierten Leistungshybriden

H. KHAKZAR

1. Aufgabenstellung sowie Grundlagen und Modelle der Wärmeübertragung
2. Entwurf und Auswahl einiger Aufbautechniken für Leistungshybride
3. Simulation der Testobjekte
4. Messung der Testobjekte
5. Vergleich zwischen Messung und Simulation
6. Entscheidungshilfen für temperaturoptimiertes Aufbaukonzept

7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto-Nr. 29775 (BLZ 610 500 00)

1.

Aufgabe :

Es sind Tools für ein Konstruktions-Aufbaukonzept für Leistungshybride zu entwickeln, mit deren Hilfe die vielen Varianten kundenspezifischer Leistungshybride (z.B. Motorsteuerungen) wärmetechnisch und elektrisch entwickelt werden können.

Durchführung :

Zu untersuchen sind die Kombinationen verschiedener gut leitender Materialien im Verbund mit Al_2O_3-Substraten.
Dabei sollten die Einflüsse der Dimensionierung, der Anordnung und der Verbindung bezüglich Kühlkörper, Bauelement und Substrat im Hinblick auf den Wärmetransport von der Wärmequelle zur Wärmesenke berücksichtigt werden.
Die physikalischen Vorgänge sind meßtechnisch über Thermoelemente, IR-Meßkopf bzw. $\triangle U_{BE}$-Messung zu erfassen und auszuwerten.
Ferner sind die Einsatzmöglichkeiten von Simulationsprogrammen zu erörtern.

Ziel :

Es sind tabellarische Entscheidungshilfen zu erstellen, mit denen ein thermisch-mechanisch gesicherter Hybridaufbau entsprechend der unterschiedlichen Kundenanforderungen möglich ist.

1.1 Die Wärmeleitungsgleichungen

DGL der Ausgleichsprozesse

z.B.: - Wärmeleitung
- Diffusion
- Elektizitätsleitung

Φ : Vektor von der Größe und Richtung des Wärmeflusses

Q : Wärmemenge

T : Temperatur

t : Zeit

dV : infinitesimal kleines Volumenelement

$\mathrm{div}\Phi\ dV$ ist die Wärmemenge, die pro Zeiteinheit aus dem Volumen dV herausfließt.

Dies entspricht einer Abnahme der Wärmemenge von dV, die durch $-\delta Q/\delta t$ beschrieben werden kann

$$\Longrightarrow \quad \mathrm{div}\Phi\ dV = -\frac{\delta Q}{\delta t} \qquad (1)$$

FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Laser Mikro Elektronik
LME
FHTE Aussenstelle GP

Wärmezufuhr bewirkt Temperaturerhöhung
Wärmeabfuhr bewirkt Temperaturerniedrieg.

$$\delta Q = c \cdot \underbrace{\rho \, dV}_{dm} \, \delta T \qquad (2)$$

(2) in (1):

$$\mathrm{div}\,\Phi = - c \cdot \rho \cdot \frac{\delta T}{\delta t} \qquad (3)$$

Zusammenhang zwischen Φ und T :

$$\Phi = - \lambda \, \mathrm{grad}\, T \qquad (4)$$

Bedeutung : Wärmefluß findet in Richtung des Temperaturgefälles statt und ist proportional der Steigung des Gefälles

Proportionalitätsfaktor λ heißt
Wärmeleitfähigkeit

aus (3), (4):

$$\mathrm{div}\,\mathrm{grad}\, T = \Delta T = \frac{c \cdot \rho}{\lambda} \cdot \frac{\delta T}{\delta t} \qquad (5)$$

Laplace-Operator

Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

Berücksichtigung der Wärmezufuhr durch stationäre Wärmequellen W und der Wärmeabfuhr durch Wärmesenken V

$$c\,\rho\,\frac{\delta T}{\delta t} = \lambda\,\Delta T + W - V \qquad (6)$$

stationärer Zustand : $(\delta T/\delta t = 0)$

$$\boxed{\lambda\,\Delta T + W - V = 0} \qquad (7)$$

im kartesischen Koordinatensystem :

$$\lambda_x \frac{\delta^2 T}{\delta x^2} + \lambda_y \frac{\delta^2 T}{\delta y^2} + \lambda_z \frac{\delta^2 T}{\delta z^2} +$$

$$W(x,y,z) - V(x,y,z) = 0 \qquad (8)$$

7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto-Nr. 29775 (BLZ 610 500 00)

1.2. Wärmestrahlung und Konvektion

Strahlungsgesetz nach Stefan-Boltzmann

$$R' = \varepsilon \sigma T^4 \qquad (9)$$

ε : Emissionsgrad $\leq$ 1.0

σ : $5.67 \times 10^{-8} \frac{W}{m^2 K^4}$

==> bei der Umgebungstemperatur T_0 [K] abgegebene Wärmemenge eines Körpers mit der Temperatur T [K]

$$R = \varepsilon \sigma [T^4 - T_0^4] \, dA \qquad (10)$$

Wärmeübertragung durch Konvektion

$$K = \alpha \, T^p \, (T - T_0) \, dA \qquad (11)$$

$$\alpha \approx 0.3 \ldots 2.0 \quad \frac{W}{m^2 K^{1+p}}$$

$$p \approx 1/3 \ldots 1/4$$

α ist kein reiner Stoffwert, sondern auch von der Viskosität des Gases oder der Flüssigkeit, von den Strömungsverhältn., von der Oberflächenbeschaffenheit, etc. abhängig.

insgesamt : $V = R + K \qquad (12)$

$$V = \left[\varepsilon \, \sigma \, [\, T^4 - T_0^4 \,] + \alpha \, T^p \, (T - T_0) \right] dA \qquad (13)$$

nichtlinear !

1.3. Das Wärmeleitungsmodell zur Herleitung der partiellen Differenzengleichungen

DGL der Wärmeleitung :

$$\lambda_x \frac{\delta^2 T}{\delta x^2} + \lambda_y \frac{\delta^2 T}{\delta y^2} + \lambda_z \frac{\delta^2 T}{\delta z^2} +$$

$$W(x,y,z) - V(x,y,z) = 0 \qquad (8)$$

Übergang zu endlich kleinen Volumenelementen $\Delta V = \Delta x\, \Delta y\, \Delta z$ durch Diskretisierung der Variablen x, y und z.

==> Differenzengleichung der Wärmeleitung

$$\lambda_x \frac{\Delta^2 T}{\Delta x^2} + \lambda_y \frac{\Delta^2 T}{\Delta y^2} + \lambda_z \frac{\Delta^2 T}{\Delta z^2} +$$

$$W(x,y,z) - V(x,y,z) = 0 \qquad (14)$$

$$T = f(i\Delta x,\ j\Delta y,\ k\Delta z) = T_{i,j,k} \qquad (15)$$

$$i,j,k \quad \text{ganz}$$

FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

Beispiele für die Konstanten λ und ε :

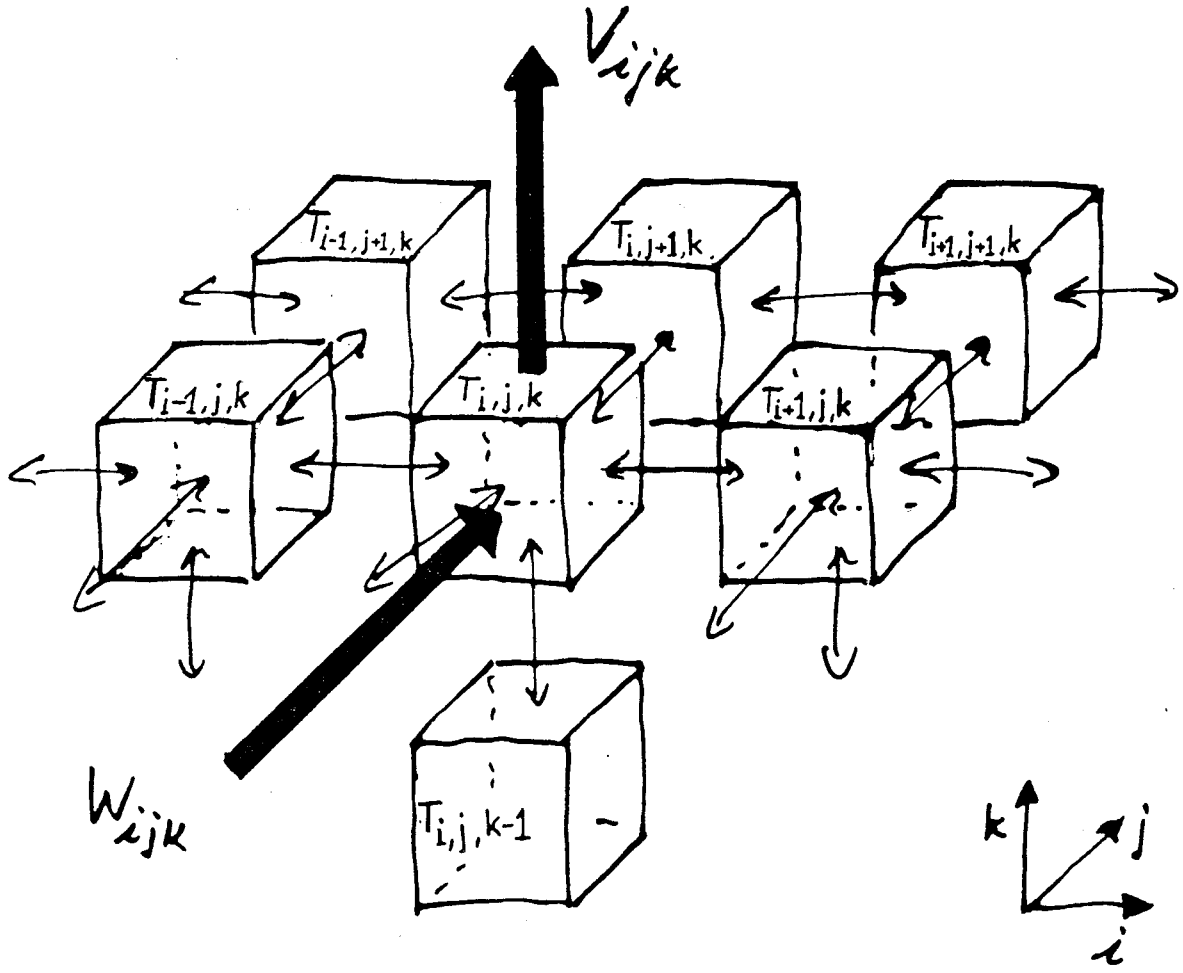
Material	Wärmeleitfähigkeit λ in W/(mK)	Emissionsgrad ε
Aluminium		
poliert	220	0.04
oxidiert	220	0.2-0.5
Kupfer		
poliert	390	0.03
oxidiert	390	0.6-0.8
Stahl	45	0.2-0.65
Gold	310	0.02
Silber	430	0.02
Glas	$\approx$ 1	0.94
96% Al_2O_3	35	>0.8
99.5% Al_2O_3	37	>0.8
Beryllium	210	-

7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto Nr. 29775 (BLZ 610 500 00)

FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

Betrachtung eines Volumenelements ΔV an der Oberfläche eines Körpers



Volumenelemente ΔV mit unendlich großer Wärmeleitfähigkeit sind voneinander durch unendlich dünne Schichten mit

Die 2.Ableitung $f''(x) = \frac{d}{dx} f'(x) = \frac{d^2 f(x)}{dx^2}$

$$\Longrightarrow f''(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} \qquad (16)$$

$$\Longrightarrow \lambda_x \frac{\Delta^2 T}{\Delta x^2} = \lambda_x \frac{T_{i+1,j,k} - 2T_{i,j,k} + T_{i-1,j,k}}{\Delta x^2} \qquad (17)$$

Für Volumenelemente ΔV mit den Koordinaten $i\Delta x$, $j\Delta y$, $k\Delta z$ gilt :

$$\left[T_{i-1,j,k} - T_{i,j,k}\right]\frac{\lambda_x}{\Delta x^2} + \left[T_{i+1,j,k} - T_{i,j,k}\right]\frac{\lambda_x}{\Delta x^2}$$

$$+ \left[T_{i,j-1,k} - T_{i,j,k}\right]\frac{\lambda_y}{\Delta y^2} + \left[T_{i,j+1,k} - T_{i,j,k}\right]\frac{\lambda_y}{\Delta y^2}$$

$$+ \left[T_{i,j,k-1} - T_{i,j,k}\right]\frac{\lambda_z}{\Delta z^2} + 0$$

$$+ W_{i,j,k} - V_{i,j,k}(T_{i,j,k}) = 0 \qquad (18)$$

Jedes Volumenelement liefert eine Gleichung

$$\begin{bmatrix} \ddots & & \\ & \ddots & \\ & & \ddots \end{bmatrix} \cdot \begin{bmatrix} T_{111} \\ \vdots \\ T_{ijk} \\ \vdots \end{bmatrix} + \begin{bmatrix} W_{111} \\ \vdots \\ W_{ijk} \\ \vdots \end{bmatrix} - \begin{bmatrix} V_{111}(T) \\ \vdots \\ V_{ijk}(T) \\ \vdots \end{bmatrix} = 0 \qquad (19)$$

H — Wärme-zufuhr — Wärme-abfuhr

Beispiel: Bei Unterteilung eines Würfels in 10x10x10 Volumenelemente ergibt sich ein nichtlineares Gleichungssystem 1000. Ordnung.

Ebenes (zweidimensionales) Problem mit 3x4=12 Flächenelementen

31	32	33	34
21	22	23	24
11	12	13	14

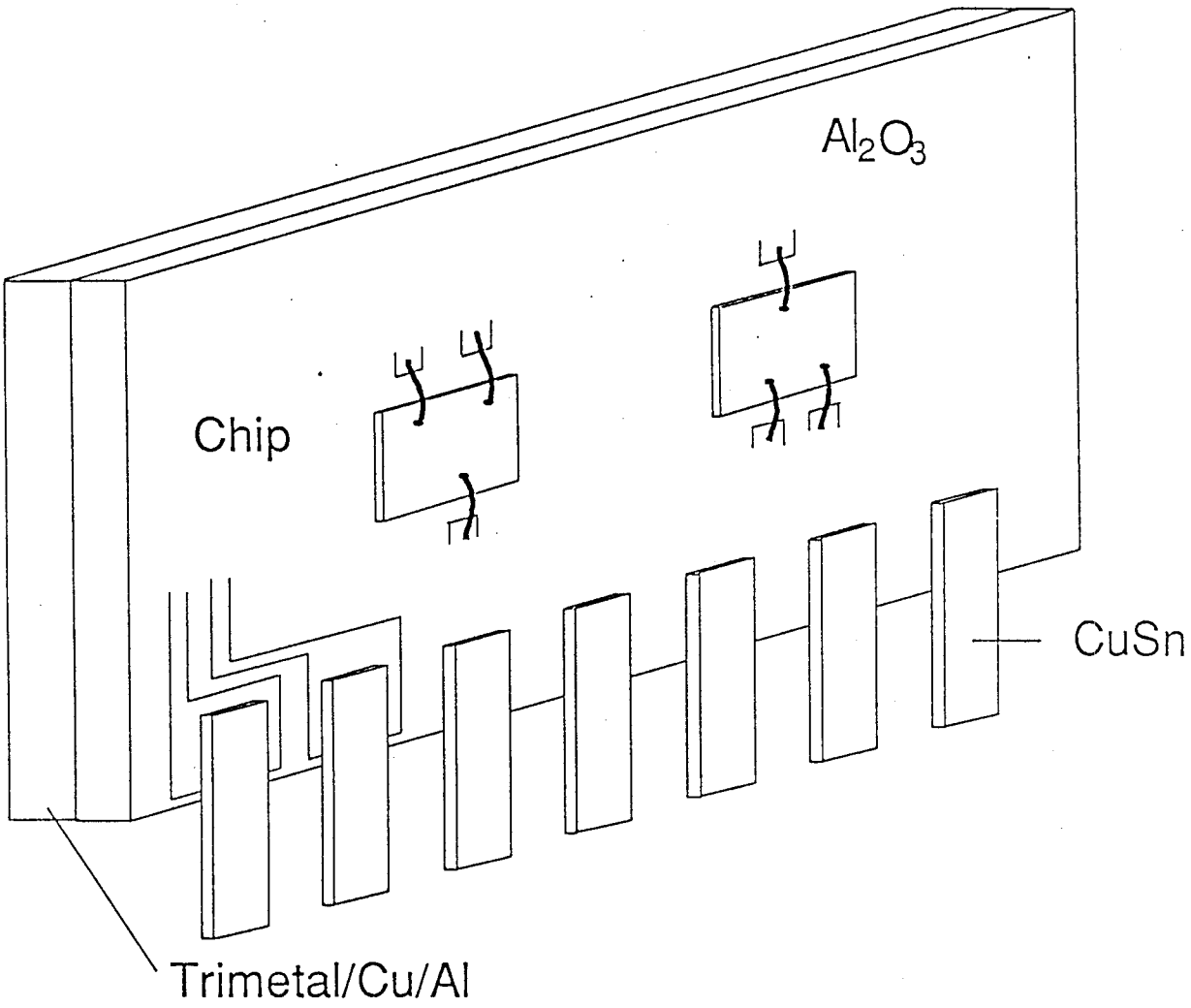
$y = j \cdot \Delta y \longrightarrow x = i \cdot \Delta x$

Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Laser Mikro Elektronik
LME
FHTE Aussenstelle GP

2. Aufbautechniken

2.1. STANDARDTECHNIK

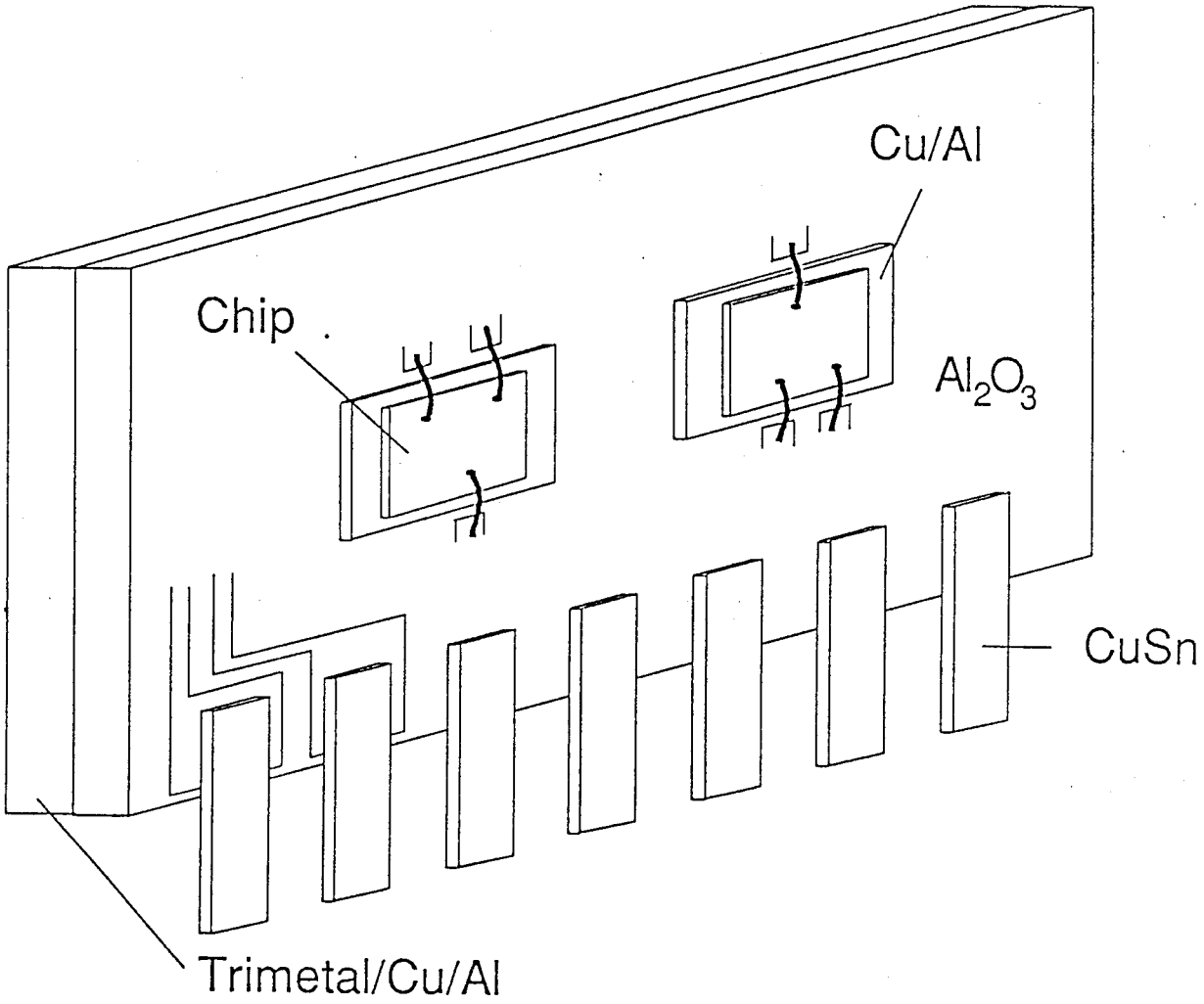


7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto-Nr. 29775 (BLZ 610 500 00)

Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

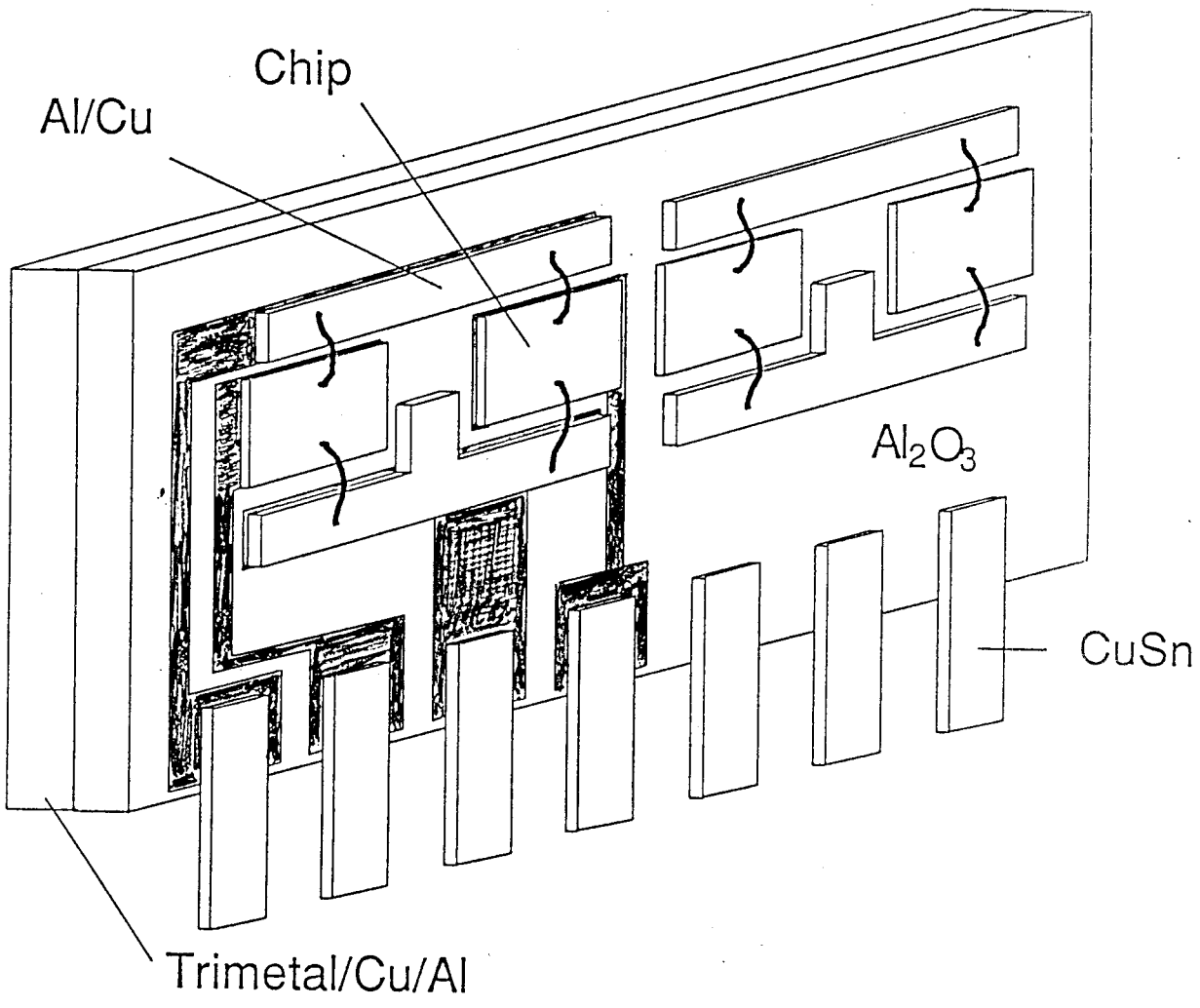
2.2 SANDWICHTECHNIK



FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

2.3 RIEGELTECHNIK

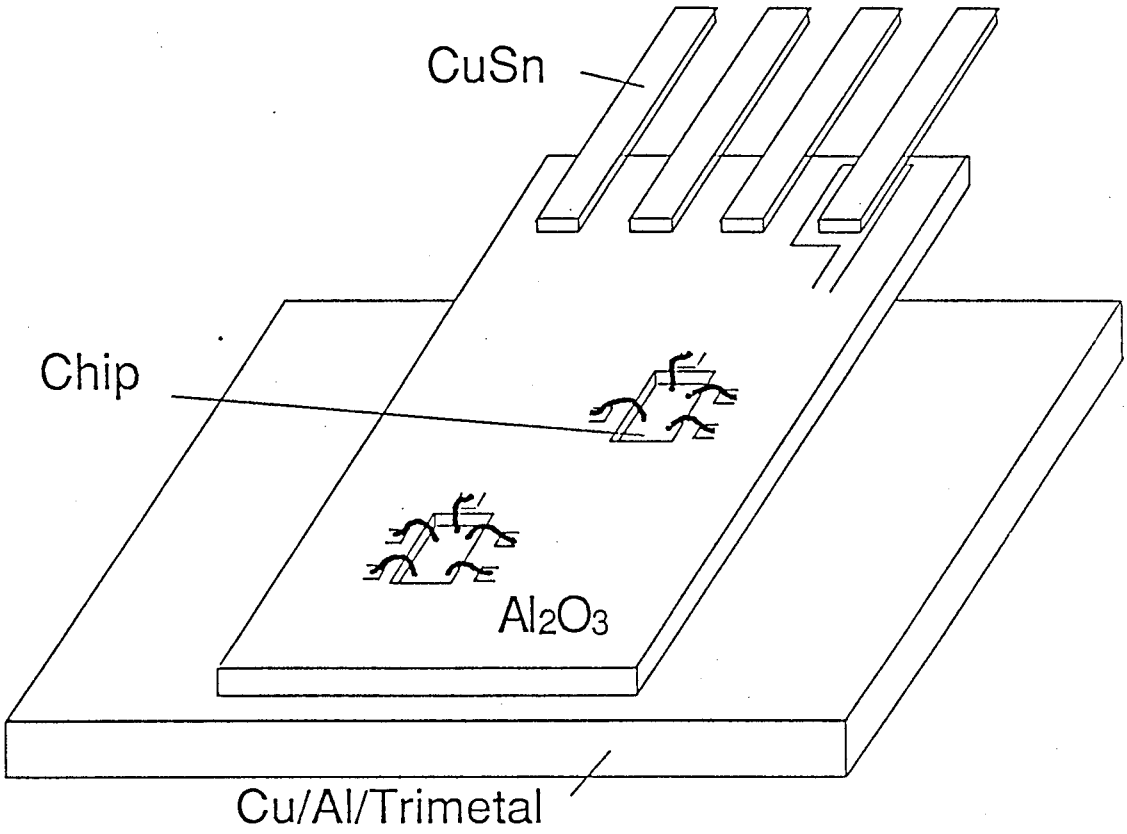


7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto-Nr. 29775 (BLZ 610 500 00)

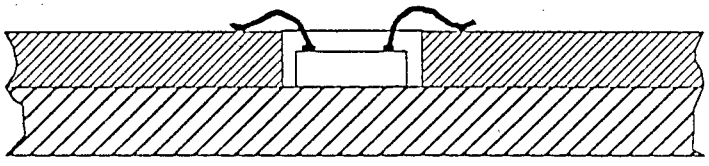
FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Laser Mikro Elektronik
LME
FHTE Aussenstelle GP

2.4 FENSTERTECHNIK



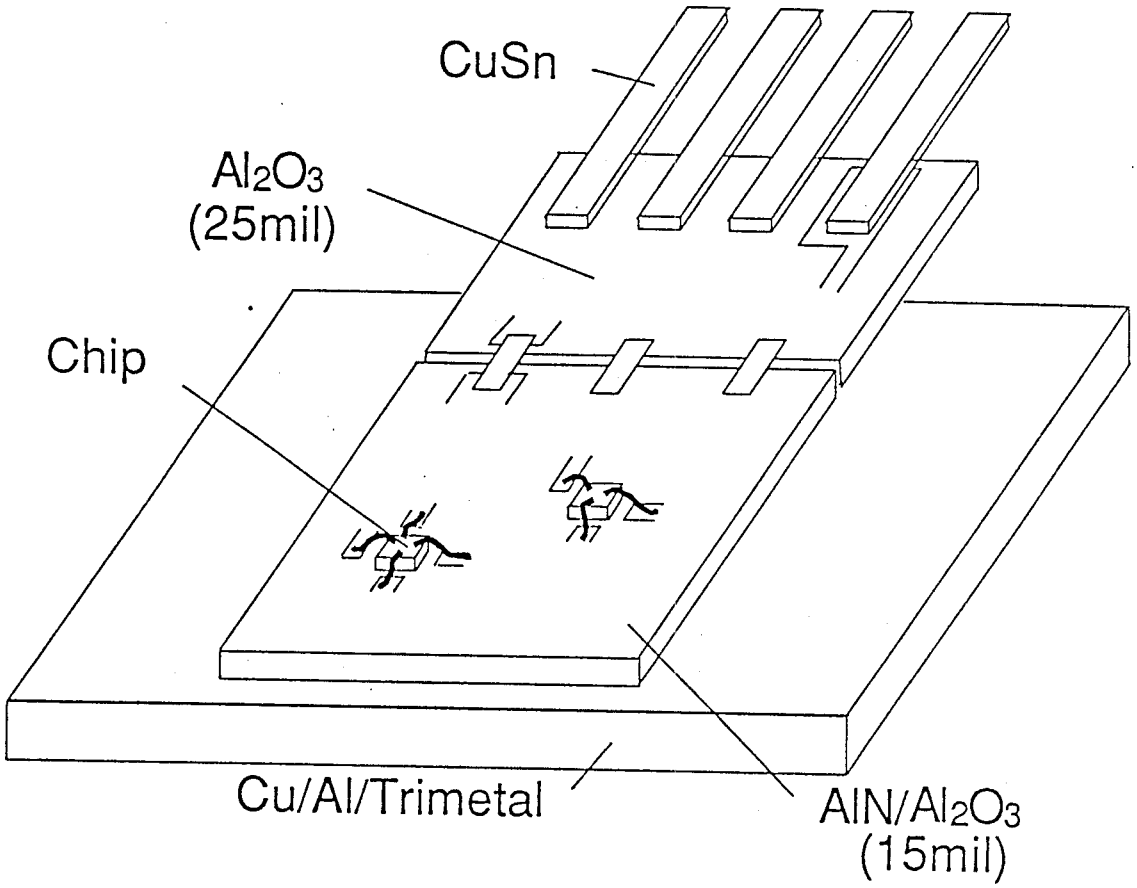
Schnitt



FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Laser Mikro Elektronik
LME
FHTE Aussenstelle GP

2.5 MEHRFACHSUBSTRATTECHNIK



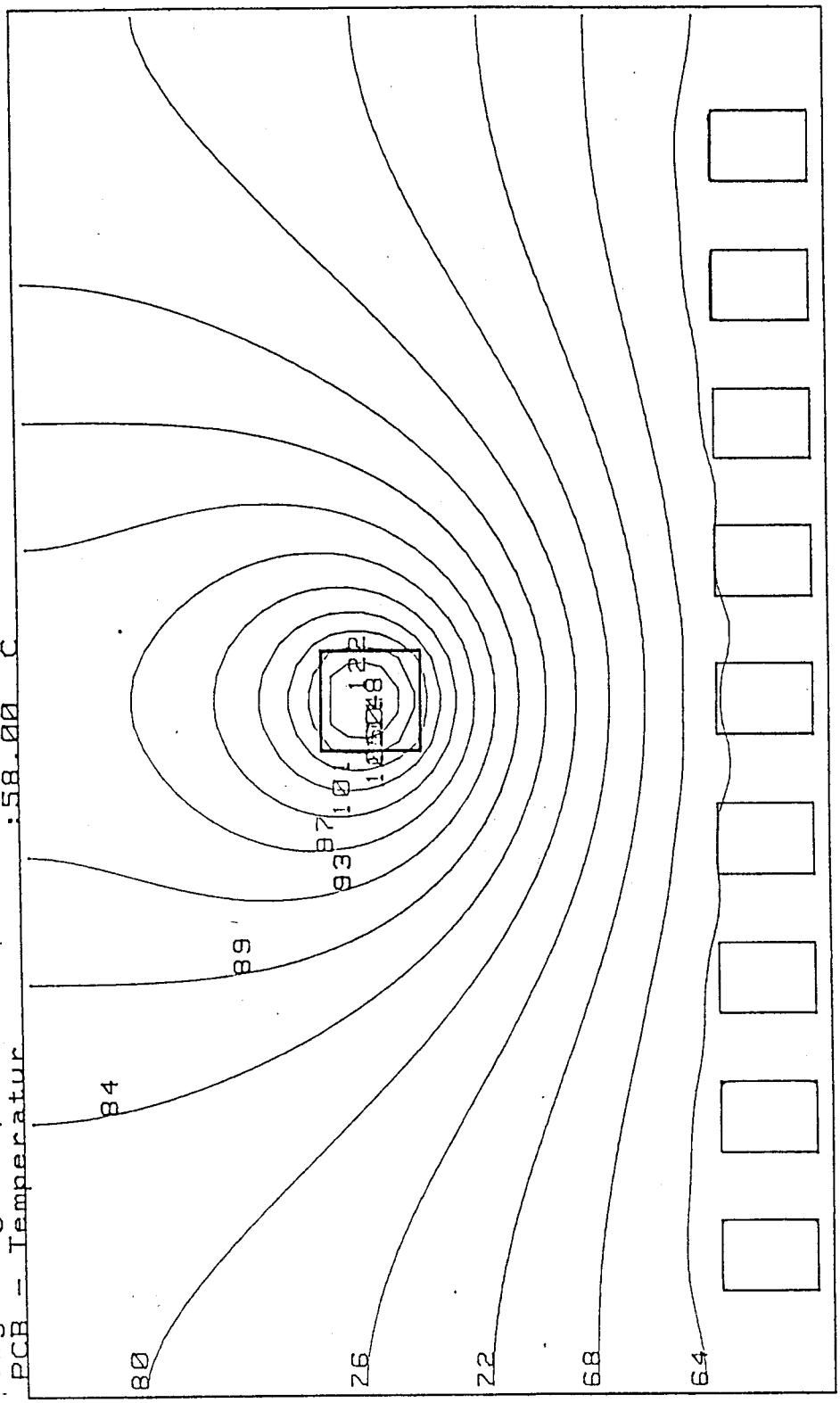
7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto-Nr. 29775 (BLZ 610 500 00)

3. Simulation der Testobjekte

3.1

Temperatursimulation :V11
V11. Substrat: Al2O3 0.635 Chip: 1.83*1.83 mm^2
Gesamtverlustleistung :1750.00 mW

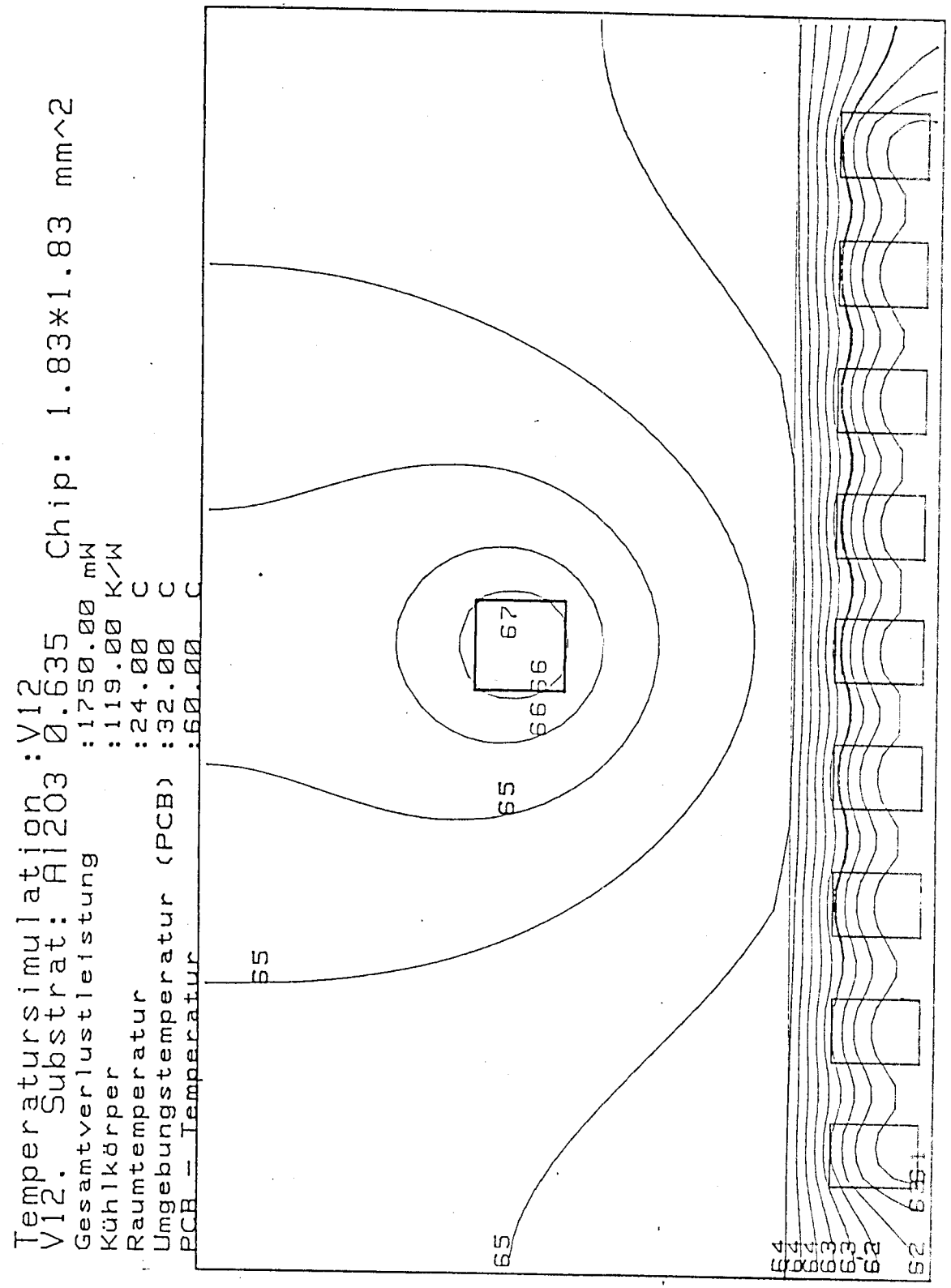
Raumtemperatur :24.00 C
Umgebungstemperatur (PCB) :31.00 C
PCB - Temperatur :58.00 C



Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

3.2

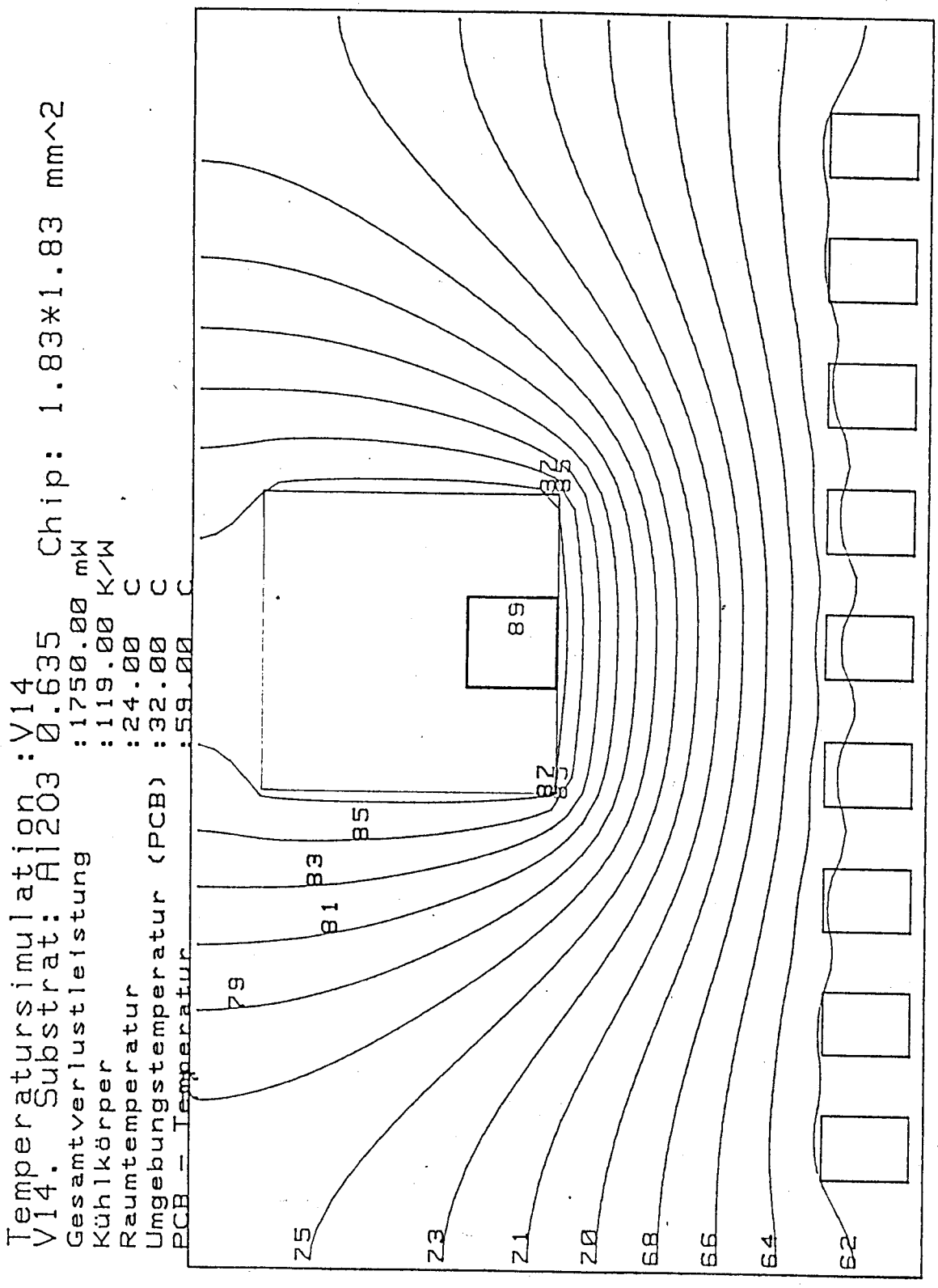


7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto-Nr. 29775 (BLZ 610 500 00)

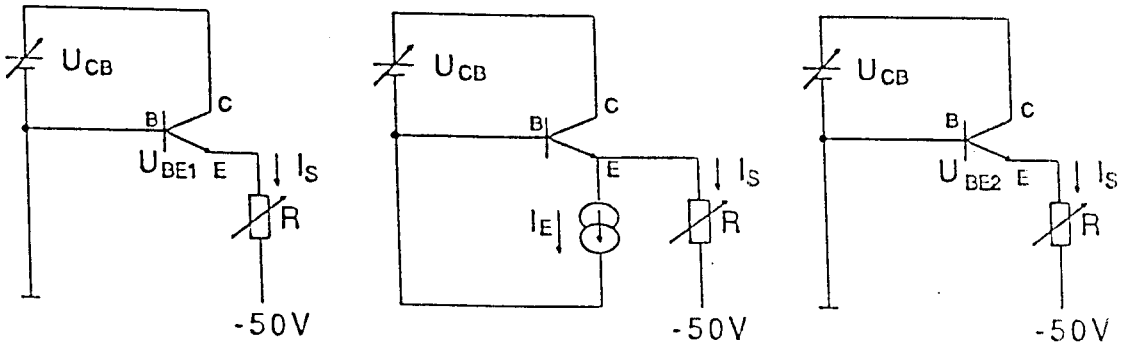
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

3.3



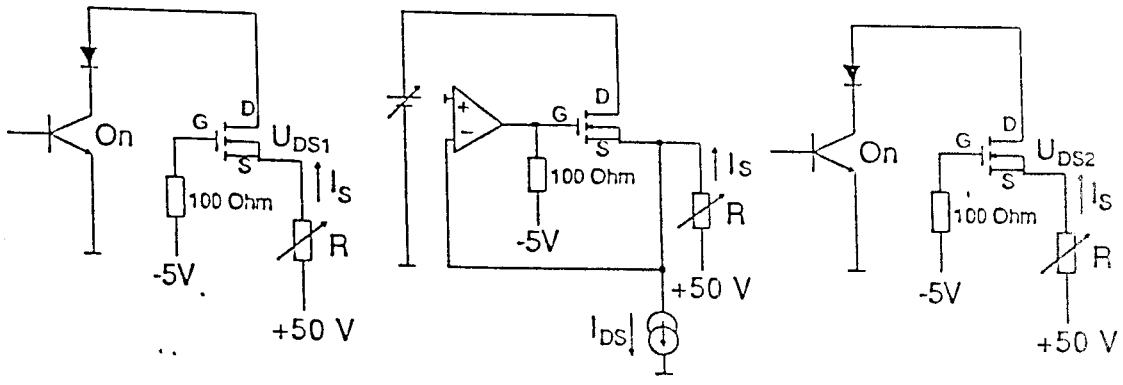
4. Messung der Testobjekte



U_{BE1} POWER U_{BE2}

$\triangle U_{BE} = U_{BE1} - U_{BE2}$

Meßprinzip für Bipolartransistoren

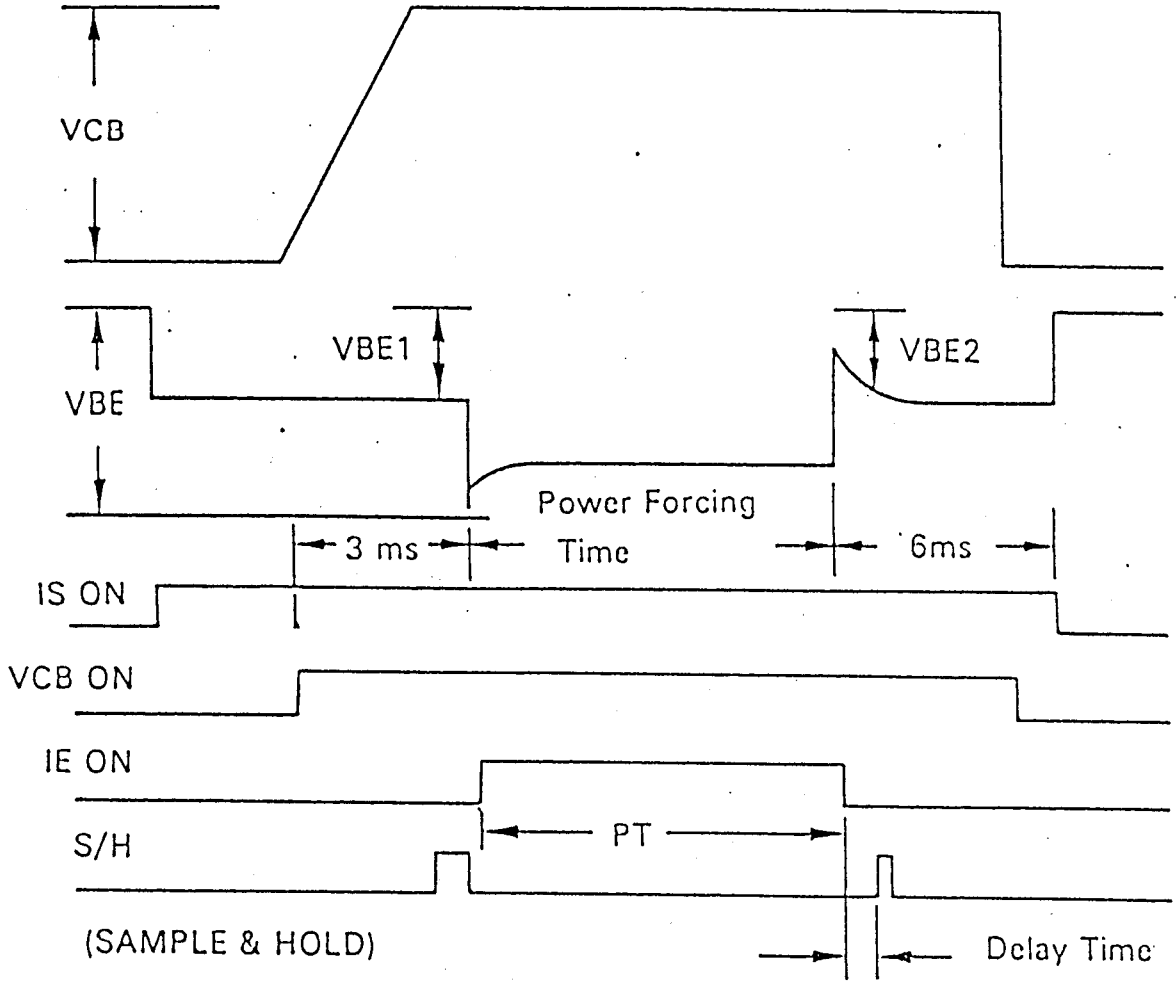


U_{DS1} POWER U_{DS2}

$\triangle U_{DS} = U_{DS1} - U_{DS2}$

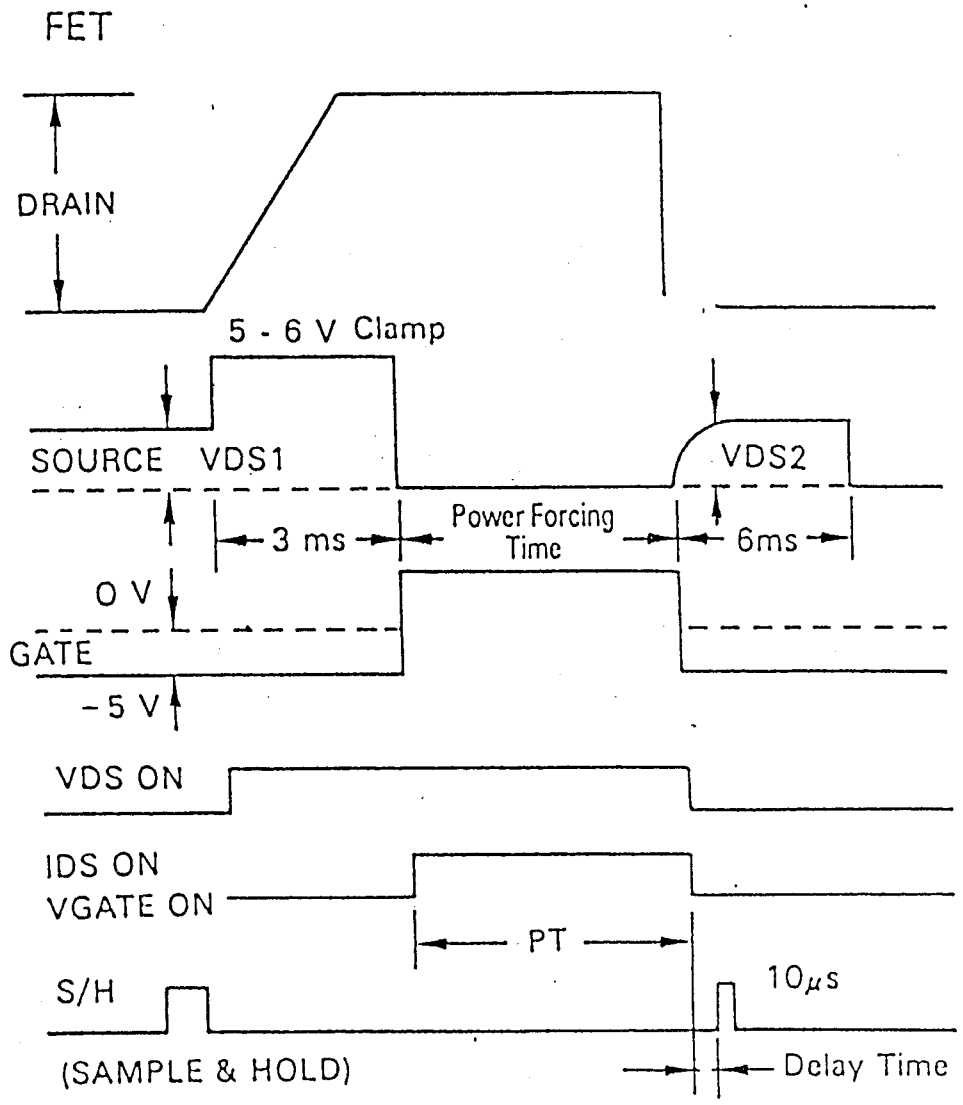
Meßprinzip für Unipolartransistoren

Bipolar Transistor



Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP



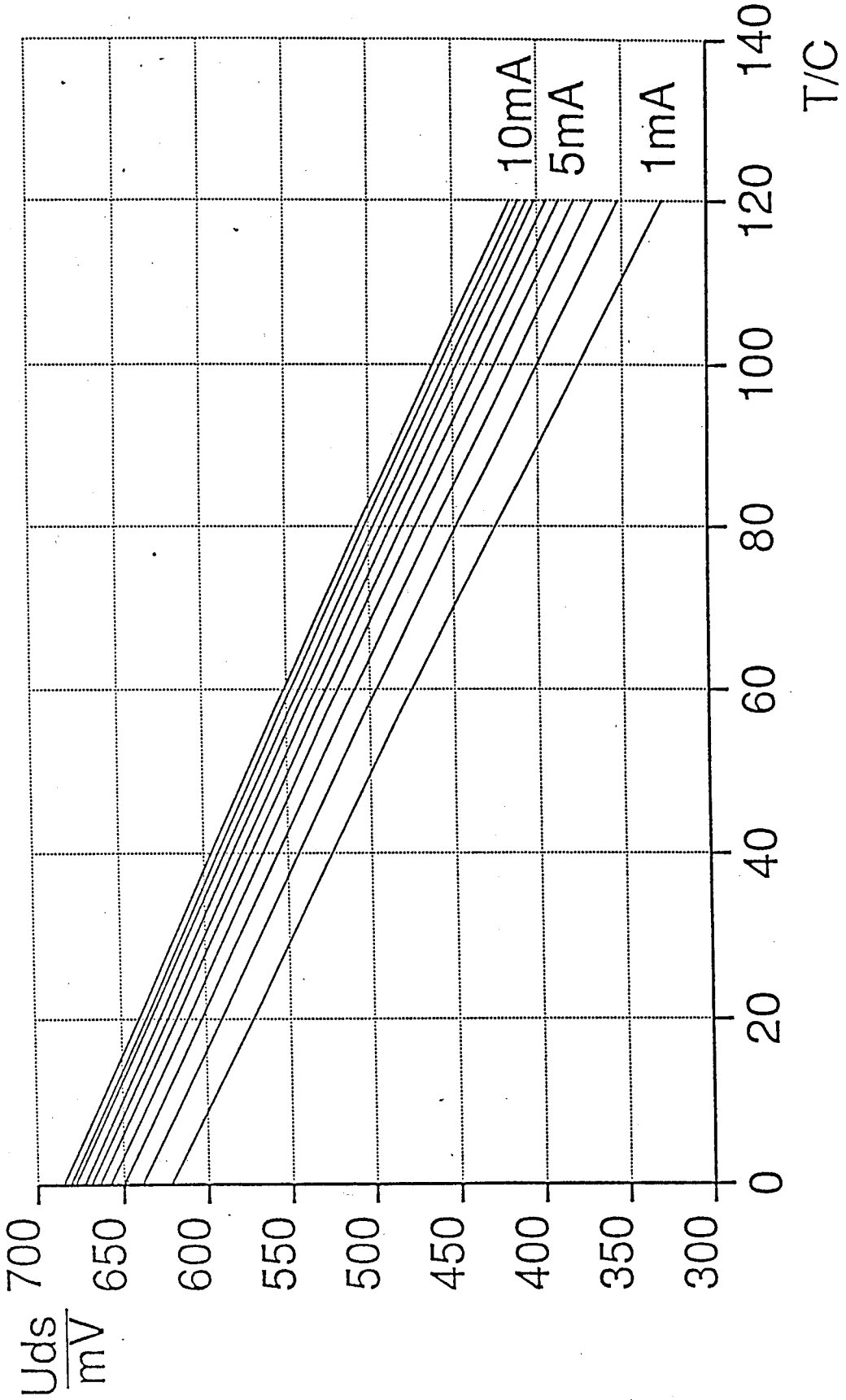
7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto-Nr. 29775 (BLZ 610 500 00)

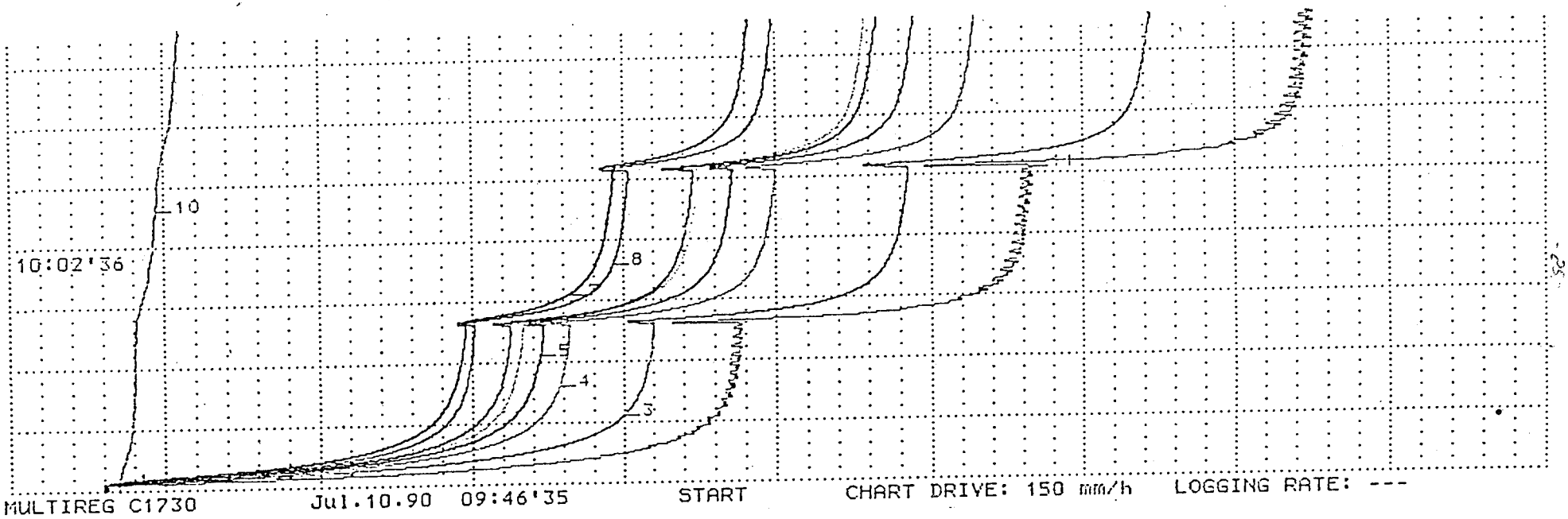
FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

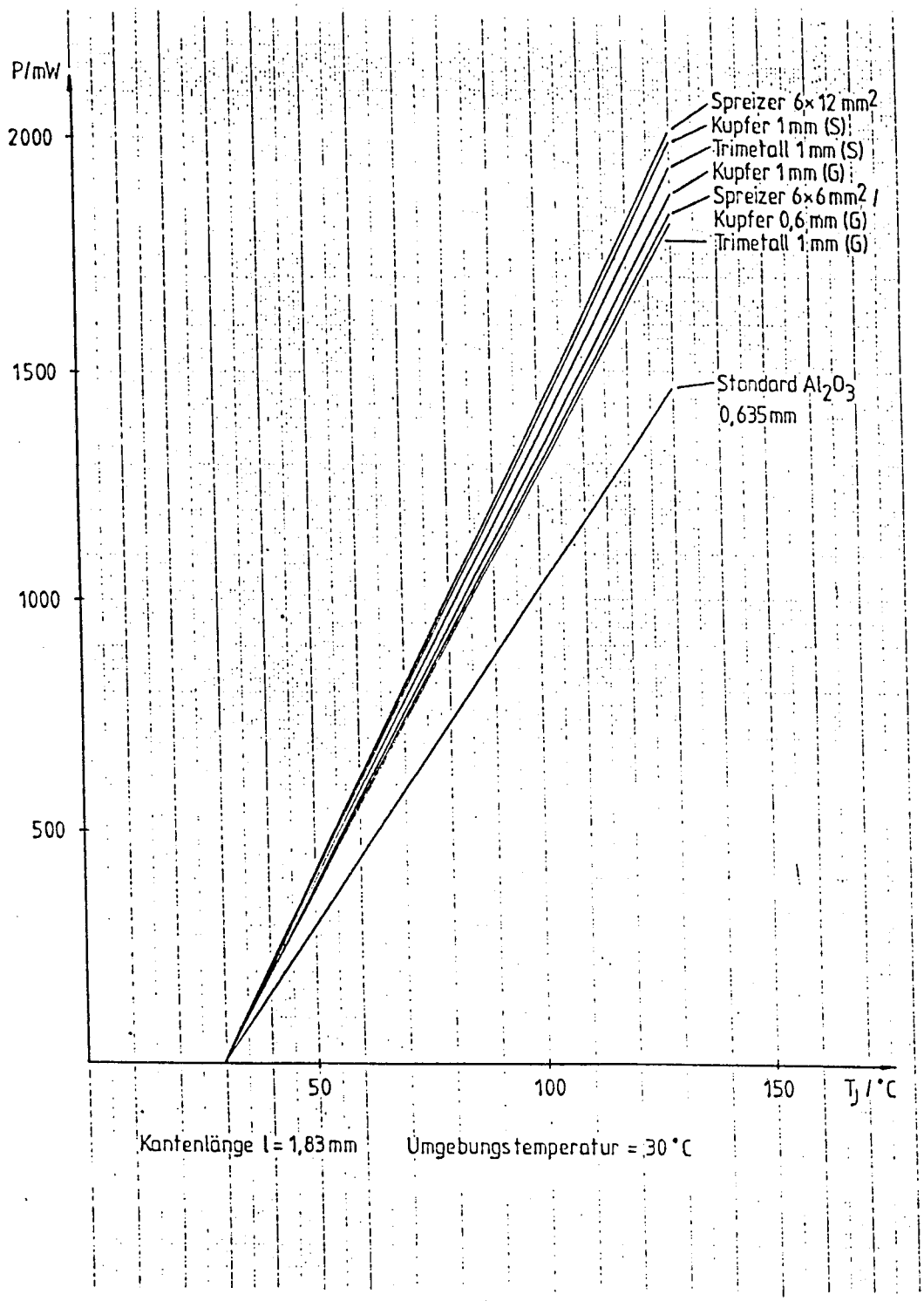
Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

Transistor BS97

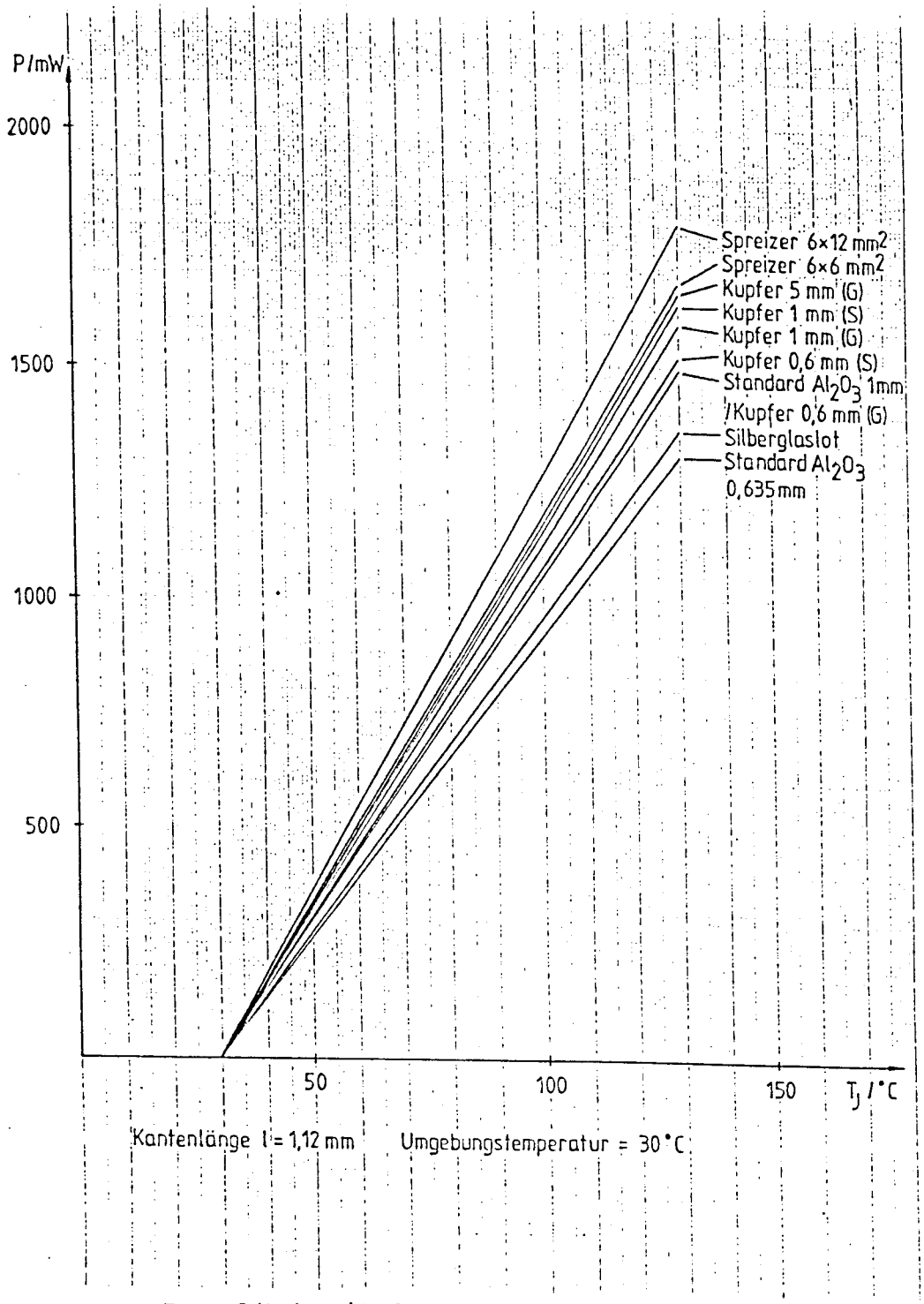
Temperaturkoeffizient



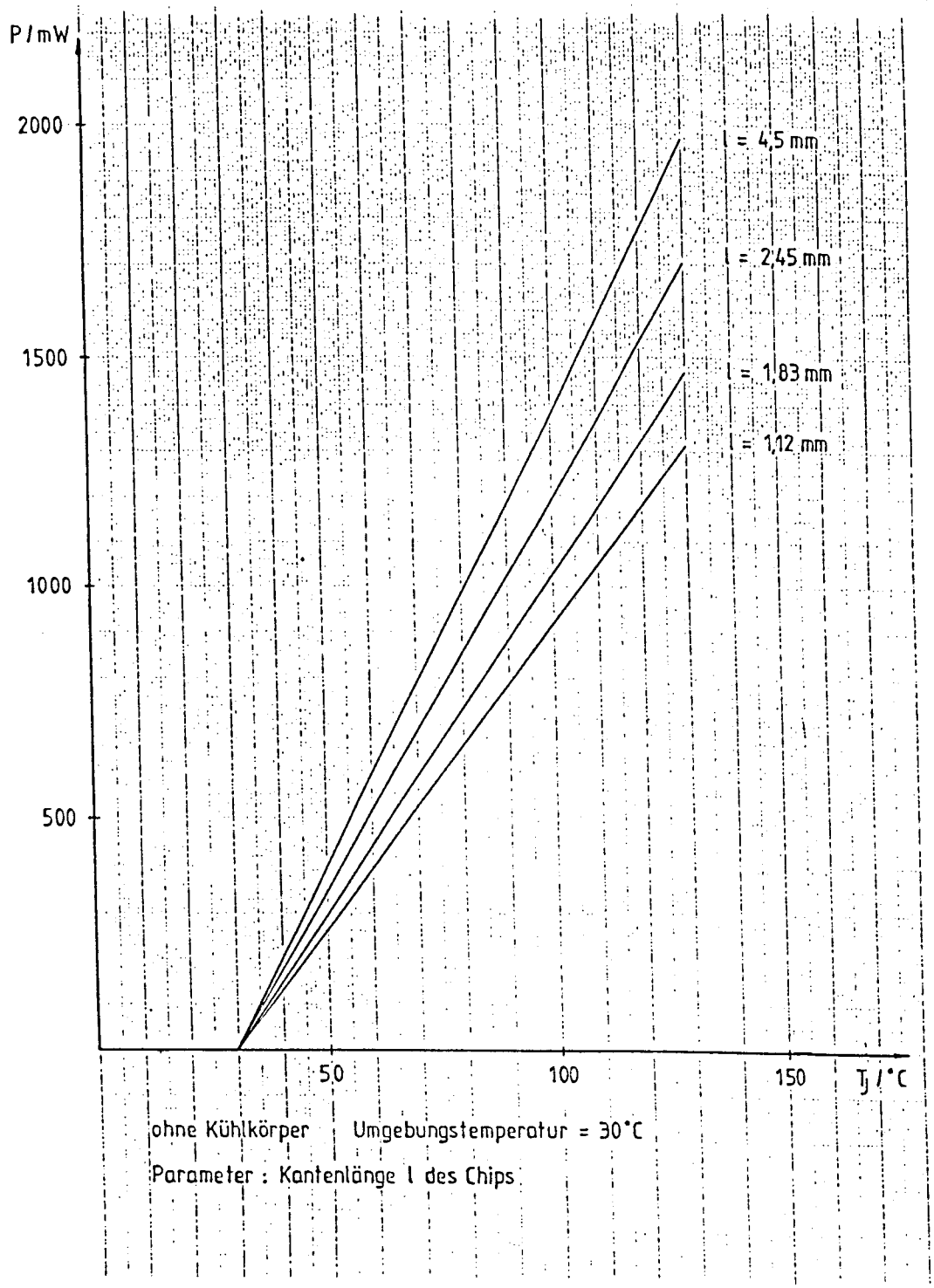

10
10:02'36
8
5
4
3
MULTIREG C1730
Jul.10.90 09:46'35
START
CHART DRIVE: 150 mm/h
LOGGING RATE: ---



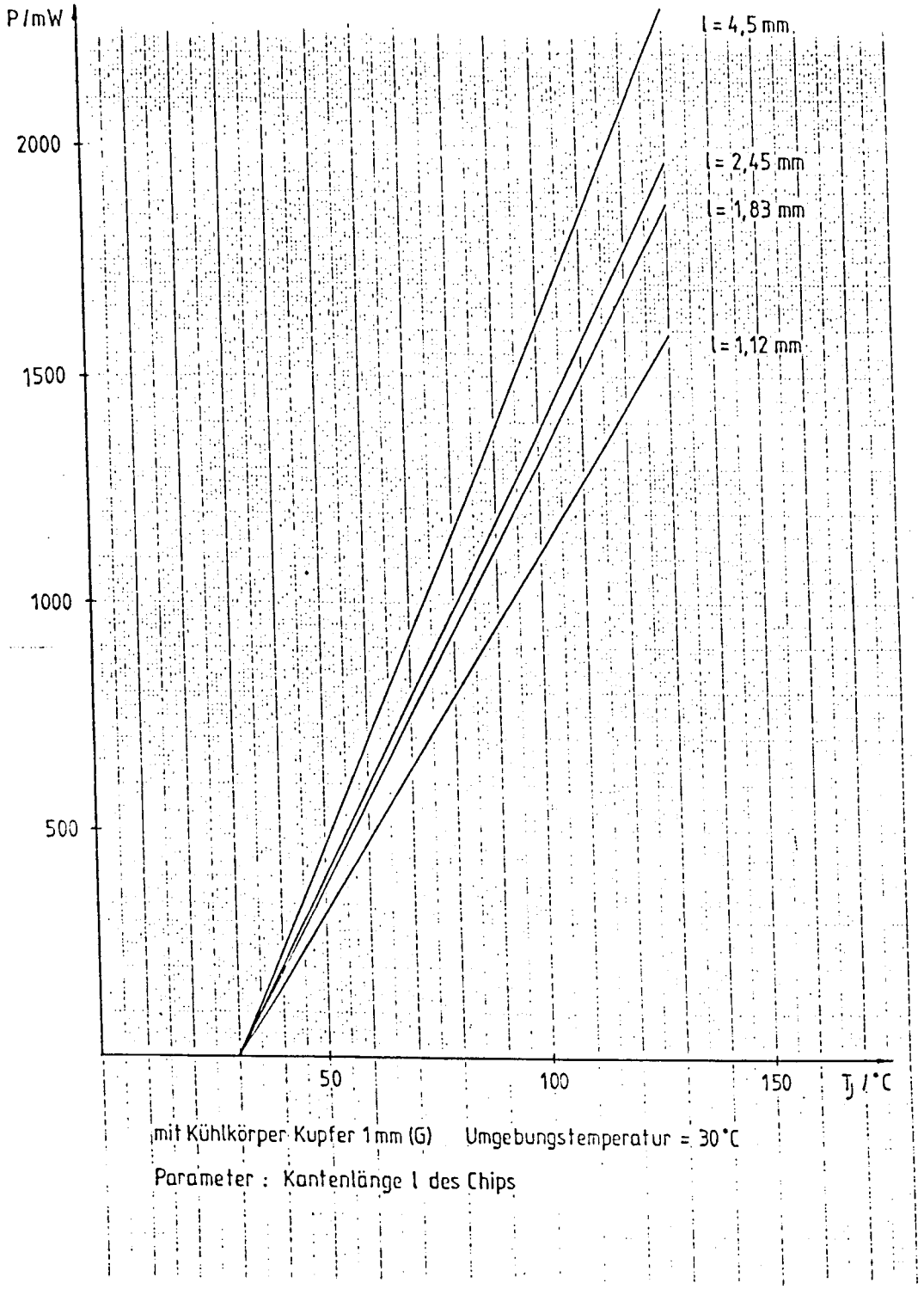
P = f(T_J) mit l = 1.83 mm



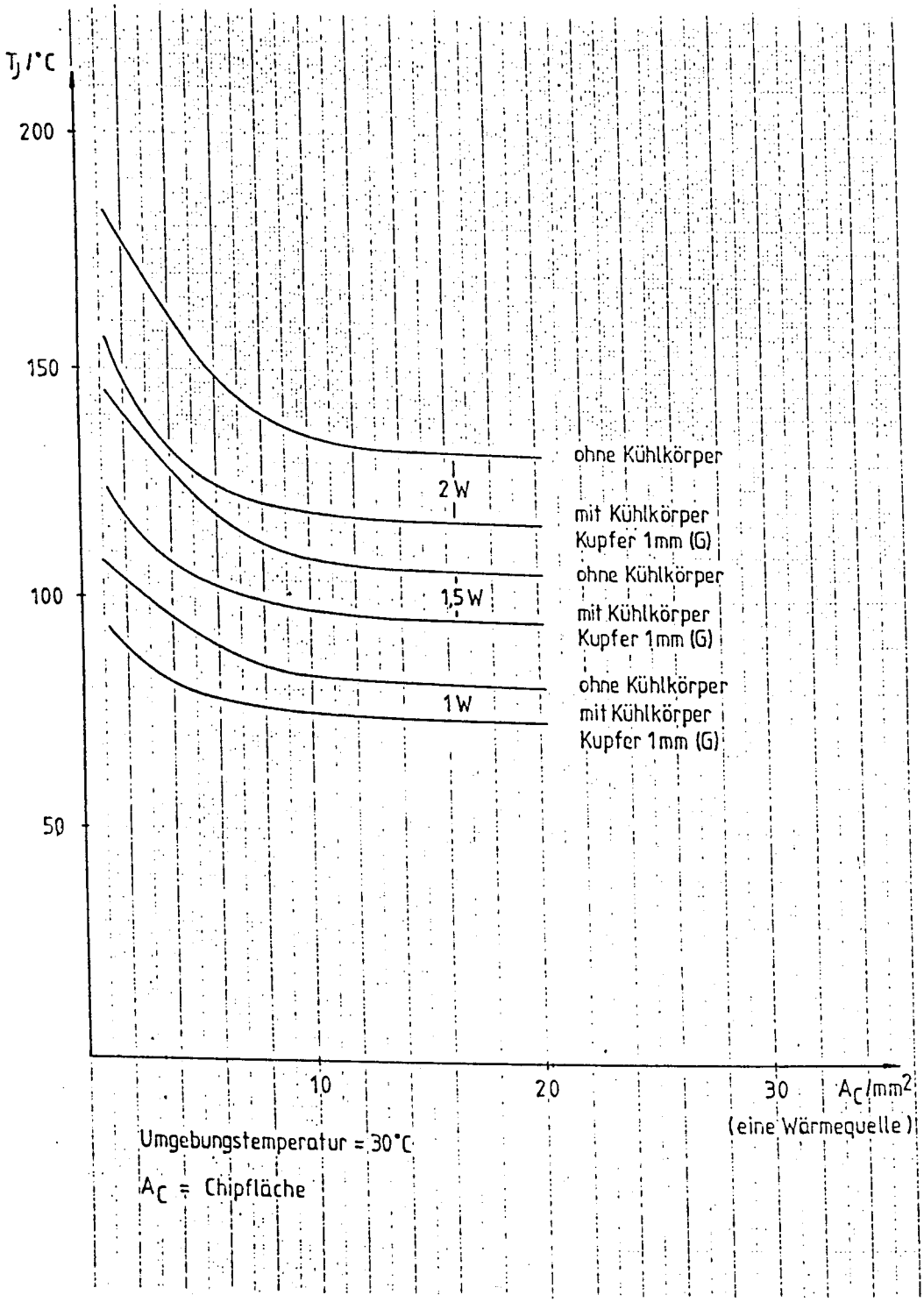
$P = f(T_J)$ mit l = 1.12 mm



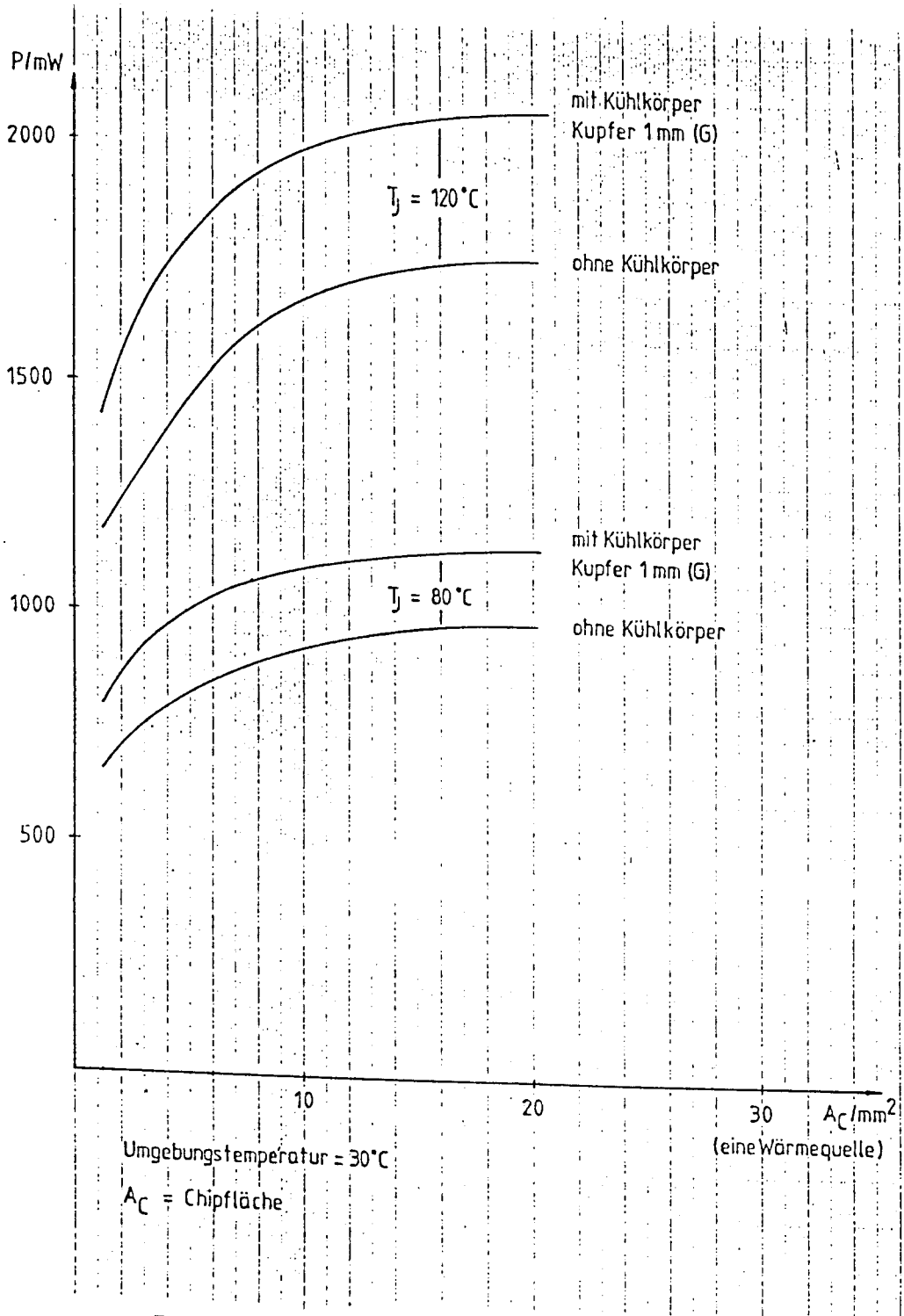
$P = f(T_j)$ mit Parameter Kantenlänge



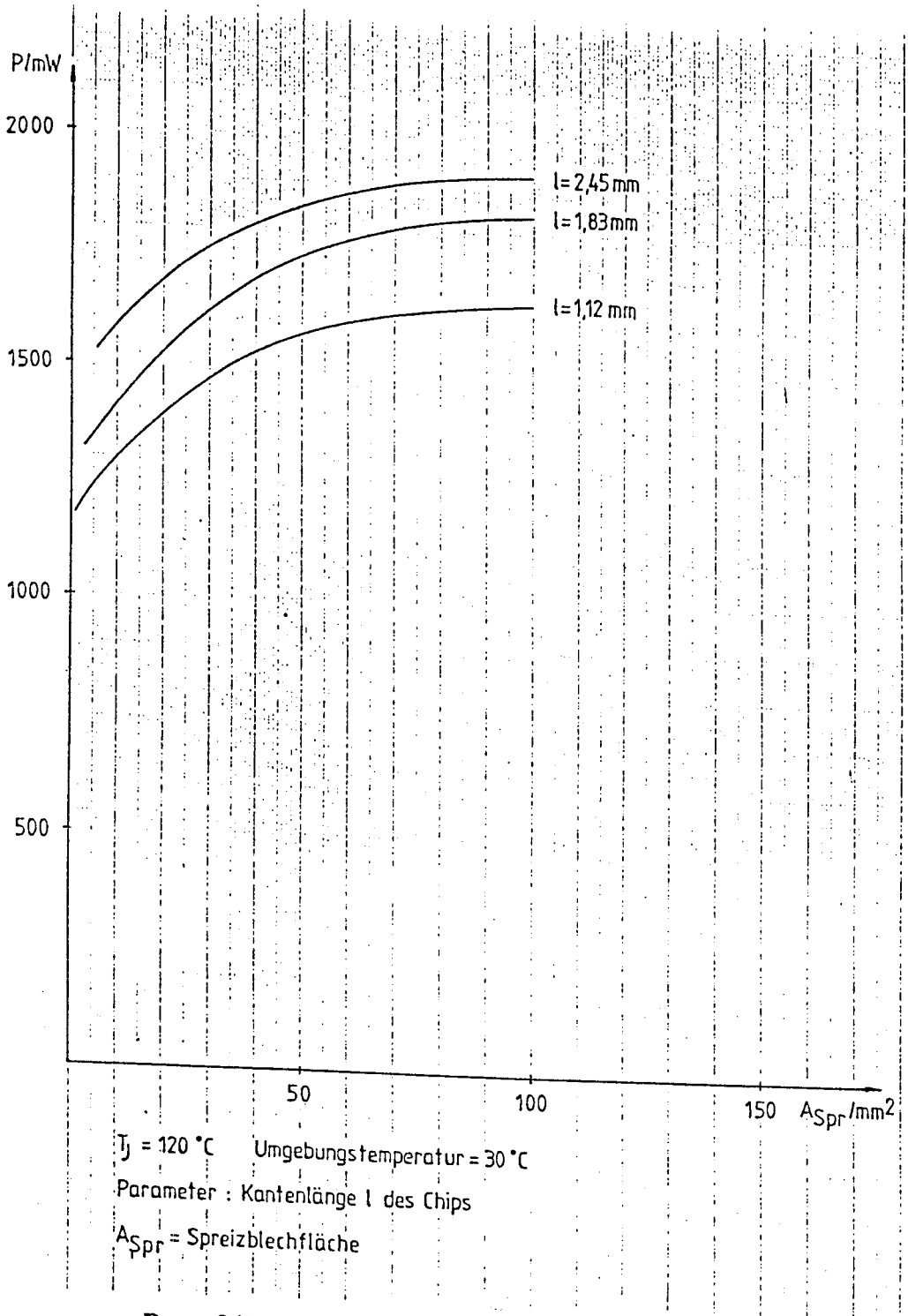
P = f(T_j) mit Parameter Kantenlänge


T_J / °C
200
150
100
50
2 W
1,5 W
1 W
ohne Kühlkörper
mit Kühlkörper
Kupfer 1mm (G)
ohne Kühlkörper
mit Kühlkörper
Kupfer 1mm (G)
ohne Kühlkörper
mit Kühlkörper
Kupfer 1mm (G)
10
20
30
A_C / mm²
(eine Wärmequelle)
Umgebungstemperatur = 30°C
A_C = Chipfläche

$T_J = f(A_C)$


P/mW
2000
1500
1000
500
mit Kühlkörper
Kupfer 1 mm (G)
T_j = 120 °C
ohne Kühlkörper
mit Kühlkörper
Kupfer 1 mm (G)
T_j = 80 °C
ohne Kühlkörper
10
20
30
A_C/mm²
(eine Wärmequelle)
Umgebungstemperatur = 30°C
A_C = Chipfläche

$P = f(A_c)$

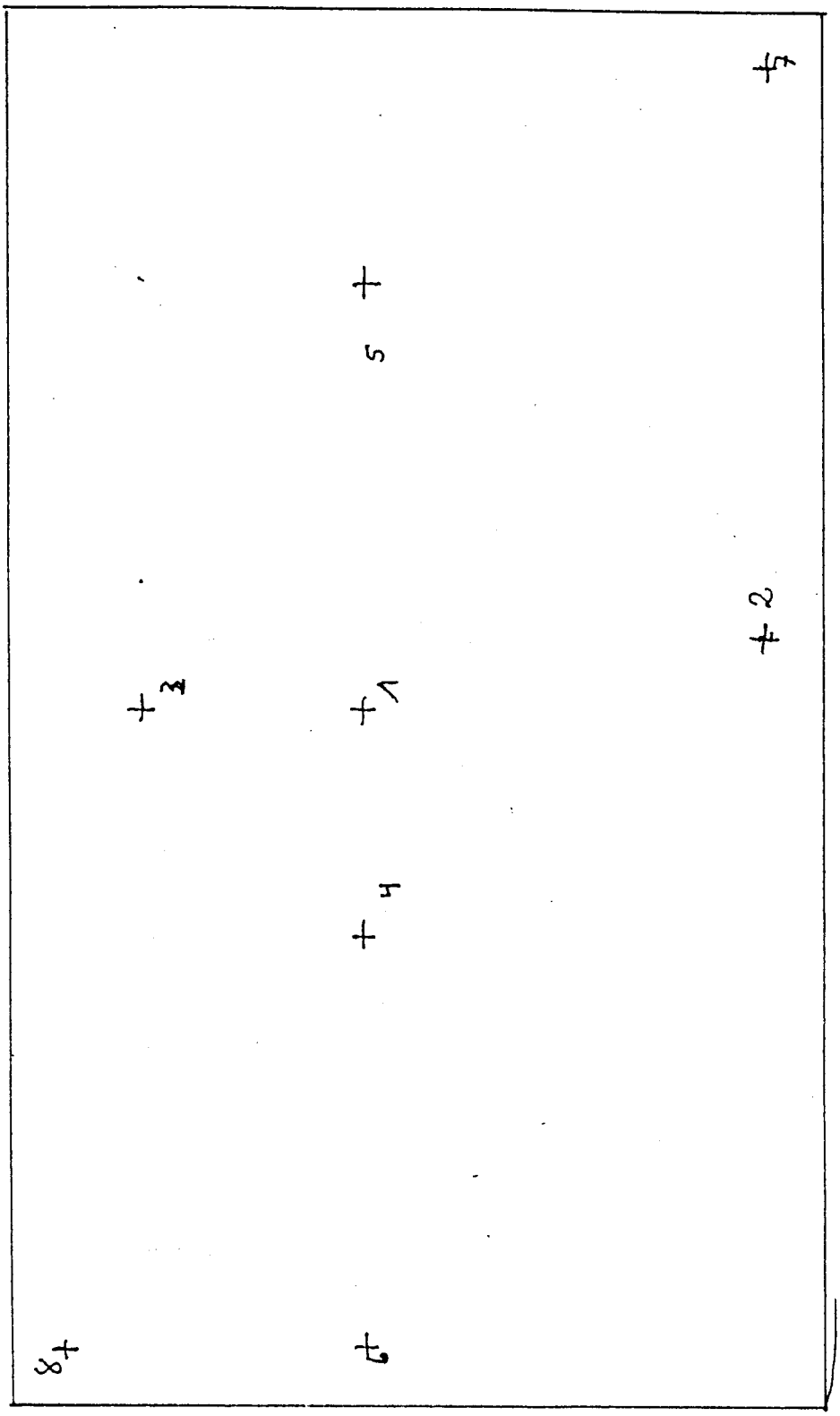

P/mW
2000
1500
1000
500
l = 2,45 mm
l = 1,83 mm
l = 1,12 mm
50
100
150
A_{Spr}/mm²
T_j = 120 °C
Umgebungstemperatur = 30 °C
Parameter : Kantenlänge l des Chips
A_{Spr} = Spreizblechfläche

$P = f(A_{sp})$

FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

5.1 Mess- und Simulationspunkte



7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177
Bankverbindung: Kreissparkasse Göppingen Konto-Nr. 29775 (BLZ 610 500 00)

Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

5. VERGLEICH

zwischen Simulation und Messung

Schaltung V11 : Standardaufbau
Ohne Kühlkörper
Chipgröße : 1,83 mm Kantenlänge

T/°C	Tj	T1	T2	T3	T4	T5	T6	T7	T8
TSIM		122	64	98	89	79	76	64	80
Autotherm	138	113	64	99	87	78	75	61	81
Messung	146	80	79	92	84	78	76	70	69

FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

VERGLEICH

zwischen Simulation und Messung

Schaltung V12 : Standardaufbau
Mit Kühlkörper Cu 1mm
Chipgröße : 1,83 mm Kantenlänge

T/°C	Tj	T1	T2	T3	T4	T5	T6	T7	T8
TSIM		67	62	65	65	65	65	62	65
Autotherm	124	62	60	65	63	62	61	60	62
Messung	119	73	80	89	80	79	79	76	75

7320 Göppingen Robert-Bosch-Straße 1 Telefon (07161) 679 - 0, 175 Telefax (07161) 679 - 177

FHTE
Fachhochschule für Technik Esslingen
Außenstelle Göppingen

Labor Mikro Elektronik
LME
FHTE Aussenstelle GP

VERGLEICH

zwischen Simulation und Messung

Schaltung V14 : Sandwichaufbau

Spreizblech Cu 1mm dick, 6*6mm^2

Chipgröße : 1,83 mm Kantenlänge

T/°C	Tj	T1	T2	T3	T4	T5	T6	T7	T8
TSIM		89	62	89	83	75	72	62	76
Autotherm	125	101	61	96	90	79	79	61	83
Messung	120	75	78	99	85	80	76	70	71

6. Zusammenfassung

Der Aufbau von Hybridschaltungen erfordert in der heutigen Zeit immer mehr mikromechanisches und physikalisches "Know-How". Das gilt besonders für den Bereich der Leistungshybride, da die Hochintegration elektrischer Funktionen hier schnell voranschreitet. Die somit erhöhte Packungsdichte hat aber auch zur Folge, daß Verlustleistungen stark ansteigen. Die dabei entstehende störende Wärme soll durch entsprechende Kühlmaßnahmen abgeführt werden, bei möglichst geringen Herstellkosten. Bis jetzt existieren hauptsächlich individuelle Lösungen für kundenspezifische Anforderungen. Somit waren bisher separate Testaufbauten notwendig. Das Ziel dieser Arbeit ist, aus einer Vielzahl von Aufbauvarianten einige technisch bevorzugte auszuwählen und diese im Hinblick auf verschiedene Kombinationen von Materialien und deren Anordnungen zu untersuchen. Diese Untersuchungen wurden mit Hilfe von Simulationen und einer Meßeinrichtung ausgeführt, um Sperrschichttemperaturen und Wärmeverteilungen zu ermitteln. Bei den Simulationsprogrammen handelt es sich um ein firmeninternes, spezielles Programm für Schichtschaltungen und ein konventionelles einer Softwarefirma. Das Resultat von Simulationsprogrammen ist die Verteilung der Temperatur auf dem Substrat, geeignet für entsprechende Kühl- und Aufbaumaßnahmen. Mit Hilfe von Messungen wurden die Ergebnisse verglichen und deren Aussagen erhärtet. Dabei wurde festgestellt, daß durch die Verwendung eines Spreizbleches auf der Oberfläche zwischen Substrat und Chip nahezu die gleiche Sperrschichttemperatur erreicht werden kann, wie bei Einsatz eines Kühlkörpers. Die Verwendung von großflächigen Halbleitern wirkte sich auf die Temperaturverteilung auch positiv aus, da die Chipgröße als Wärmespreizer fungiert. Außerdem konnten Verbesserungen durch die Anwendung optimierter Klebeverbindungen erzielt werden. Betrachtet man die Ergebnisse der Simulationsprogramme, so fällt auf, daß teilweise sehr große Diskrepanzen bei der Wärmeverteilung zwischen Messung und Simulation vorliegen. Das bedeutet, je komplizierter die Maßnahmen der Wärmeentsorgung sind, um so mehr muß auf die Details wie Wärmesenken, Konvektion und Strahlung eingegangen werden. Schließlich kommt man zu dem Schluß, daß die vielen reproduzierbaren Meßergebnisse für die Korrektheit der Messung sprechen und die Simulationen bei dieser Problemstellung noch entwicklungsfähig sind. Das bedeutet, die Komplexität der Schaltung muß datentechnisch stärker in das Simulationsprogramm eingehen. Trotzdem verdeutlichen die Simulationen durch den verhältnismäßig

geringen Berechnungsaufwand und der schnellen Veränderung von Parametern die Unverzichtbarkeit in der heutigen Technik bei der Entwicklung und dem Aufbau von Leistungshybridschaltungen. Es werden aber weiterhin Testschaltungen nötig sein, um die Ergebnisse zu erhärten und zu verbessern. Zukunftsweisend bei der Idealisierung der Simulationsprogramme ist sicher die Darstellung der Wärmeverteilung in dreidimensionaler Form, da die Aufbauten an Komplexität zunehmen.
Abschließend wird empfohlen, die gegenwärtig vorliegenden Ergebnisse in einer Datenbank (Nachschlagewerk) zu speichern, um einen schnellen Zugriff auf die Daten bei der weiteren Entwicklung von Hybridschaltungen sicherzustellen.

7 Literaturverzeichnis

[1] Programm zur Berechnung des zeit- und frequenzabhängigen Sperrschichttemperaturverlaufs von Leistungshalbleitern mit Kühlkörpermontage
Werner Schäfer, Studienarbeit 1987, FH Aalen

[2] Zusammenstellung von Dimensionsunterlagen zur Berechnung des zeit- und frequenzabhängigen Sperrschichttemperaturverhaltens von Leistungshalbleitern mit Kühlkörpermontage unter Verwendung des Rechnerprogramms ST
Hartmut Stadelmaier, Studienarbeit 1989, FH Aalen

[3] Durchgängiger Temperatur-Meßplatz für Halbleiter Hybrid-Schaltungen
Rüdiger Haag, Diplomarbeit 1987, FH Rheinland-Pfalz

[4] Simulation des Temperaturverlaufs bei Schichtschaltungen
Erwin Biebl, Diplomarbeit 1986, TU München

[5] Das Temperaturfeld auf einem bestückten Substrat mit Optimierung des Leistungsverbrauchs
Harry Beer, Diplomarbeit 1987, Universität Stuttgart

[6] Hybridschaltungen - Schlüssel zu moderner Aufbau- und Verbindungstechnik
Walter Neugebauer, Peter Wilhelm
PKI Tech. Mitt. 3/1989, Seite 83

[7] Wärmeableitung bei Halbleiterbauelementen
Otmar Kilgenstein
Stuttgart, 1/1974, 13. Jahrgang

[8] Hybridintegration
Reichl H.
Hüthig Verlag 1986

[9] Verbindungstechnik'88
VDI Berichte 673

[10] Principles of Electronic Packaging
Donald P. Seraphim, Ronald Lasky and Che-Yu Li
MC Graw-Hill, 1989, ISBN 0-07-056306-3

[11] Circuits, Interconnections and Packaging for VLSI
Bakoglu H.B.
Addison-Wesley, 1990, ISBN 0-201-06008-6

[12] A Thermal Modell for Hybrid Circuits
G. Casselman and G. De Mey
Hybrid Circuits No.10, May 1986, Seite 9

[13] Models and Algorithms for the Temperature Distribution Evaluation on Substrates of HIC's
Erich Wehrhahn, Proceedings of ISCAS 85
IEEE 1985, p.415

[14] Low Thermal Resistance Hybrid IC Package
Hirosi Arikawa, Shigeto Hanada, Tsunehito Yokoi, Takashi Sekino
ISHM Proceedings 1988

[15] Taschenbuch der Physik
H. Kuchling, Harry Deutsch Verlag 1986

[16] Physik für Ingenieure
Hering, Martin, Stohrer
VDI Verlag 1988, ISBN 3-18-400655-7

[17] Siemens Datenbücher
SIPMOS Bauelemente 1987/88

[18] Smart SIPMOS, TEMPFET and PROFET
Siemens, Data Sheets 1990

[19] Halbleiterbauelemente und integrierte Mikroschaltungen
Normblätter DIN 41862, (1971)

[20] Thermische Optimierung der Aufbautechnik von Leistungshybriden, Diplomarbeit SS 1990, FH München, G. Schmidt

[21] Entwurf und Simulation von temperaturoptimierten Leistungshybriden, J. Kerler, Diplomarbeit SS 1991 FHT-Esslingen Außenstelle Göppingen