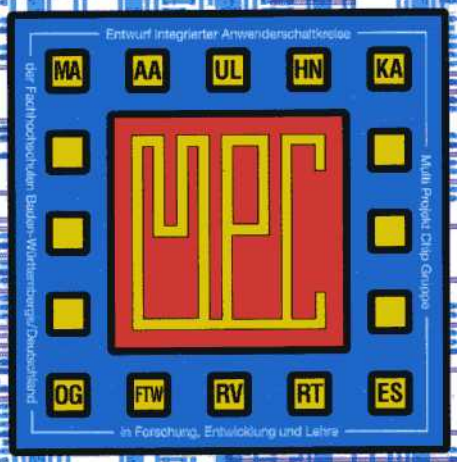
MULTIPROJEKT CHIP-GRUPPE

BADEN - WÜRTTEMBERG

Workshop Juli 1994

Mannheim



MULTIPROJEKT CHIP-GRUPPE

BADEN - WÜRTTEMBERG

Workshop Juli 1994

Mannheim

Herausgeber: Fachhochschule Ulm

Inhaltsverzeichnis

Selbstgetaktete Schaltungen
"Motivation und Entwurfsmethodiken"

C. Heer, H.-J. Pfleiderer
Abteilung für Allgemeine Elektrotechnik und Mikroelektronik
Universität Ulm
D-89069 Ulm

0. Einleitung

In dem Beitrag sollte eine Einführung in den Entwurf selbstgetakteter Schaltungen gegeben werden. Der vorliegende Text stellt dabei eine kurze Erklärung zu den folgenden Folien dar, die auf dem MPC-Workshop präsentiert wurden.

1. Motivation

Mit der technologischen Entwicklung zu immer kleineren Bauelementen im Sub-µm-Bereich wächst die Komplexität integrierter Schaltungen. Vergleicht man die Komplexität von CMOS-Logik Bausteinen mit der Komplexität eines Straßennetzes, so hat sich die Größe des Netzes innerhalb der letzten 30 Jahre um den Faktor 750 erhöht. Das Verdrahtungsnetz heutiger Mikroprozessoren entspricht dabei einem innerstädtischen Straßennetz von der Größe ganz Europas. Mikroprozessoren werden von einem zentralen Takt angesteuert. Übertragen auf das Modell des Straßennetzes bedeutet das, daß alle Verkehrsampeln von Moskau bis Madrid gleichzeitig schalten.
In selbstgetakteten Schaltungen werden deshalb ebenso wie in komplexen Straßennetzen nur noch Module aufeinander synchronisiert, die über Datenpfade (Verkehrswege) miteinander verbunden sind.
Mit der Entwicklung zu kleineren Bauteilen ändern sich auch die Schaltzeiten der Bauelemente. Während die Schaltzeit eines MOS-Transistor immer weiter reduziert wird, bleibt die Laufzeit von RC-Leitungen konstant. Wird die Fläche der integrierten Schaltungen erhöht, so steigt sogar die Laufzeit auf globalen Leitungen. Dieses Problem wird seit einigen Jahren durch Verwendung von systolischen Feldern gelöst, in denen nur noch lokal Operationen durchgeführt werden. Das Taktnetz zur Synchronisation der Schaltungen ist aber weiterhin global. Selbstgetaktete Schaltungen führen auch hier zu einer Lokalisierung, so daß auch das globale RC-Netzwerk des Taktsystem nicht mehr vorhanden ist.

2. Ingenieurtechnische Grundbegriffe selbstgetakteter Schaltungen

Selbstgetaktete Schaltungen lassen sind in drei verschiedene Einheiten teilen. Sie bestehen aus einem Logikelement, in dem die logische Funktion realisiert ist, einem Speicherelement zur Aufnahme der berechneten Daten und einer Koordinations- oder Handshake-Schaltung zur Steuerung der Operationen.
Bisher wird für die Logik vorwiegend die DCVS-Logik (Differential Cascode Voltage Switch Logic [1]) verwendet. Die Schaltung wird von einem Start-Signal angesteuert und es kann aus internen Signalen ein Fertigsignal erzeugt werden. In einer Vorladephase

(Start="0") werden zwei interne Knoten intQ und intQQ auf den Wert der Versorgungsspannung VDD aufgeladen und damit die Ausgangsknoten über einen Inverter auf das Massepotential gelegt. Dann wird über das Startsignal (Start="1") die Rechenoperation ausgelöst. Abhängig von den komplementären Eingangsdaten wird nun entweder intQ oder intQQ über einen Baum aus N-Kanal Transistoren (N-Baum) auf das Massepotential entladen und damit einer der Ausgangsknoten Q oder QQ auf VDD aufgeladen. Aus den nun komplementären Werten auf den internen Knoten intQ und intQQ kann über ein NAND-Gatter sehr einfach das Ende der Operation erkannt und als Fertigsignal an die Koordinationsschaltung weitergegeben werden. Der interne Knoten, der nicht entladen wurde, muß statisch gegen Spannungseinbrüche durch Ladungsumverteilung gesichert werden.
Die einfachste Handshake-Schaltung besteht aus einem sogenannten Muller-C-Element. Wenn beide Eingangssignale des C-Elementes denselben Wert besitzen, wird dieser an den Ausgang übernommen. Bei unterschiedlichen Werten an den Eingängen, bleibt der vorhergehende Ausgangswert gespeichert. Aus einem Inverter und diesem C-Element kann ein sehr einfacher Handshake aufgebaut werden. Nach einer Initialisierung aller Signale auf logisch "0" wird ein Vier-Phasen-Protokoll abgearbeitet. Nach dem Setzen der Anfrage am Eingang (Request_in="1") wird das Requestsignal am Ausgang gesetzt (Request_out="1") und gleichzeitig die Bestätigung (Acknowledge_out="1") zurückgegeben. Nachdem die Bestätigung vom Nachfolger (Acknowledge_in = "1") empfangen wurde, wird die Rücksetzphase des Protokolls analog durchlaufen (Request_in="0" ; Request_out="0"; Acknowledge_out="0" ; Acknowledge_in="0"). In einem sogenannten Signalübergangsgraphen kann dieses Protokoll anschaulich dargestellt werden. Aus diesem Graphen kann neben der Verifikation auch eine Analyse der Schaltung durchgeführt werden. Für einen kompletten Zyklus des Handshake-Protokolls muß jeder Kreis im Signalübergangsgraphen einmal durchlaufen werden. Der Kreis mit der längsten Verzögerungsdauer bestimmt dann die minimale Zykluszeit des Protokolls.
Am Beispiel einer einfachen Kette (Pipeline) von Speicherelementen und Funktionsblöcken können verschiedene Konfigurationen für selbstgetaktete Schaltungen diskutiert werden.
Deutlich wird die Trennung zwischen einem Daten- und einem Kontrollpfad. Die Speicherelemente des Datenpfades bestehen jeweils aus einem Transmissionsgatter und einem Inverter. Parallel zu einem gültigen Datum wird nun ein Steuersignal durch die Pipeline geführt, das die Speicherelemente in den nichttransparenten Zustand schaltet. Es muß garantiert werden, daß das Datum das Speicherelement vorher passiert hat. Bei einer "leeren" Pipeline wird ein Datum sofort die gesamte Kette durchlaufen, während das Steuersignal erst langsam nachfolgt und die Speicherelemente sperrt. In der "vollen" Pipeline wird jedes zweite Speicherelement im nichttransparenten Zustand ein Datum enthalten und das dazugehörige Muller-C-Element wird gesetzt sein.
Um in einer solchen Pipeline auch logische Operationen durchführen zu können, werden Funktionsblöcke nach den oben beschriebenen Schaltungstechniken eingesetzt.
Für die sogenannte Full-Handshake-Konfiguration wird nach jedem zweiten Latch ein Funktionsblock eingesetzt. Der Signalpfad wird dabei aufgetrennt und es wird der Funktionsblock eingesetzt. Dadurch wird eine zusätzliche Verzögerung in den Kontrollpfad eingebracht. Aus dem Signalübergangsgraphen für diese Konfiguration kann

man die minimale Zykluszeit ermitteln. Diese besteht hier aus einer Rechenphase, einer Vorladephase und vier Signalübergängen im Handshake. Es läßt sich kein einfacher Kreis im Signalübergangsgraphen finden, der zwei Rechenoperationen enthält. Das bedeutet, daß alle Funktionsblöcke der Pipeline gleichzeitig Operationen ausführen können (Full-Handshake [2]).
Bei der Half-Handshake-Konfiguration wird nach jedem Latch ein Funktionsblock eingesetzt. Im zugehörigen Signalübergangsgraphen finden wir einfache Kreise mit zwei Rechenoperationen. Das bedeutet, daß die Operationen sequentiell ausgeführt werden müssen. Deshalb kann nur noch die Hälfte aller Funktionsblöcke gleichzeitig eine Operation ausführen (Half-Handshake [2]). Dadurch erhöht sich die minimale Zykluszeit auf zwei Rechenoperationen und vier Handshake-Signalübergänge, aber der Speicheraufwand pro Funktionsblock reduziert sich.
Das für den Handshake beschriebene Protokoll wird als Return-to-zero oder Vier-Phasen Protokoll bezeichnet. Nach einer Rechenphase mit dem Setzen des Startsignals und dem Warten auf die Bestätigung erfolgt die Rücksetzphase mit dem Rücksetzen des Startsignals und dem Rücksetzen der Bestätigung. Interpretiert man die Rücksetzvorgänge ebenfalls als Startaufforderung bzw. als Bestätigung, so erhält man ein Zwei-Phasen Protokoll, das aus alternierenden Startaufforderungen und Bestätigungen besteht. Die Signalisierung erfolgt jetzt aber nicht mehr zustands- sondern flankengesteuert. Der Datendurchsatz kann mit diesem Protokoll nahezu verdoppelt werden. Allerdings sind jetzt aufwendigere Funktionsblöcke und Speicherelemente erforderlich. Diese müssen flankengesteuert auslösbar sein. Die Handshake-Schaltung kann übernommen werden.

3. Algebraische Grundbegriffe

Man unterscheidet in der Literatur drei Klassen asynchroner selbstgetakteter Schaltungen. Verzögerungszeitunabhängige Schaltungen gewährleisten korrektes Verhalten unabhängig von allen auftretenden Gatter- und Leitungsverzögerungszeiten. Gatterlaufzeitunabhängige Schaltungen gewährleisten korrektes Verhalten unabhängig von allen auftretenden Gatterlaufzeiten, aber Annahmen über Leitungslaufzeiten sind notwendig. Für laufzeitabhängige Schaltungen werden Annahmen über Gatter- und Leitungslaufzeiten benötigt, um korrektes Verhalten zu garantieren.
Das einzelne C-Element kann als verzögerungszeitunabhängige Schaltung betrachtet werden. Allerdings muß auch die Umgebung diese Eigenschaft erfüllen. Wenn ein Eingangssignal des C-Elementes geändert worden ist, das zu einer Änderung des Ausgangszustandes führt, darf sich kein weiteres Eingangssignal ändern, bis sich das Ausgangssignal auch geändert hat. In den verschiedenen Darstellungsformen kann das Verhalten des C-Elementes erklärt werden.
Aber schon ein Gatter aus zwei C-Elementen ist nicht mehr verzögerungszeitunabhängig. Es läßt sich ein Fall konstruieren, bei dem eine Änderung an einem gemeinsamen Eingang der beiden C-Elemente aufgrund von Leitungslaufzeit zu unterschiedlichen Zeitpunkten an den einzelnen Elementen detektiert wird. Dies kann zu Fehlverhalten führen. Hier müssen also Bedingungen über Leitungslaufzeiten auf sich verzweigenden Leitungen angenommen werden (Isochronic Forks).

Als gatterlaufzeitunabhängig gilt eine Schaltung, wenn die Freigabe eines Signalübergangs erst durch dessen Ausführung wieder zurückgenommen werden kann. Es läßt sich aber zeigen, daß bei dieser Bedingung aufgrund von Laufzeiteffekten Hazards auftreten können. Diese lassen sich verhindern, wenn die Bedingung zur Freigabe schärfer formuliert wird. Wenn die Freigabe eines Signalüberganges durch ein Steuersignal erfolgt ist, so kann erst durch die Ausführung des Signalüberganges das Steuersignal zurückgesetzt werden.
In laufzeitabhängigen Schaltungen werden die Verzögerungszeiten von Logik und Speicherelementen ermittelt und dann als Verzögerungselemente im Kontrollpfad implementiert.

4. Entwurfsmethodik und Überblick über entworfene Schaltungen

Die verschiedenen Entwurfsmethodiken sollen hier nicht explizit beschrieben werden. Es wird auf die sehr ausführliche Literatur verweisen. Dort finden sich auch detaillierte Beschreibungen der mit den jeweiligen Methoden entworfenen Schaltungen.

Delay-Insensitiv Algebra

M. Josephs, J.-T. Udding; "The Design of a Delay-Insensitiv Stack" in Designing Correct Circuits; Jones and Sheeran, Workshops in Computing, 1991; Springer
M. Josephs, J.-T. Udding; "Delay-insensitiv Circuits: an Algebraic Approach to their Design" in CONCUR 90, Theories of Concurrency; Baeten and Klop, 1990; Springer

Communicating Sequential Processes

A. Martin; "Formal program Transformations for VLSI Circuit Synthesis" in Formal Developement of Programs and Proofs; ed. E. Dijkstra, 1990, Addison-Wesley
A. Martin; "Programming in VLSI" in Developements in concurrency and communication; ed. C.A.R. Hoare, 1990, Addison-Wesley
C.A.R. Hoare; "Communicating sequential processes"; Comm. of the ACM, 21, 1978

Tangram

K. van Berkel; "Handshake Circuits, an asynchronous architecture for VLSI Programming"; 1993, Cambridge Int. on Parallel Comp., Cambridge University Press
Block Diagramm Ansatz und Signalübergangsgraphen
T. Meng; "Synchronization Design for Digital Systems"; 1991, Kluwer Academic Pub.
L. Lavagno; "Algorithms for Synthesis and Testing of Asynchronous Circuits", 1993, Kluwer Academic Publishers

Micropipeline

I. Sutherland, "Micropipelines"; Turing Award, 1989, Comm. of the ACM

5. Vergleich verschiedener Entwurfsmethodiken

An der dänischen technischen Hochschule in Lyngby ist ein Vergleich verschieden Entwurfsmethoden durchgeführt worden. Dem synchronen Entwurf kam von Aufwand und Durchsatzrate der Micropipeline-Entwurf am nächsten. Die Schaltung ist laufzeitabhängig, d.h. implementiert Verzögerungszeiten im Kontrollpfad und kommt daher dem synchronen Ansatz am nächsten. Die Entwürfe der DTH und vom CALTECH (Martin) waren aufgrund ungünstiger Logikrealisierungen bzw. aufwendiger aber gatterlaufzeitunabhängiger Realisierung wesentlich flächenintensiver und langsamer.
Trotz dieses ernüchternden Vergleichsergebnisses sind bereits in Nischenanwendungen selbstgetaktete Schaltungen zu finden. Ein Feldmultiplizierer für höchste Datenraten der Telecom Bretagne nutzt die dynamische Logik und kann wahlweise synchron oder selbstgetaktet betrieben werden [3].

6. Zusammenfassung

Für kommende Technologiegenerationen werden selbstgetaktete Schaltungen ein größeres Interesse finden. Die Methoden, die heute zur Schaltungsentwicklung und Verifikation eingesetzt werden finden bereits bei der Verifikation von Datenübergabeprotokollen Verwendung. Die Entwurfsgrundlagen sind gelegt. Es fehlt eine bessere Entwurfsunterstützung und eine bessere Modellierung selbstgetakteter Schaltungen.

7. Literatur

[1] L. G. Heller, W. R. Griffin; "Cascode Voltage Switch Logic: A Differential CMOS Logic Family", Proceedings of the IEEE ISSCC 84, pp 16,17

[2] O. Aumann, C. Heer; "Handshake-Schaltungen für selbstgetaktete Systeme", ITG Fachbericht 119, Seiten 217-222

[3] R. Marc, E.H. Bachar; "A Low Power, 100MHz 12x18-30b Multiplier-Accumulator Operating in Asynchronous and Synchronous Modes"; ESSCIRC '94, Ulm

Fachhochschulen in Baden-Württemberg
Multi Projekt Chip Gruppe
Workshop SS94
Fachhochschule Mannheim, 1.7.1994

Selbstgetaktete Schaltungen "Motivation und Entwurfsmethodiken"

C. Heer, H.-J. Pfleiderer
Abteilung für Allgemeine Elektrotechnik
und Mikroelektronik
Universität Ulm
89069 Ulm

Überblick

Motivation

Ingenieurtechnische Grundbegriffe selbstgetakteter Schaltungen

Logik und Steuerung / Protokolle

Algebraische Grundbegriffe

Entwurfsmethodik

Algebraisch, programmtechnisch, CSP basierend
Ingenieurtechnischer Ansatz

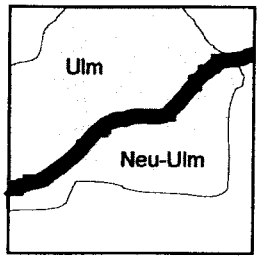
Überblick über entworfene Schaltungen

Vergleich verschiedener Entwurfsmethodiken

Zusammenfassung

Komplexität integrierter Schaltungen

4 km

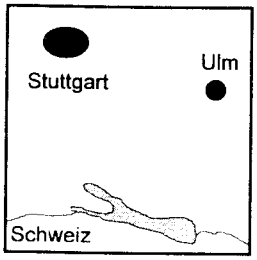


1963

Leiterbahnweite: 25 µm
Leiterbahnabstand: 50µm

Chipgröße: 1mm x 1mm

100km

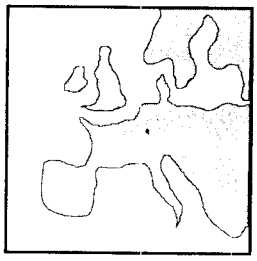


1978

Leiterbahnweite: 5µm
Leiterbahnabstand: 10µm

Chipgröße: 5mm x 5mm

3.000 km



1993

Leiterbahnweite: 0.5 µm
Leiterbahnabstand: 1µm

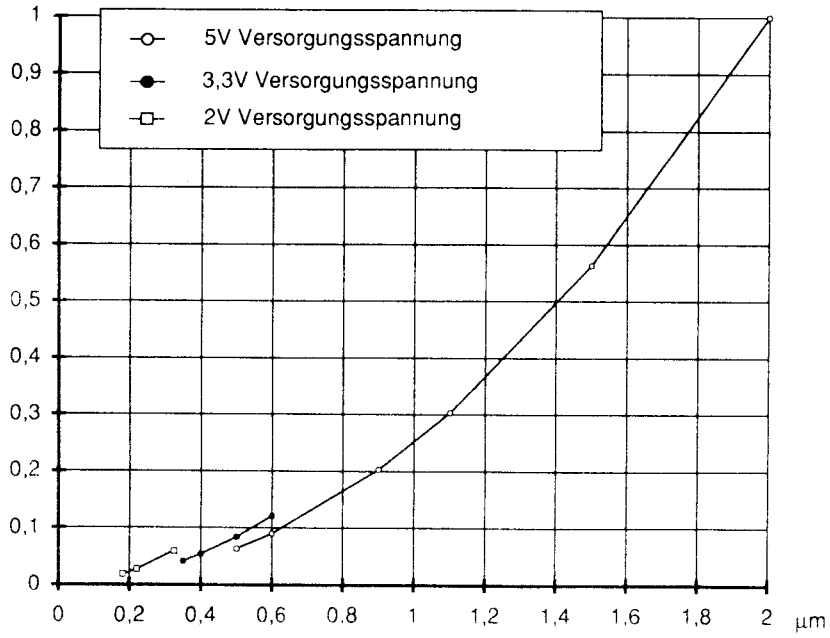
Chipgröße: 15mm x 15mm

Annahme: Straßenblockgröße: 200 x 200 m

Quelle: C. Seitz, Caltech VLSI Conference 1979

Technologische Entwicklung

Gatterverzögerungszeit



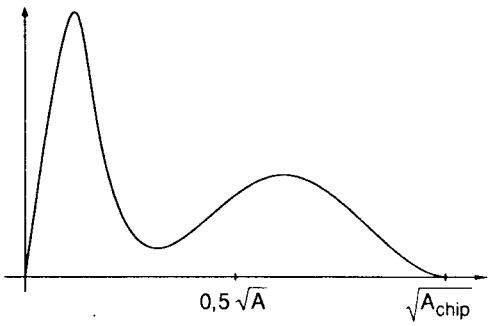
Kanallänge der Transistoren L=2λ

Parameter	Skalierungsfaktor
Geometrie (W, L, d_{ox})	1/s
Spannungen (V_{dd}, V_{th})	1
Sättigungsstrom I_{DS}	s
Gate-Kapazität	1/s
intrisische Gatterlaufzeit	$1/s^2$
RC-Leitungslaufzeiten	1

Skalierung der Schaltungsgröße

Parameter	Skalierungsfaktor
Chipfläche	S_c^2
lange Verbindungen	S_c
deren Widerstände	$s^2 S_c$
deren Kapazitäten	Sc
deren RC-Laufzeit	$S^2 S_c^2$

Anzahl der Verbindungen



RC-Laufzeit globaler Leitungen nimmt zu

↓

Lokalisierung der Operationen (Systolisch Zellenfelder)

↓

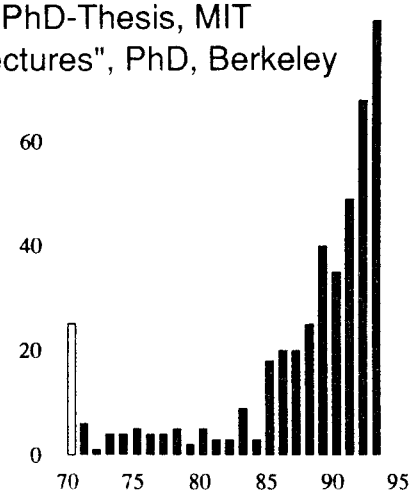
Aber der Taktnetzwerk bleibt global

↓

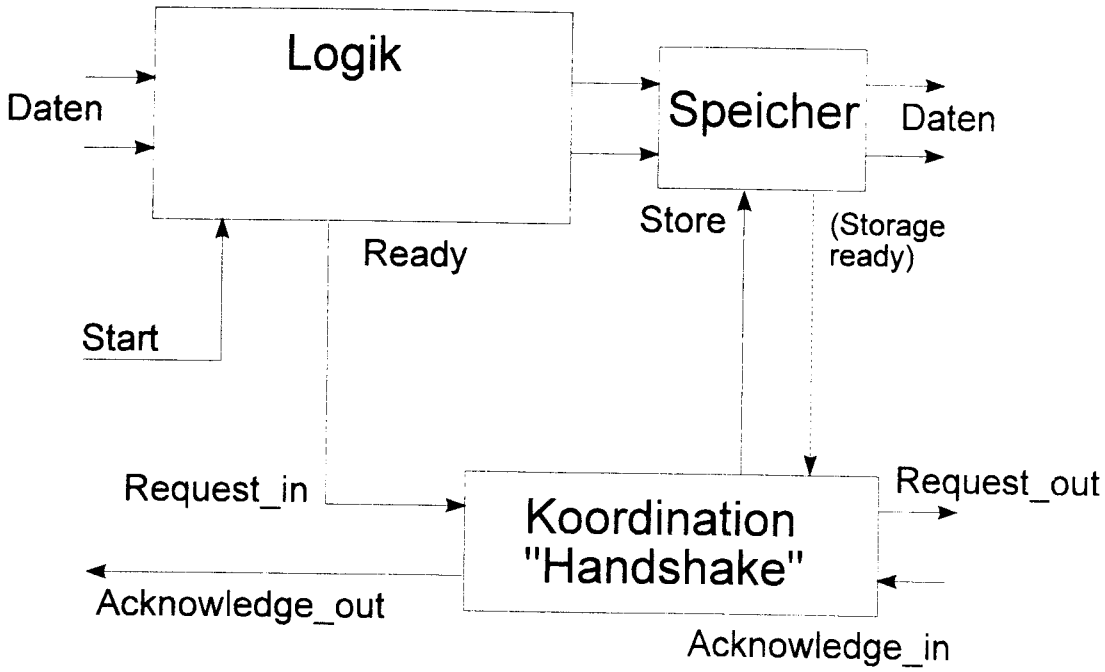
lokale Kommunikation
lokale Steuerung

Historischer Überblick (Publikationen)

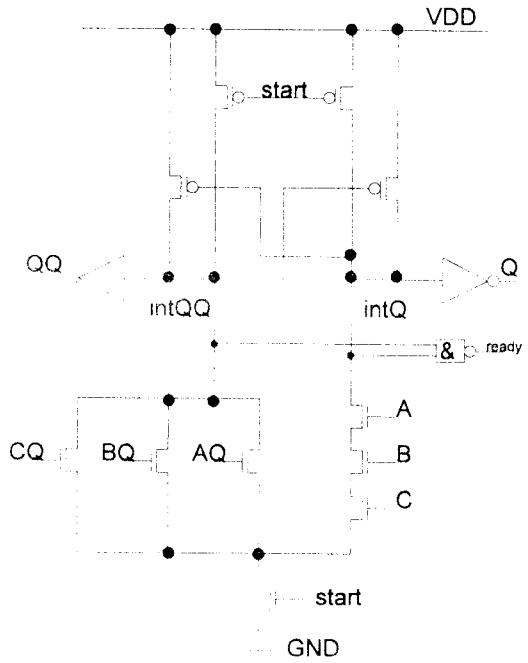
1957	D. E. Muller: "A Theory of Asynchronous Circuits", Wiley
1969	S. H. Unger: "Asynchronous Sequential Switching Circuits", Wiley
1979	C. Seitz: "System Timing" in Mead, Conway "Introduction in VLSI Systems" , Addison-Wesley
1987	T. Chu: "Synthesis of Self-Timed VLSI Circuits", PhD-Thesis, MIT
1988	T. Meng: "Asynchronous Design for DSP Architectures", PhD, Berkeley
1989	I. Sutherland: "Micropipelines", ACM Turing Award
1994	aktuelle Bibliographie zu "Self-Timed Circuits" ca. 500 Titel,



Blockbild einer selbstgetakteten Zelle



Logik für selbstgetaktete Schaltungen



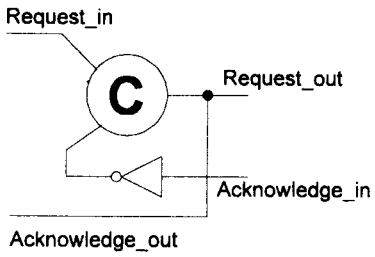
Modifizierte
Differential Cascode Voltage
Switch Logic (DCVSL)

- Vorladezustand
  start="0"; ready="0"

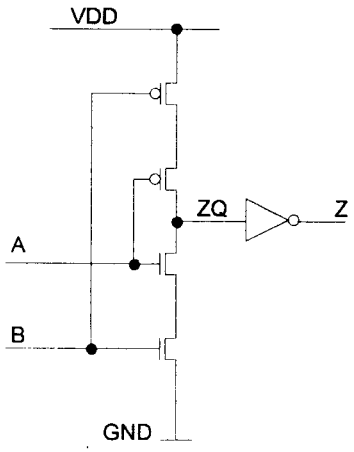
- Rechenphase
  start="1"; ready="1"

- Koppplungstransistoren zur statischen Sicherung des nicht entladenen internen Knotens

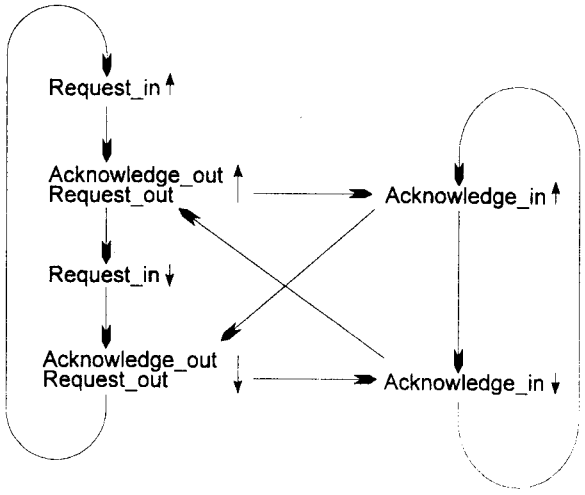
Muller-C-Element als Handshake-Schaltung



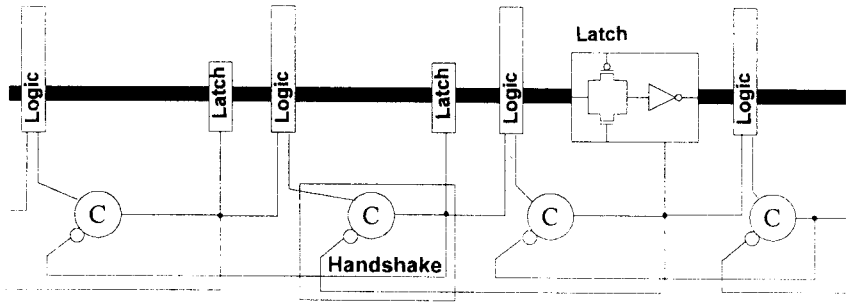
Dynamisches Muller-C-Element als Handshake-Schaltung



Signalübergangsgraph für C-Element-Handshake

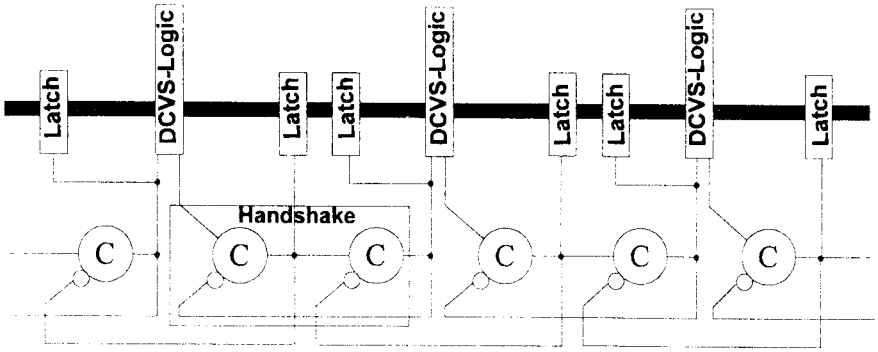


Halfhandshake-Konfigurationen einer selbstgetakteten Pipeline



Halfhandshake-Konfiguration

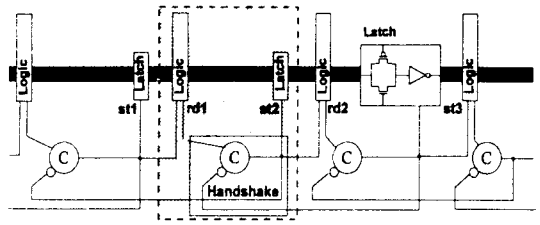
C-Element/Latch pro Logikmodul

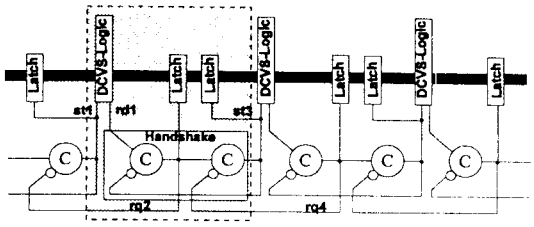


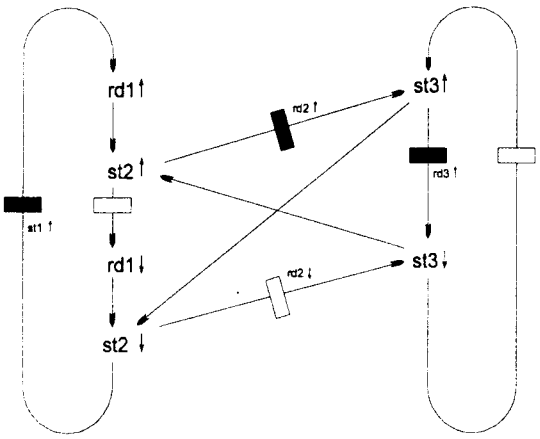
Fullhandshake-Konfiguration

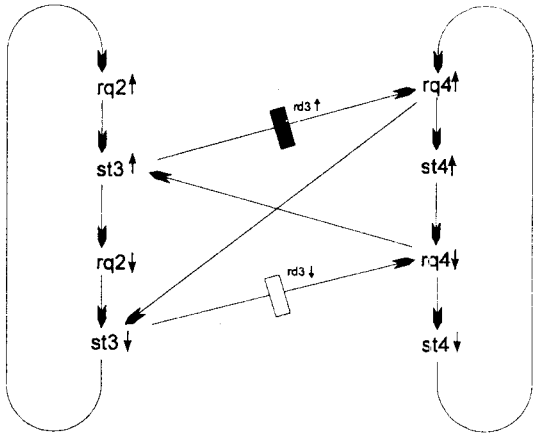
zwei C-Elemente/ zwei Latch pro Logikmodul

Signalflußgraph für die Handshake-Konfigurationen

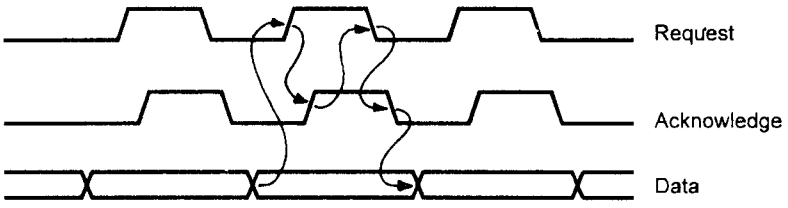






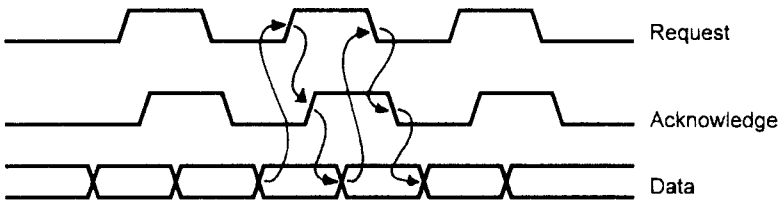


2-Phasen oder 4-Phasen Protokolle



4-Phasen Protkoll
(Return to Zero Protokoll)

Zustandsgesteuert



2 Phasen Protokoll

Flankengesteuert

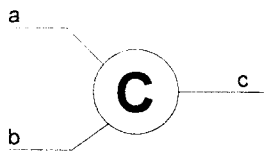
Drei Klassen asynchroner selbstgetakteter Schaltungen

Verzögerungszeitunabhängige Schaltungen ...
..... gewährleisten korrektes Verhalten unabhängig von allen auftretenden Gatter- und Leitungsverzögerungszeiten.

Gatterlaufzeitunabhängige Schaltungen ...
.....gewährleisten korrektes Verhalten unabhängig von allen auftretenden Gatterlaufzeiten, aber Annahmen über Leitungslaufzeiten sind notwendig.

Laufzeitabhängige Schaltungen (Bundled Data)...
.....benötigen Annahmen über Gatter- und Leitungslaufzeiten, um korrektes Verhalten zu garantieren.

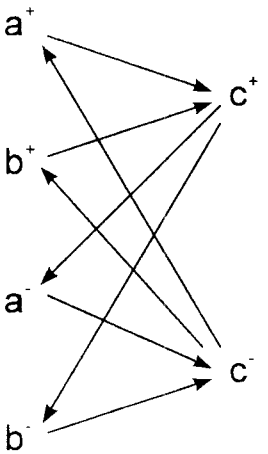
Verzögerungszeitunabhängige Schaltungen (Delay insensitiv Circuits)



Beispiel: C-Element

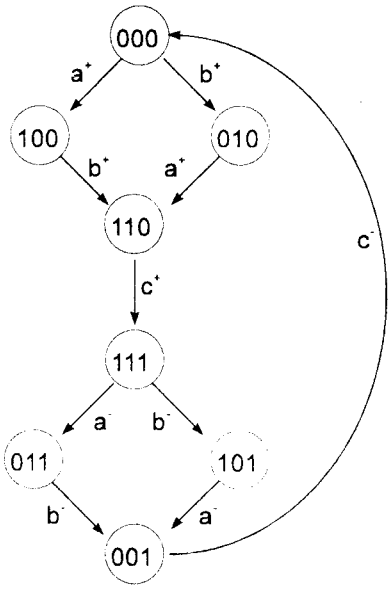
Funktionsbeschreibung in einer CSP-ähnlichen Syntax

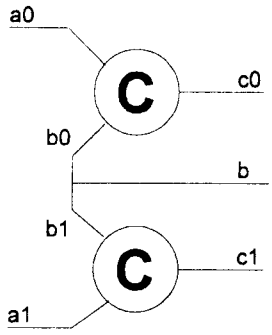
$$*\left[[a \wedge b]; c\uparrow; [\neg a \wedge \neg b]; c\downarrow\right]$$



Signalübergangsgraph

Zustandsdiagramm





Beispiel 2: Parallele C-Elemente

Funktionsbeschreibung in einer CSP-ähnlichen Syntax

$$*\left[[a0 \wedge b];c0\uparrow;[\neg a0 \wedge \neg b];c0\downarrow;[a1 \wedge b];c1\uparrow;[\neg a1 \wedge \neg b];c1\downarrow\right]$$

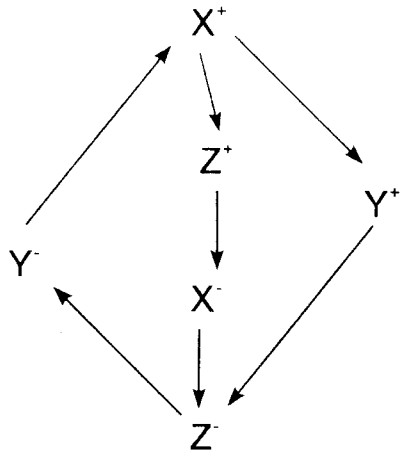
korrekte Beschreibung:

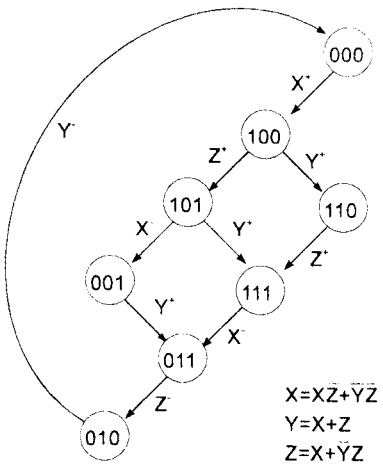
$$*\left[[a0 \wedge b0];c0\uparrow;[\neg a0 \wedge \neg b0];c0\downarrow;[a1 \wedge b1];c1\uparrow;[\neg a1 \wedge \neg b1];c1\downarrow\right]$$

Problem: mögliche Laufzeitunterschiede zwischen b0 und b1!
Forderung: "Isochronic Fork" b=(b0,b1)

Eine Schaltung ist Gatterlaufzeitunabhängig, wenn gilt:

- Das Zustandsdiagram ist streng zusammenhängend.
- Jeder Zustand ist eindeutig beschreibbar durch einen Bitvektor
- Wenn ein Signalübergang freigegeben worden ist, bleibt er solange freigegeben, bis er ausgeführt worden ist.

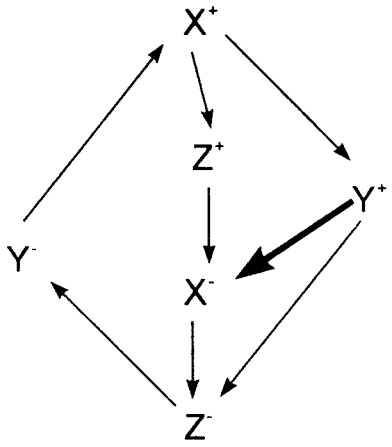


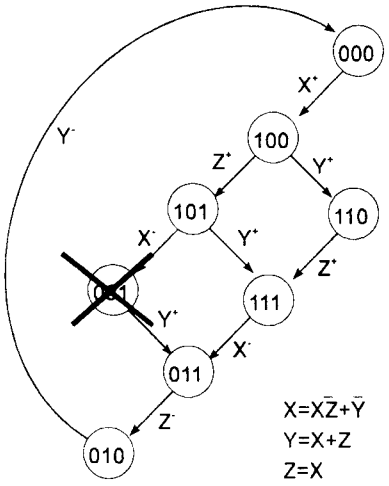


Beispiel für einen Signalübergangsgraphen und zugehörigen Zustandsgraphen

Ein Signalübergangsgraph ist semi-modular, wenn gilt:

- Die einmal erfolgte Freigabe des Signalübergangs kann nur durch das Ausführen dieses Signalübergangs wieder zurückgesetzt werden.
- Ein semi-modularer Signalübergangsgraph ist hazardfrei



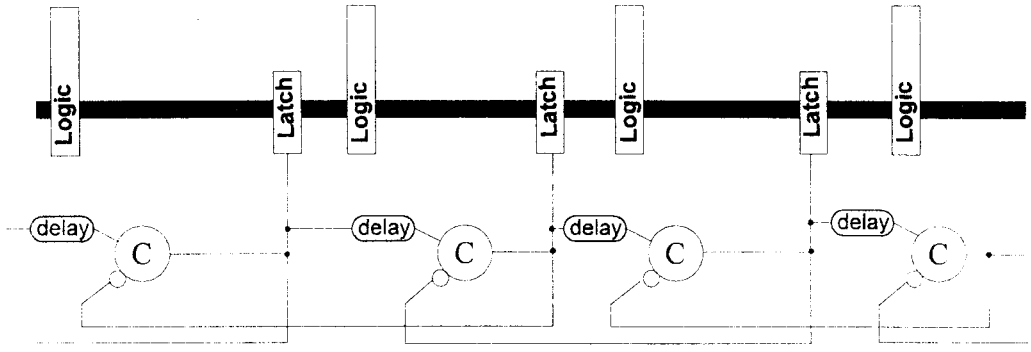


Beispiel für einen Signalübergangsgraphen und zugehörigen Zustandsgraphen

- Annahmen über Gatter- und Leitungslaufzeiten (Bounded Delay-Model)

- Koordination mit Handshakeschaltungen

Beispiel einer Pipeline:



Algebraisch programmtechnische Methoden (CSP - Communicating Sequential Processes)

Delay***I***nsensitiv-Algebra

CSP-Methode (Alain Martin, Caltech)

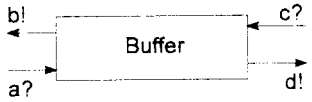
TANGRAM (Philips, Eindhoven)

Ingenieurtechnischer Ansatz

UC Berkeley, Kalifornien (Meng, Jacobs, Brodersen)

Micropipeline (University of Manchester)

Entwurfsbeispiel: One-Place Buffer



DI-Algebra: Modell-, Prozeß-Spezifikation und Auflösung

$E = a?;b!;F$

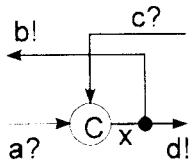
$F = c?;d!;E$

aktiver Ausgang: Initialisierung mit c?

$E/c? = [a? \rightarrow [b! \rightarrow [skip \rightarrow d!;E \# c? \rightarrow \perp] \# c? \rightarrow \perp] \# c? \rightarrow \perp]$

$= [a? \rightarrow b!;[d! \rightarrow E \# c? \rightarrow \perp] \# c? \rightarrow \perp]$

$= [a? \rightarrow b!;[d! \rightarrow c?;(E/c?) \# c? \rightarrow \perp] \# c? \rightarrow \perp]$



Implementation mit:

$B/c? = [a? \rightarrow b!;d!;c?;(B/c?) \# c? \rightarrow \perp]$

$B = c?;a?;b!;d!;B$

C – Element: $C(a,b,c) = a?;b?;c!;C(a,b,c)$

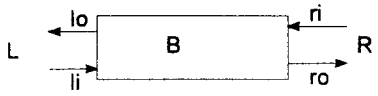
Fork – Element: $F(a,b,c) = a?;b!;c!;F(a,b,c)$

$B = C(a,c,x) \| F(x,b,d)$

DI-Algebra:

- Hochsprachliche Beschreibung der Prozesse
- Abbildung auf definierte Funktionsmodule
- Sehr aufwendige Beschreibung und Verifikation
- Sehr schwerer Zugang zur Theorie

Entwurfsbeispiel: One-place buffer



$$B \equiv *[L;R]$$

Handshake-Expansion:

$$B \equiv *[[li];lo\uparrow;[\neg li];lo\downarrow;ro\uparrow;[ri];ro\downarrow;[\neg ri]]$$

Reshuffling:

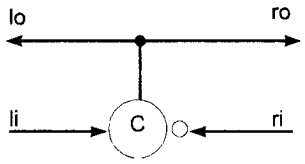
$$B \equiv *[[li];lo\uparrow;ro\uparrow;[ri];[\neg li];lo\downarrow;ro\downarrow;[\neg ri]]$$

Production Rules:

$\neg ri \wedge li \mapsto lo\uparrow,\ ro\uparrow$

$ri \wedge \neg li \mapsto lo\downarrow,\ ro\downarrow$

Schaltkreis:

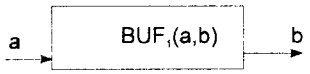


CSP-Methode:

Einfache Beschreibung auf Prozeßebene

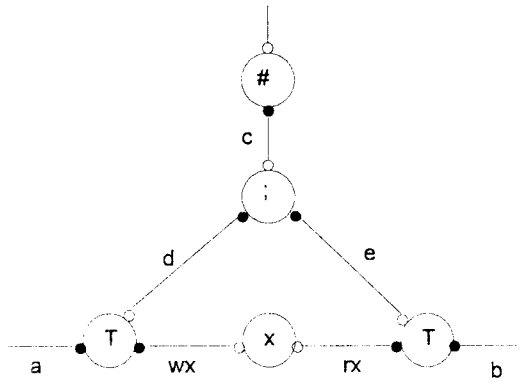
Entwicklung bis auf Transistorebene / relativ einfacher Zugang

Entwurfsbeispiel: One-place buffer



$$(a?W \,\&\, b!W) \left[\left[x: \text{var } W \mid \#[a?x; b!x]\right]\right]$$

Handshake-Schaltkreis für BUF:

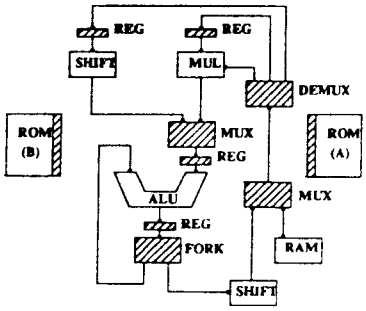


- Repeater
; - Sequencer
T - Transferrer
x - Variable

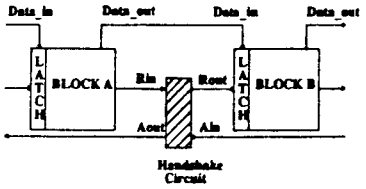
● - aktiver Port
○ - passiver Port

TANGRAM:
- einziges kommerziell eingesetztes Entwurfswerkzeug
- erzeugt nicht immer die optimalen Schaltungen (Programmabhängig)

Block Diagramm Ansatz



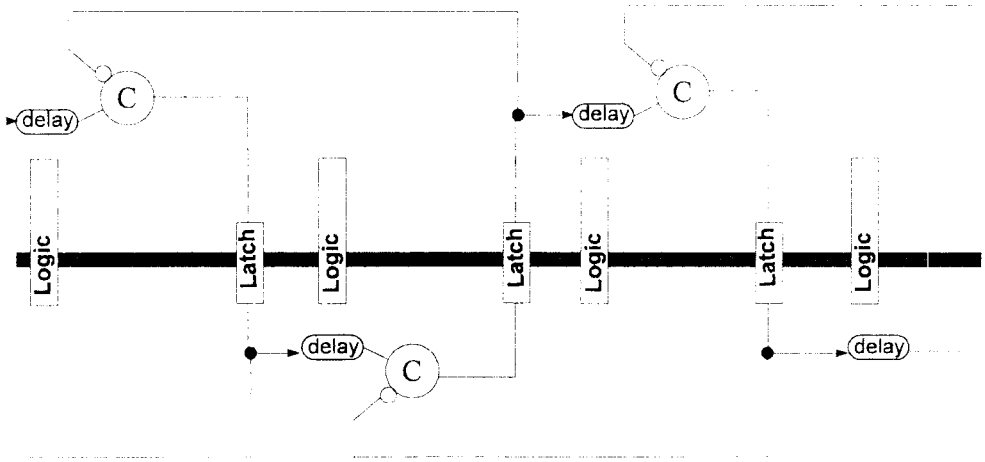
Ausschnitt aus Verbindungsblock



- Signalübergangsgraphen zur Beschreibung
- Überprüfung der Gatterlaufzeitunabhängigkeit (Semi-Modularity)
- Durchsatzrate aus graphischer Analyse

I.Sutherland "Turing Award Lecture", Comm. ACM 6/1989

- Pipeline Architekturen
- Flankengesteuerte Protokolle
- "Bundled Data"-Protokoll
- Aufbau aus Basis Elementen (XOR-, C-, Toggle-, Call-Element, ...)



Digitaler Signalprozessor (UC Berkeley, Meng, Jacobs)

Digitaler Signalprozessor (ARM, Universität Manchester)

Digitaler Signalprozessor (Caltech, Alain Martin)

Reed-Salomon Decoder (Philips, Eindhoven)

Vergleich verschiedener Methodiken an einem Multiplizierer

Beispiel: 100 MHz-Feldmultiplizierer

Digitaler Signalprozessor (UC Berkeley, Meng, Jacobs)

Block Diagram

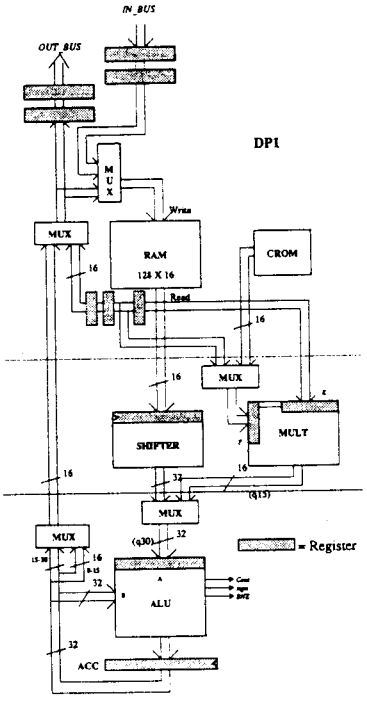
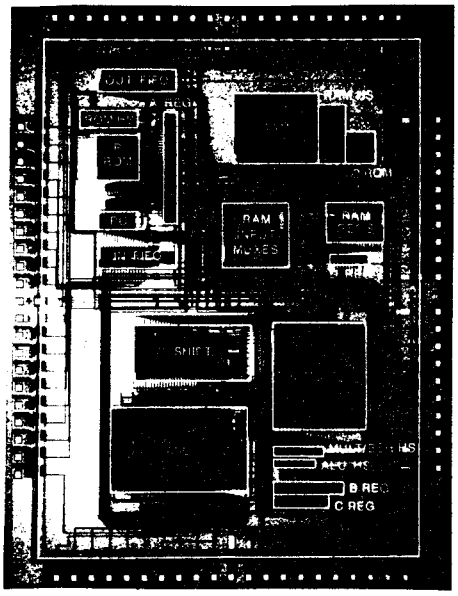


Photo des asynchronen DSP



Digitaler Signalprozessor (UC Berkeley, Meng, Jacobs)

4-Phasen-Protokoll, Full-Handshake Konfiguration
DCVS-Logik, Ein-Zyklus Multiplizierer, -Alu und -Shifter

2µm CMOS Technologie (MOSIS)
20.000 Transistoren
6,6 mm x 4,7 mm aktive Fläche

Rechenzeit bei 5V:
Multiplikation: 337 ns
Shift: 73 ns

ca. 3 MHz Zyklusfrequenz

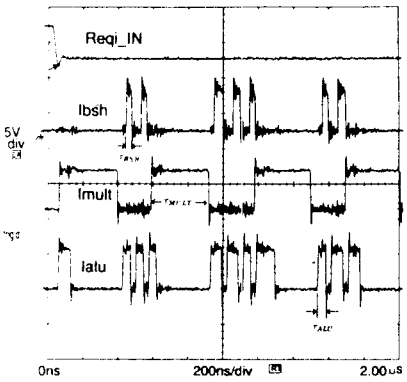


FIGURE 5—Oscilloscope traces showing timing signals from the DSP implementing an IIR filter.

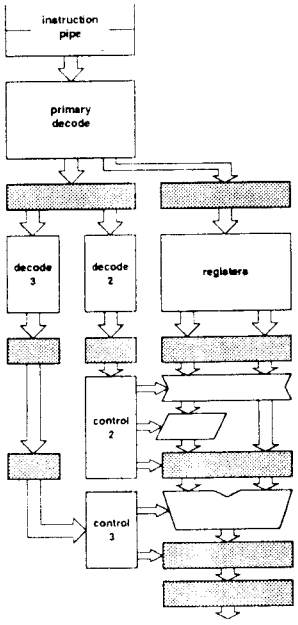
Digitaler Signalprozessor (Micropipeline, Universität Manchester)

Pipeline-Architektur basierend auf dem ARM Mikroprozessor
(Advanced RISC Machines, Acorn, Apple, VLSI Technology)

ESPRIT (OMI)
Beginn 1989

Full Custom Entwurf auf
ARM Zellenbibliothek und
besondere asynchrone Zellen

Tape-out Ende 1992
erste Exemplare 4/1993



Digitaler Signalprozessor (Caltech, Alain Martin)

einfacher Prozessor zur Demonstration verschiedener Module:
Queue, Stack, Mutual Exklusion, Arbiter

RISC-Architektur
12 Register, 4 Busse, ALU, 2 Addierer

20.000 Transistoren
2,0 µm CMOS MOSIS
6,6 mm x 4,6 mm

ca. 8 MHz Zyklusfrequenz

funktional für Versorgungsspannungen von 1V bis 7V

Reed-Salomon Decoder für DCC-Gerät (Philips, Eindhoven)

bisher einzige in einem Produkt verwendete selbst-getaktete Schaltung

automatisch aus einer TANGRAM-Beschreibung generiert:
44.000 Transistoren auf 11,5 mm^2 Gate-Array Fläche in 1,0µm CMOS

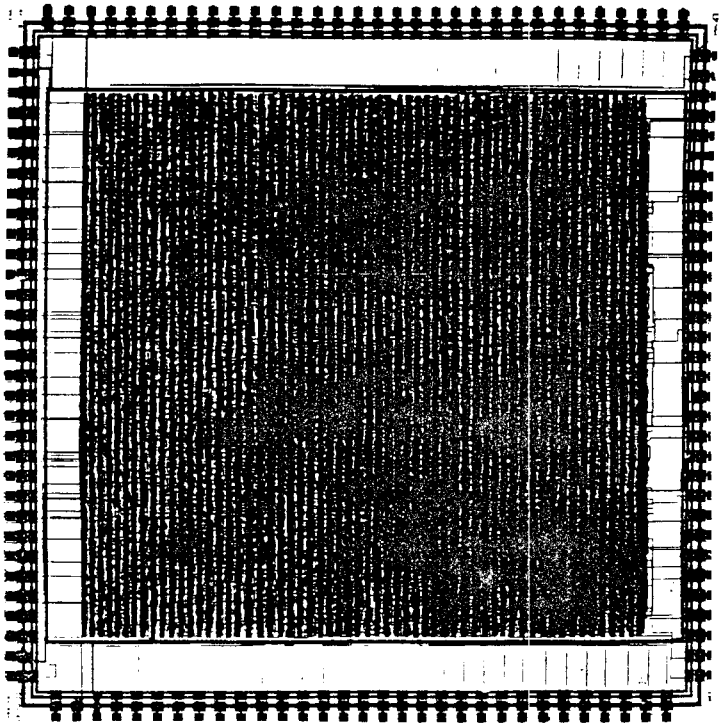
Zielapplikation sind tragbare Anwendungen
Entwurfskriterium war deshalb möglichst geringe Leistungsaufnahme

Vergleich zur synchron getakteten Version:
- doppelter Flächenbedarf
- ein Fünftel der Leistungsaufnahme (weitere Reduktion möglich)

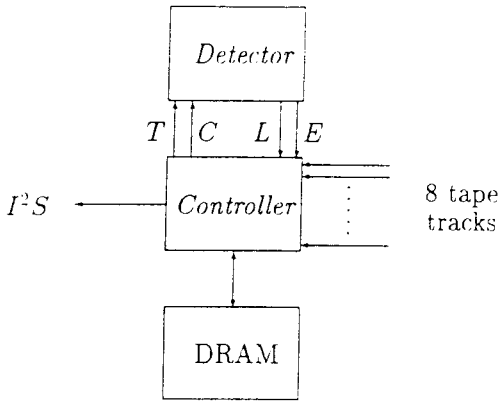
DCC-Decoder ist Demostrationsschaltung für Esprit 6143 Exact Projekt

Reed-Salomon Decoder für DCC-Gerät (Philips, Eindhoven)

Layout des Controllers



Blockbild des Controllers



Reed-Salomon Decoder für DCC-Gerät (Philips, Eindhoven)

Wie ist die im Vergleich zur synchron getakteten Schaltung reduzierte Leistungsaufnahme zu erklären?

Der Rechenaufwand des Decoders ist stark von den Eingangsdaten abhängig

muß ein Datum korrigiert werden, so ist die

- Rechenzeit um den Faktor 2,5 größer
- Energieaufnahme um den Faktor 3 größer

In der synchron getakteten Architektur wird dieser Unterschied nicht ausgenutzt
Für jedes Datum ist die gesamte Schaltung aktiv!

Im Mittel müssen aber nur 5% aller Daten korrigiert werden.
In der selbst-getakteten Realisierung wird in 95% der Fälle die Schaltung zur Korrektur nicht aktiviert:

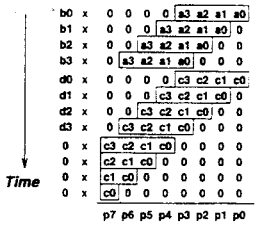
- Keine Leistungsaufnahme -

Dieser Trick ist synchron auch möglich, aber schwerer zu realisieren!

Untersuchung von drei verschiedenen Entwurfsverfahren

- CSP-basierender Entwurf (Martin, CALTECH)
- Multi-ring Entwurf (DTH Lyngby, Dänemark, ähnlich Meng et al.)
- Micropipeline

- synchron getaktete Schaltung



Testschaltung: Vektor-Multiplizierer

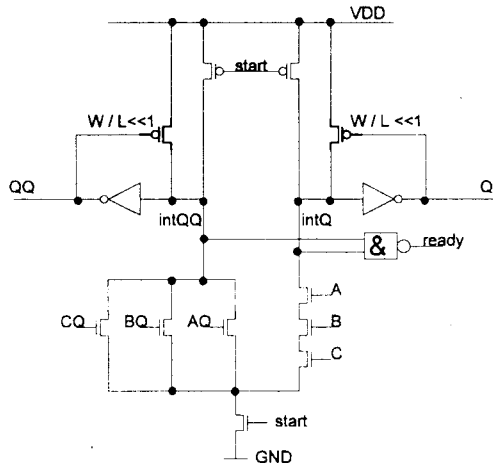
Aufbau aus Akkumulator und
Seriell-Parallel Multiplizierer

Entwurf	Transistor-zahl n	Layout-fläche mm^2	Zyklus-zeit ns	Energie pro Zyklus pJ	Energie x Zykluszeit 10^{-18} Js
Martin	347	0,28	18	365	6,6
Micropipeline	395	0,26	24	189	4,5
DIMS (DTH)	824	0,40	22	290	6,4
Synchron	188	0,10	12 (8 typ.)	88	1,1

100 MHz - Feldmultiplizierer (M. Renaudin; Telecom Bretagne)

12x18 + 30-b Multiplier-Accumulator

Vergleich von	CMOS-	DCVS-	Volladdierer
Schaltzeit	1,67 ns	0,78	
Leistungsaufnahme	$9{,}7 \cdot 10^{-5}$ W	$13{,}8 \cdot 10^{-5}$ W	
Transistorzahl	26	36	



DCVSL UND-Gatter

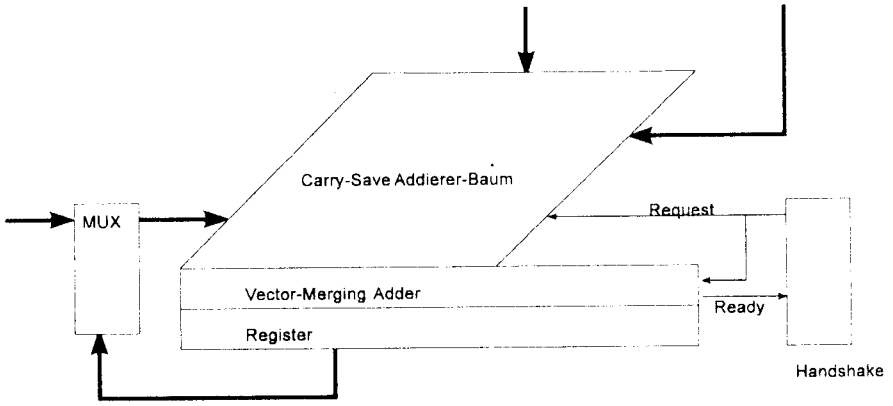
Vorlade- und Haltetransistoren

Ausgangsinverter
NAND-Gatter zur Fertigsignalgenerierung

N-Baum

100 MHz - Feldmultiplizierer (M. Renaudin; Telecom Bretagne)

Architektur



Carry-Save Addierer

DCVS-Domino Baum

synchroner und
asynchroner Betrieb
möglich

Daten der Testschaltung (0,5 µm CMOS, 3-Lagen Metallisierung)

1,32 mm² aktive Fläche, 11432 Transistoren

100 MHz typisch, 3 Volt Versorgungsspannung, 1,36 mW/MHz Leistungsaufnahme

Selbstgetaktete Schaltungen sind seit ca. 35 Jahren ein Thema

.... seit ca. 10 Jahren ein Gebiet intensiver Forschung

- mit kommenden Technologiegenerationen steigt das Potential asynchroner selbstgetakteter Schaltungen

- Grundlagen zum Entwurf und zur Verifikation sind gelegt

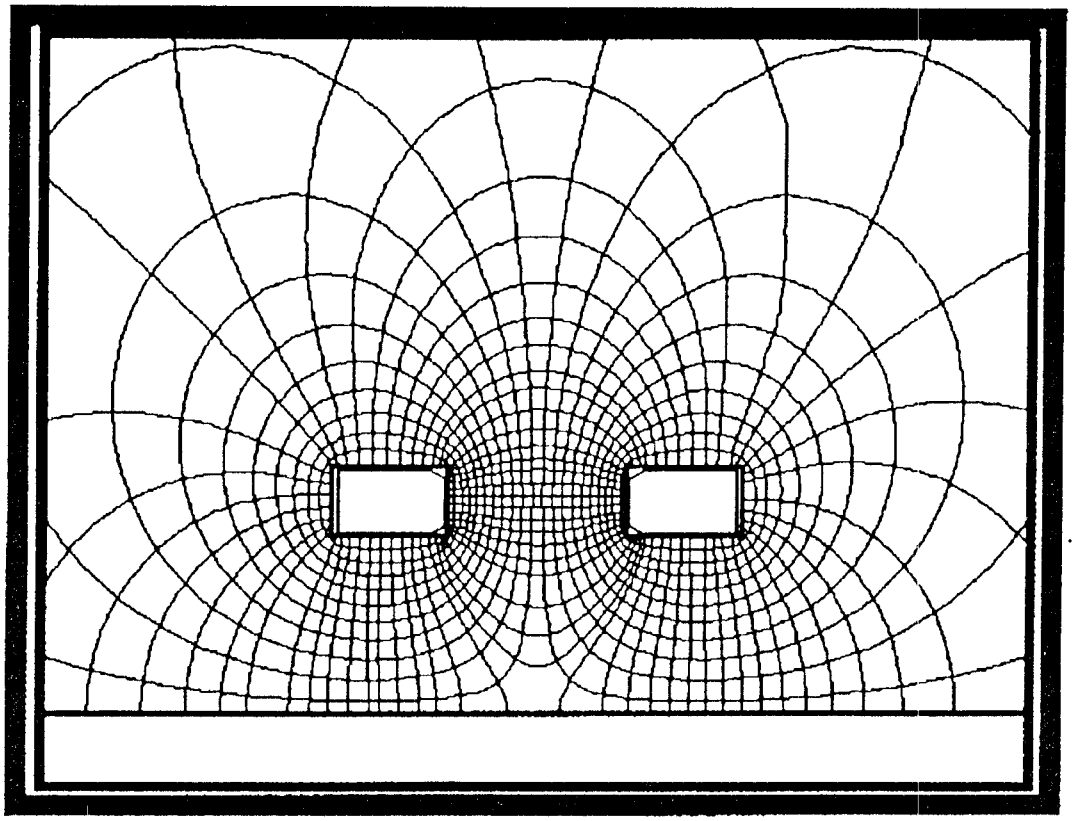
- es fehlen:
- Entwurfsunterstützung in Form von CAD-Werkzeugen
-. Effiziente Algorithmen für die Verifikation und Modellierung

"ingenieurtechnische Lösungen"

MPC- Workshop SS94

>> SIMULATION VON PARASITÄREN LEITERBAHNEFFEKTEN AUF GEDRUCKTEN SCHALTUNGEN MIT "POLARIS" <<

Bearbeiter: **Werner Frank**
Bernhard Thürmer
Prof. Dr. Horst Nielinger
FH Furtwangen

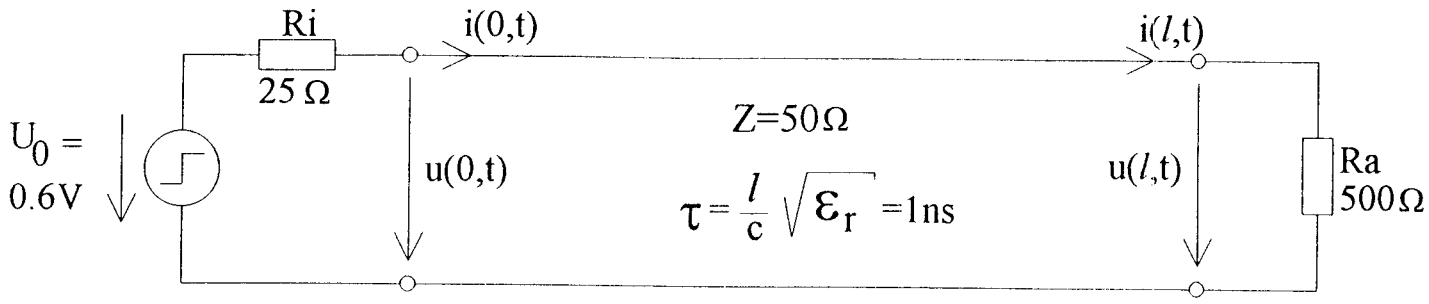


Kurze Einleitung zu POLARIS des Design Center

Das Programm POLARIS gestattet Leitungseffekte zu berücksichtigen. Das Programm wertet für die mit Hilfe eines PCB-Layout Programms entworfene Verbindungsstruktur der einzelnen Bauelemente die Maxwellschen Gleichungen aus und liefert an PSPICE die Leitungsparameter zurück wie z.B. Wellenwiderstand, Laufzeiten, Kopplungsfaktoren zwischen den Leitungen.
Dieser ganze Aufwand lohnt sich bei schnellen Digitalschaltungen, denn wenn die Laufzeit einer Verbindungsleitung zwischen zwei ICs deutlich länger ist als die Schaltzeit, ist zunächst der relativ niederohmige Wellenwiderstand der Verbindungsleitung maßgebend für Spannung und Strom unmittelbar nach dem Schalten und nicht der hochohmige Eingangswiderstand des nächsten ICs.
Dieser Konflikt führt zu ausgeprägten Einschwingvorgängen. Hier wird schon deutlich, daß man die Diskussion der Leitungseffekte im Zeitbereich führen muß unter Einbeziehung von nichtlinearen Randbedingungen, wenn man die Ergebnisse einer POLARIS-Simulation in der Ingenieurausbildung interpretieren und nicht einfach der Computersimulation vertrauen will.
Die klassische Leitungstheorie der HF-Technik bringt für diese Problematik sehr wenig!
Das hier leider sehr wenig behandelte Verfahren nach BERGERON gestattet, Sprung- bzw. Impulsantworten von Leitungsschaltungen mit linearen und nichtlinearen Randbebingungen zu ermitteln.
Um uns einen Zugang zu POLARIS zu verschaffen, haben wir einige klassische Beispiele die mit dem BERGERON-Verfahren gelöst werden können, auf dem Computer nachvollzogen.

Hinweis:
Es ist beabsichtigt eine erweiterte Fassung dieses Artikels in der Zeitschrift "Elektronik" zu veröffentlichen.

BERGERON'S METHOD



Boundary Condition Output: $$i(l,t) = \frac{u(l,t)}{Ra}$$

Boundary Condition Input: $$U_0 = i(0,t)\cdot Ri + u(0,t) \Rightarrow i(0,t) = -\frac{1}{Ri}\left[u(0,t) - U_0\right]$$

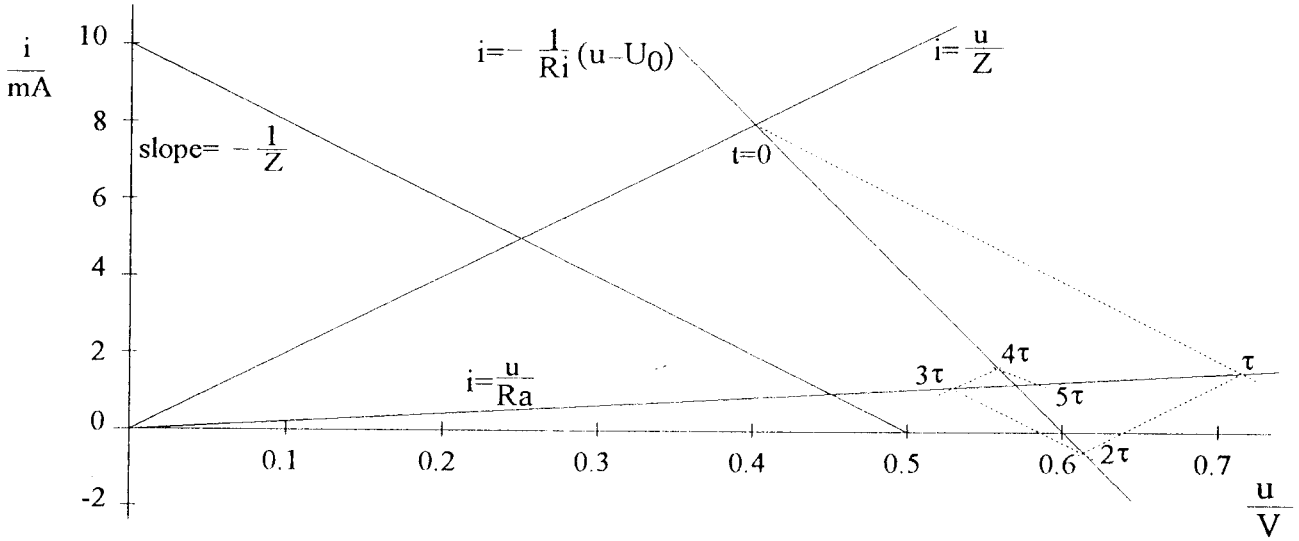
Reflected Wave at Input: $$u(0,t) - Z\cdot i(0,t) = u(l,t-\tau) - Z\cdot i(l,t-\tau)$$

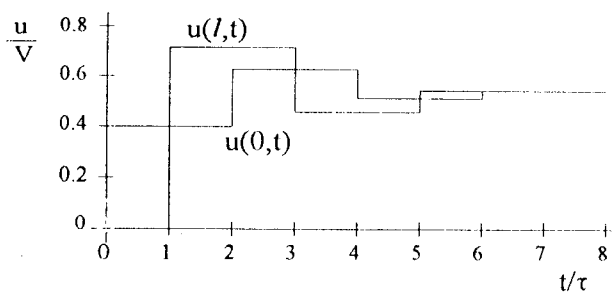
$$i(0,t) - i(l,t-\tau) = \frac{1}{Z}\left[u(0,t) - u(l,t-\tau)\right]$$

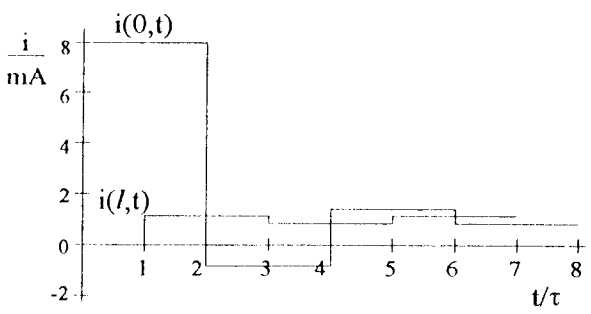
For t=0 with i(*l*,-τ)=0, u(*l*,-τ)=0: $$i(0,0) = \frac{1}{Z}u(0,0)$$

Incident Wave at Output: $$u(l,t) + Z\cdot i(l,t) = u(0,t-\tau) + Z\cdot i(0,t-\tau)$$

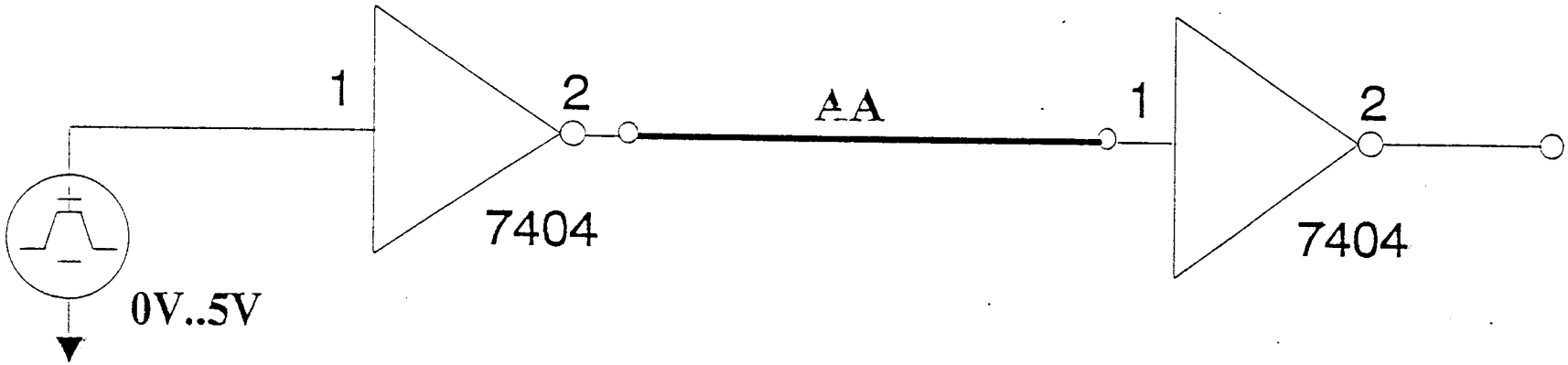
$$i(l,t) - i(0,t-\tau) = -\frac{1}{Z}\left[u(l,t) - u(0,t-\tau)\right]$$







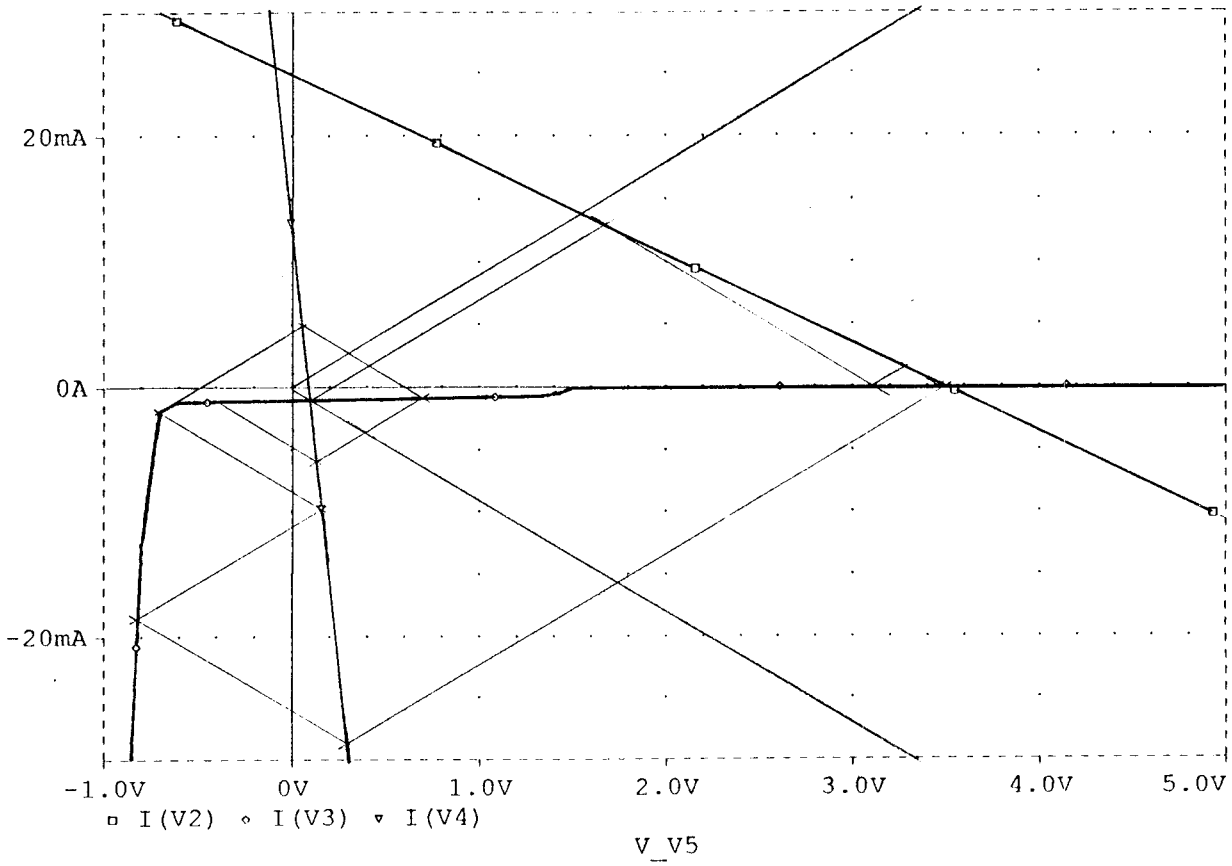
Bsp: Impuls auf eine Leitung (Nichtlineare Randbedingungen)



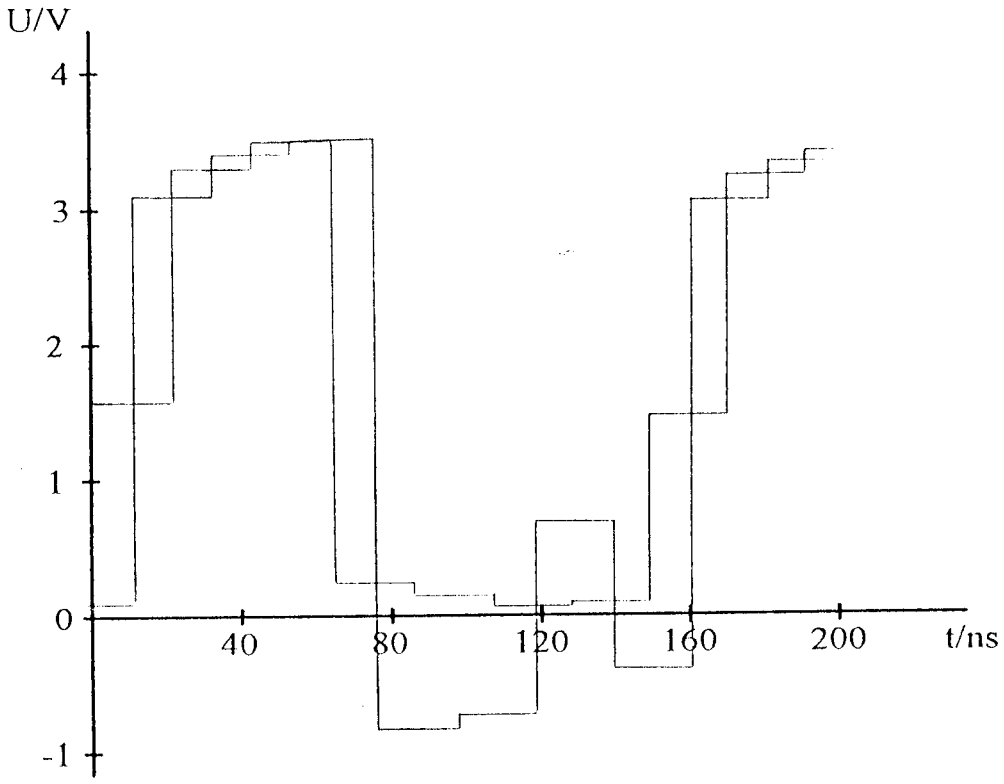
Von Polaris ermittelte Werte:

SELECT SIGNALS								
Name	Coupling	Length	L (nH)	R	C (pF)	C_via	Z0	Tpd (ns)
AA	0	1.897151	1193.20	1.901435	90.79	2.00	114.64	10.408124

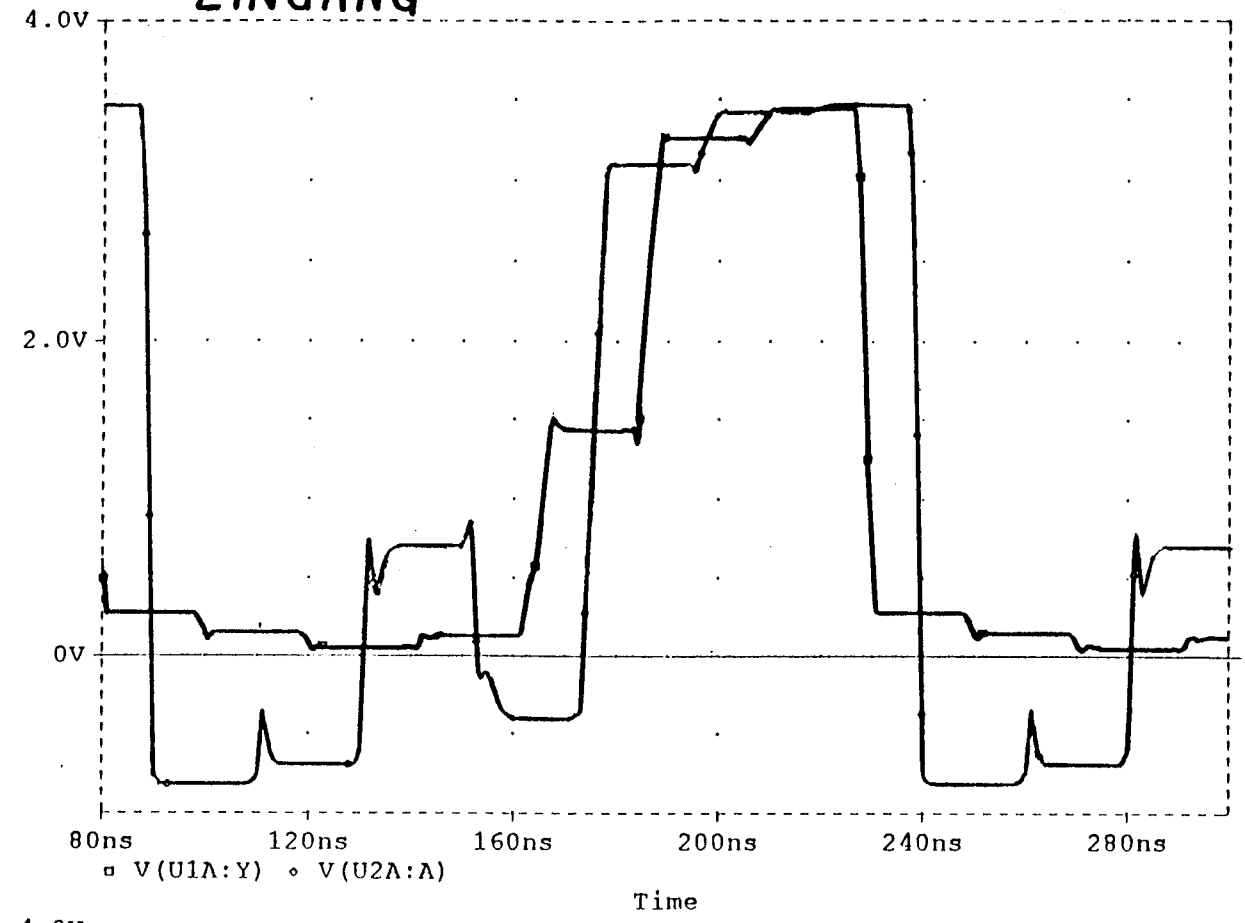
Verfahren von Bergeron

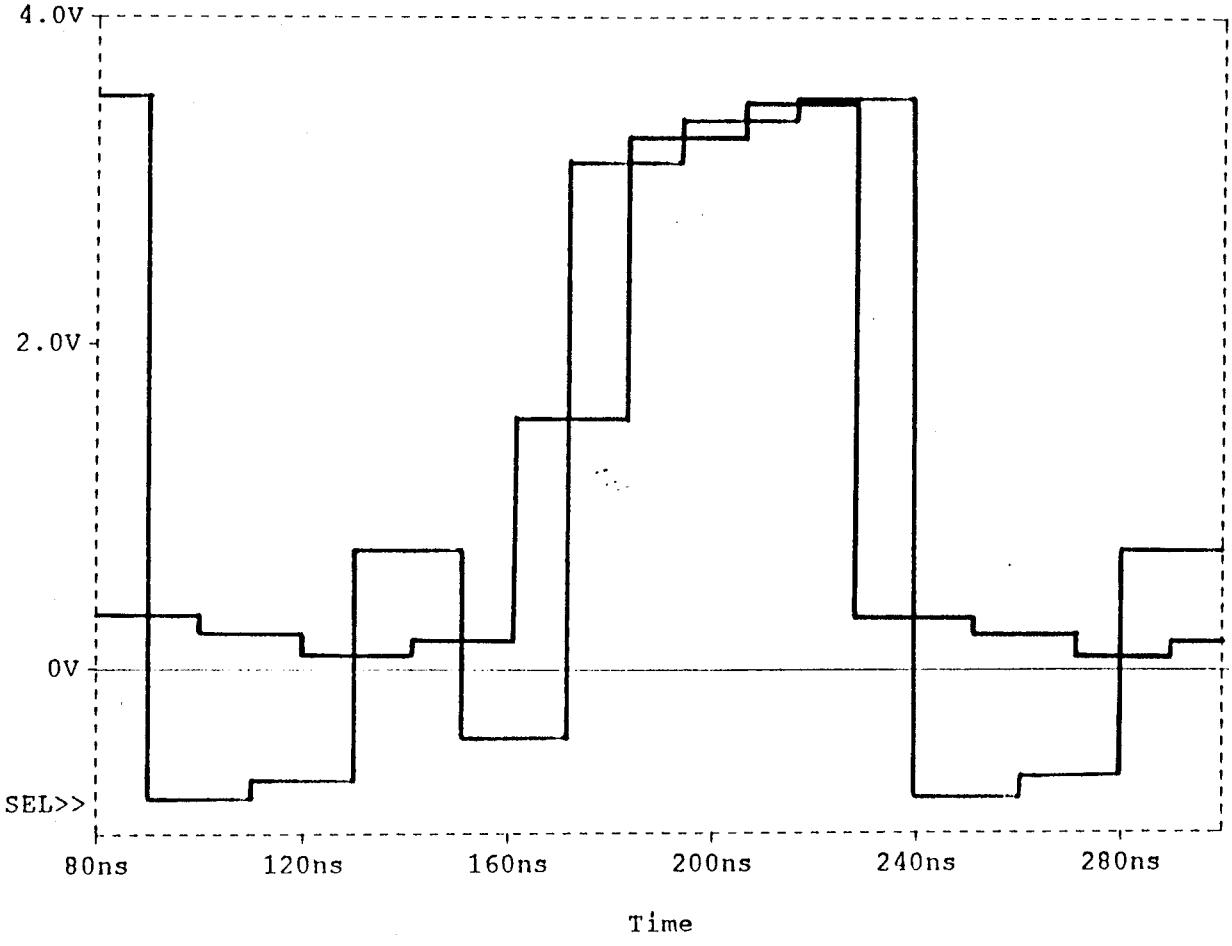


Auswertung

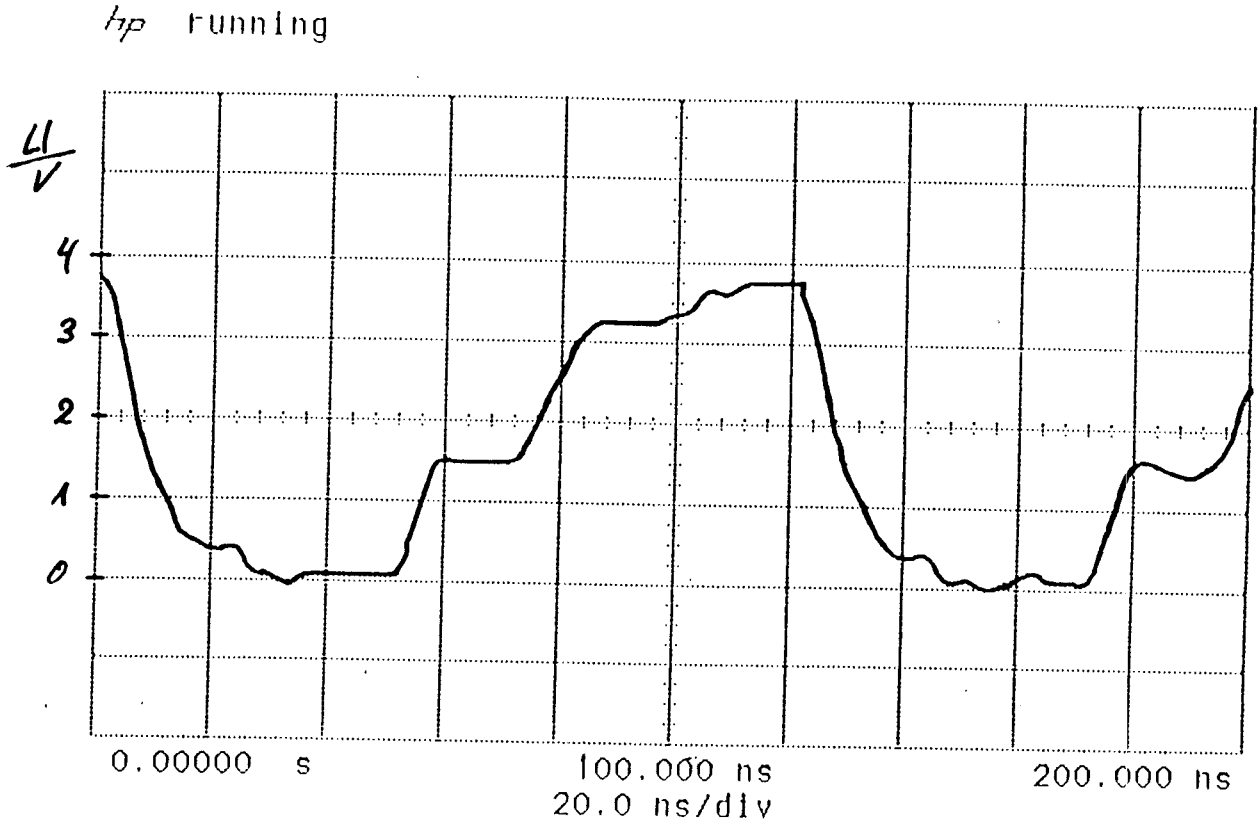


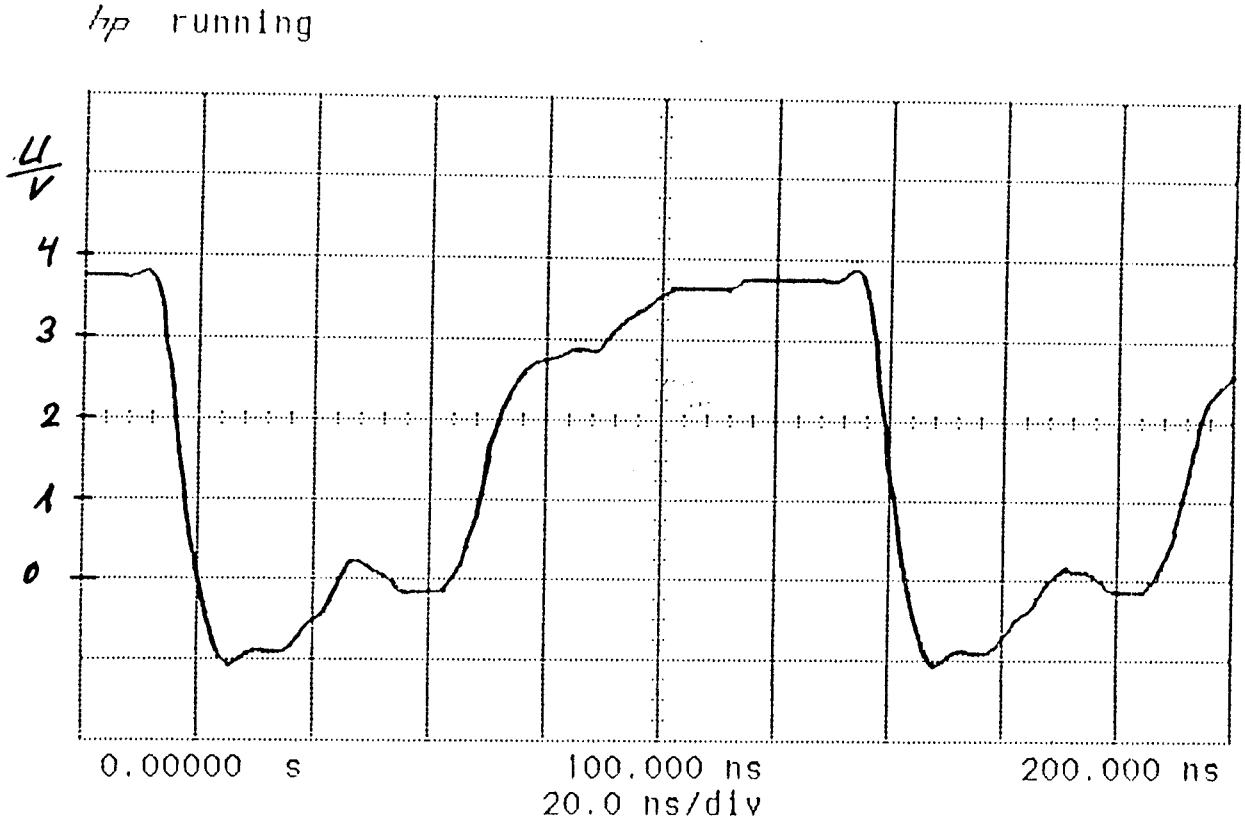
AUSGANG DER LEITUNG
EINGANG





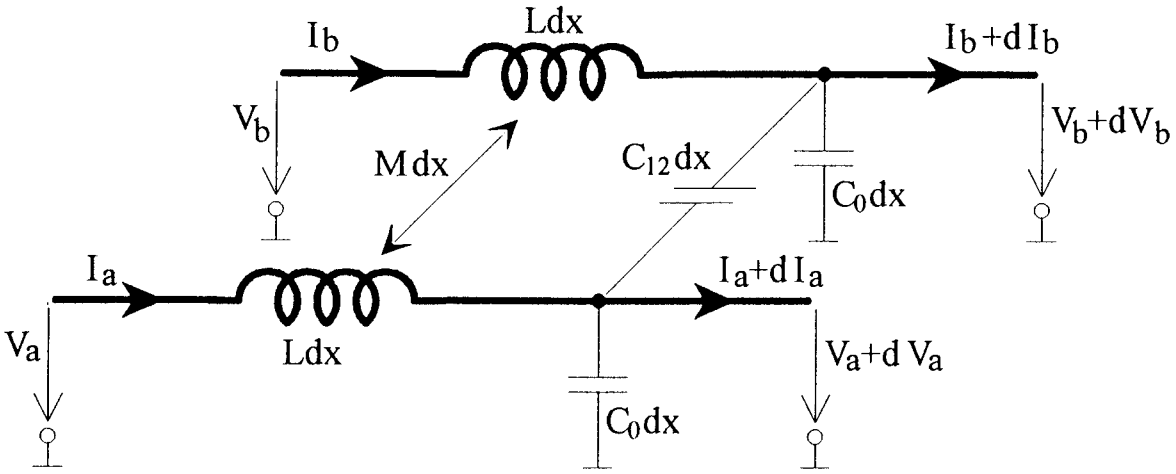
MESSERGEBNIS





COUPLED TRANSMISSION LINES

MATRIX APPROACH

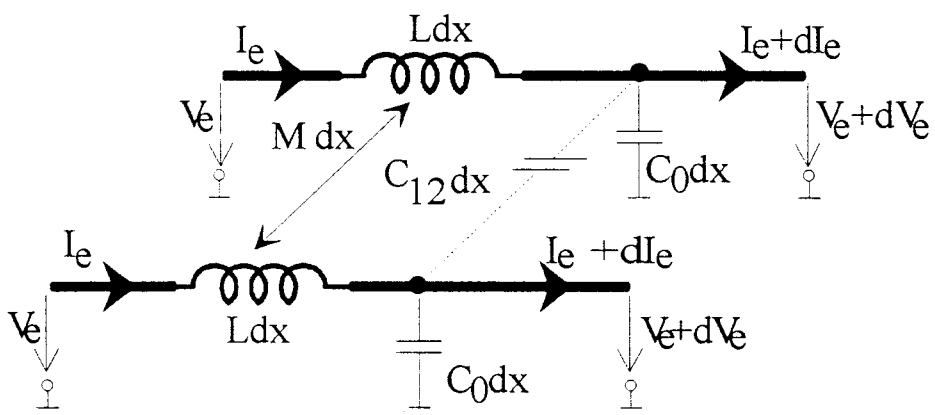


$$-\frac{\partial\left(V_a, V_b\right)}{\partial x} = \begin{bmatrix} L & M \\ M & L \end{bmatrix} \frac{\partial\left(I_a, I_b\right)}{\partial t}$$

$$-\frac{\partial\left(I_a, I_b\right)}{\partial x} = \begin{bmatrix} C_0 + C_{12} & -C_{12} \\ -C_{12} & C_0 + C_{12} \end{bmatrix} \frac{\partial\left(V_a, V_b\right)}{\partial t}$$

MODES APPROACH (e=even mode ; o=odd mode)

EVEN MODE:

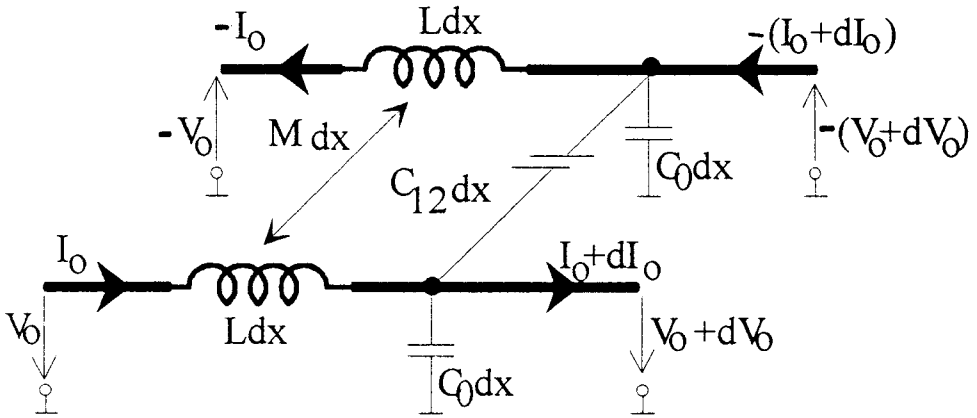


$$-\frac{dV_e}{dx} = (L+M)\frac{dI_e}{dx}; \qquad -\frac{dI_e}{dx} = C_0\frac{dV_e}{dt}$$

$$Z_{0e} = \sqrt{\frac{L+M}{C_0}} = Z_0\sqrt{\frac{1+k}{1-k}}$$

$$\frac{1}{c^2} = (L+M)C_0$$

ODD MODE:



$$-\frac{dV_o}{dx} = (L-M)\frac{dI_o}{dt}; \qquad -\frac{dI_o}{dx} = (C_0+2C_{12})\frac{dV_o}{dt}$$

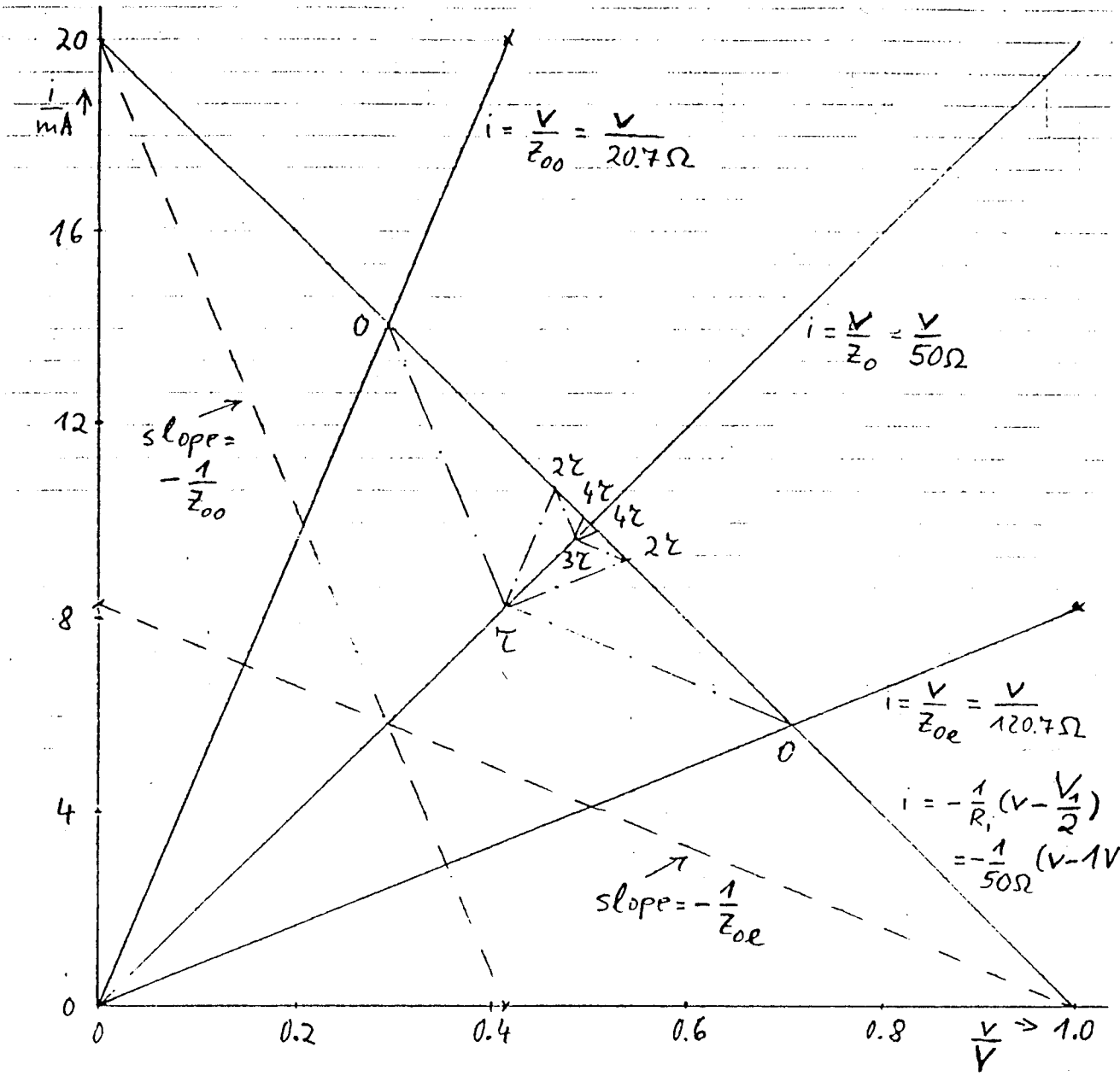
$$Z_{0o} = \sqrt{\frac{L-M}{C_0+2C_{12}}} = Z_0\sqrt{\frac{1-k}{1+k}}$$

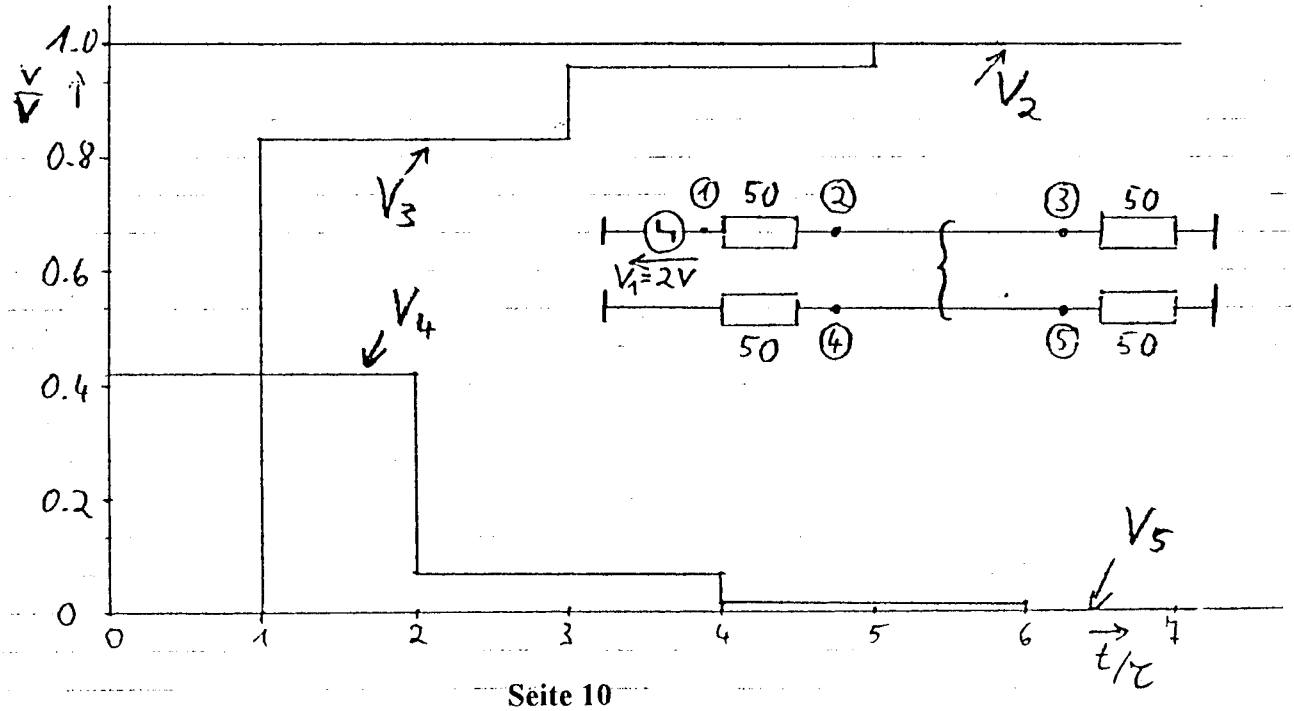
$$\frac{1}{c^2} = (L-M)(C_0+2C_{12})$$

$$\boxed{k = \frac{M}{L} = \frac{C_{12}}{C_0+C_{12}} = \frac{Z_{0e}-Z_{0o}}{Z_{0e}+Z_{0o}}}$$

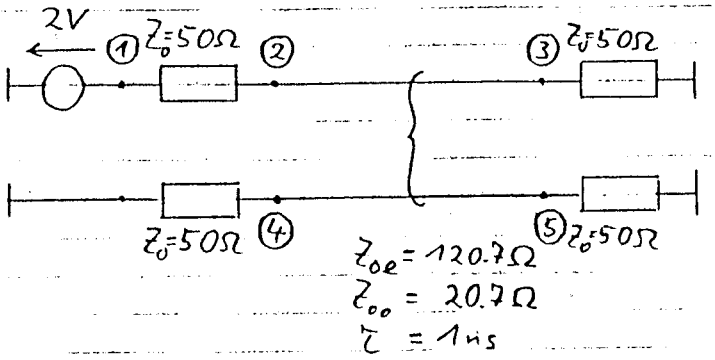
$$\boxed{Z_{0e}Z_{0o} = \frac{L}{C_0+C_{12}} = Z_0^2}$$

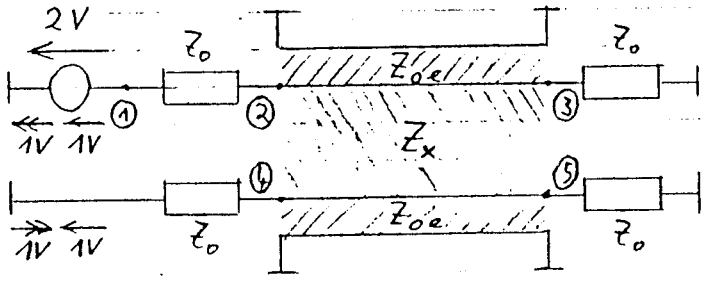
Graphical Determination of Step Responses with Coupled Lines (Bergeron's Method)



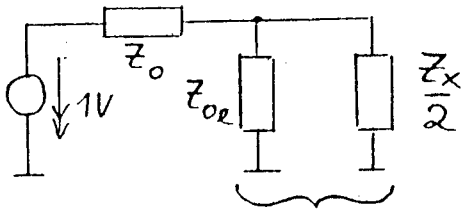


SPICE model for a 3dB-coupler





odd mode, t = 0 :



$$\doteq Z_{00} \rightarrow Z_x = 2\frac{Z_{0e}Z_{0o}}{Z_{0e}-Z_{0o}}$$

Inserting

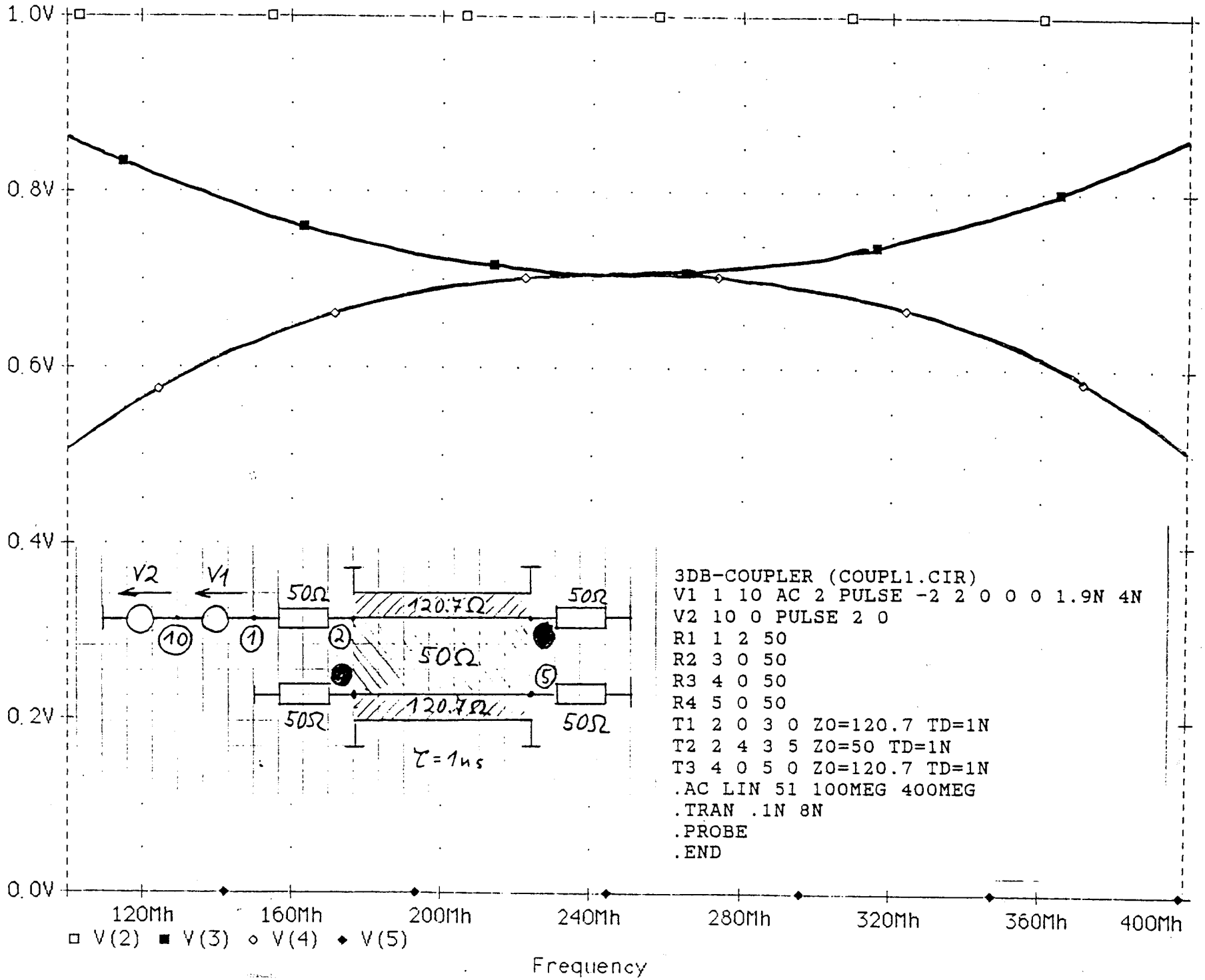
$$Z_{0e} = Z_0(\sqrt{2}+1)$$

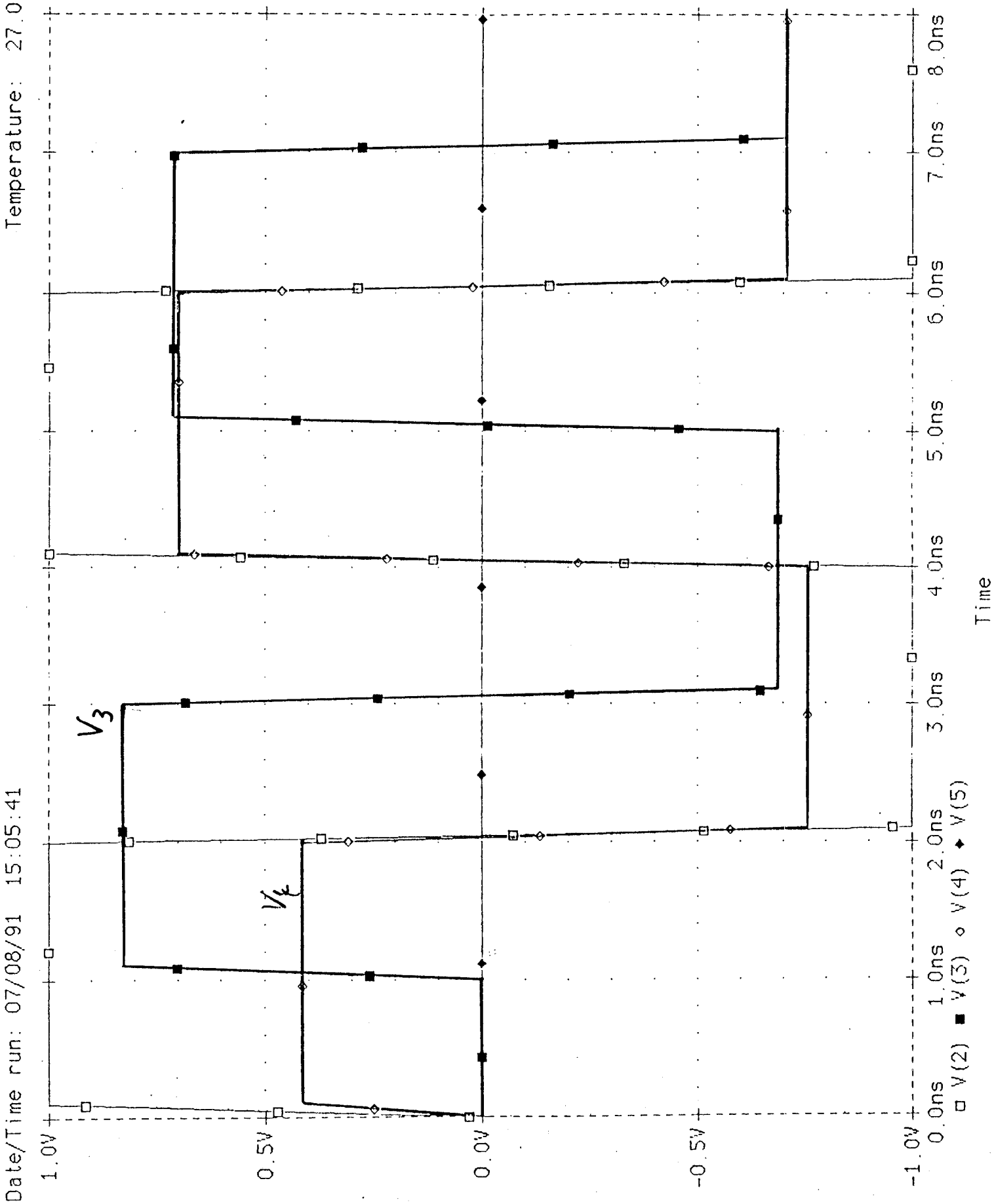
$$Z_{0o} = Z_0(\sqrt{2}-1)$$

$$Z_0 = 50\Omega$$

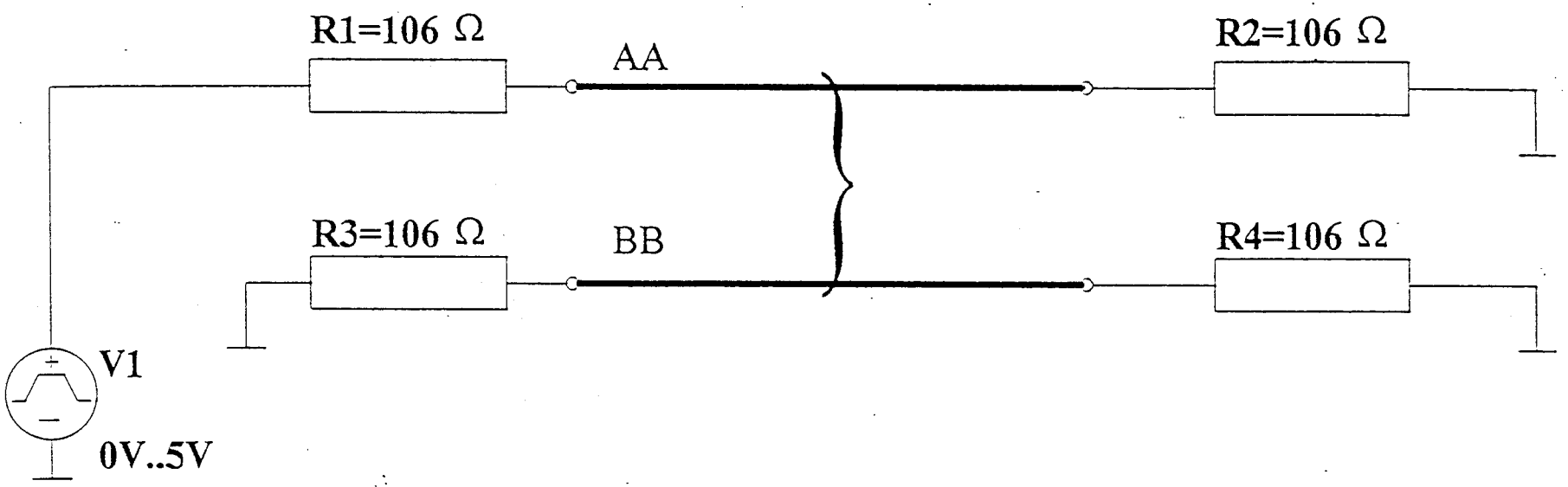
$$Z_x = 50\Omega$$

Date/Time run: 07/08/91 15:05:41 Temperature: 27.0




Temperature: 27.0
Date/Time run: 07/08/91 15:05:41
1.0V
0.5V
0.0V
-0.5V
-1.0V
0.0ns
1.0ns
2.0ns
3.0ns
4.0ns
5.0ns
6.0ns
7.0ns
8.0ns
Time
V3
V4
□ V(2) ■ V(3) ◇ V(4) ◆ V(5)

Bsp: Gekoppelte Leitungen:

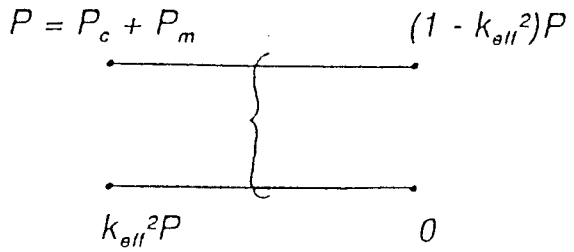


Von Polaris ermittelte Werte:

Select Signals								
Name	Coupling	Length	L(nH)	R	C (pF)	C_via	Z0	Tpd (ns)
BB	0.516718	1.268147	788,80	1.2710	69.15	2.00	106.8	7.385544
AA	0.515097	1.272312	791.42	1.2754185	69.35	2.00	106.83	7.748450

P.S.: Arguments for the missing factor 1/√2 in the coupling factor formula given in the Polaris-manual.

Given a directional coupler with power P at the input (midband!) and the asumption that half of the power is to be found in the electric field (P_c = 1/2 P) the other half in the magnetic field (P_m = 1/2 P).



$$= k_c^2 P_c + k_m^2 P_m$$

$$= k_c^2 \frac{P}{2} + k_m^2 \frac{P}{2}$$

$$= \frac{k_c^2 + k_m^2}{2} P$$

Therefore $$k_{eff} = \sqrt{\frac{k_c^2 + k_m^2}{2}} = \sqrt{\frac{\left(\frac{C_{12}}{C_0+C_{12}}\right)^2 + \left(\frac{M}{L}\right)^2}{2}}$$

If I have equal propagation velocities for the even and odd mode waves then

$$k_c = \frac{C_{12}}{C_0 + C_{12}} = \frac{M}{L} = k_m = k \rightarrow k_{eff} = k\ !$$

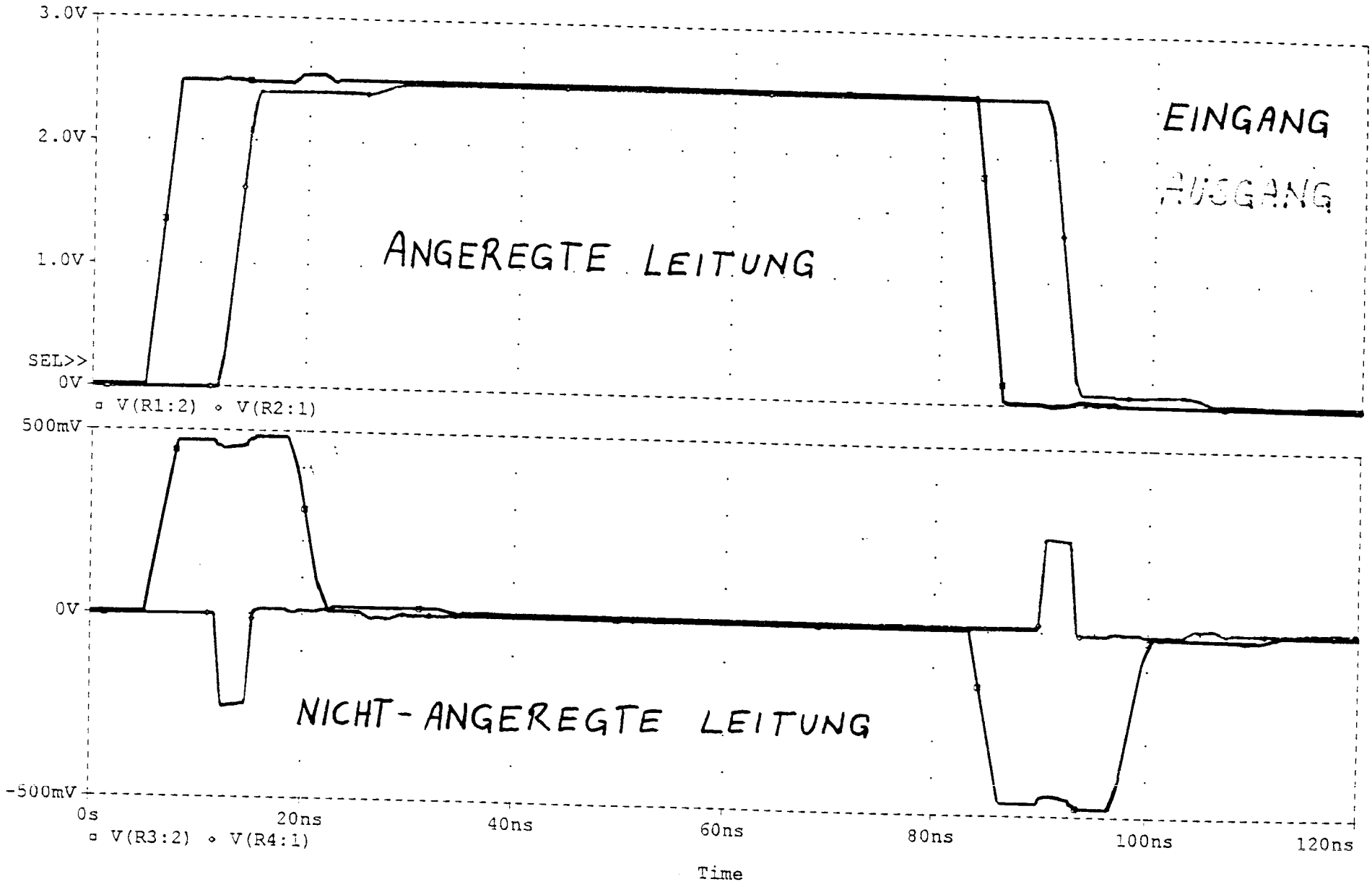
If C_{12} is only partly affected by the relative dielectric constant of the board material then $k_c \neq k_m$ and therefore

$$k_{eff} = \sqrt{\frac{k_c^2 + k_m^2}{2}}$$

If we have either pure capacitive or pure magnetive coupling the coupled power reduces by 3db compared to the fully coupled case which sounds resonable!

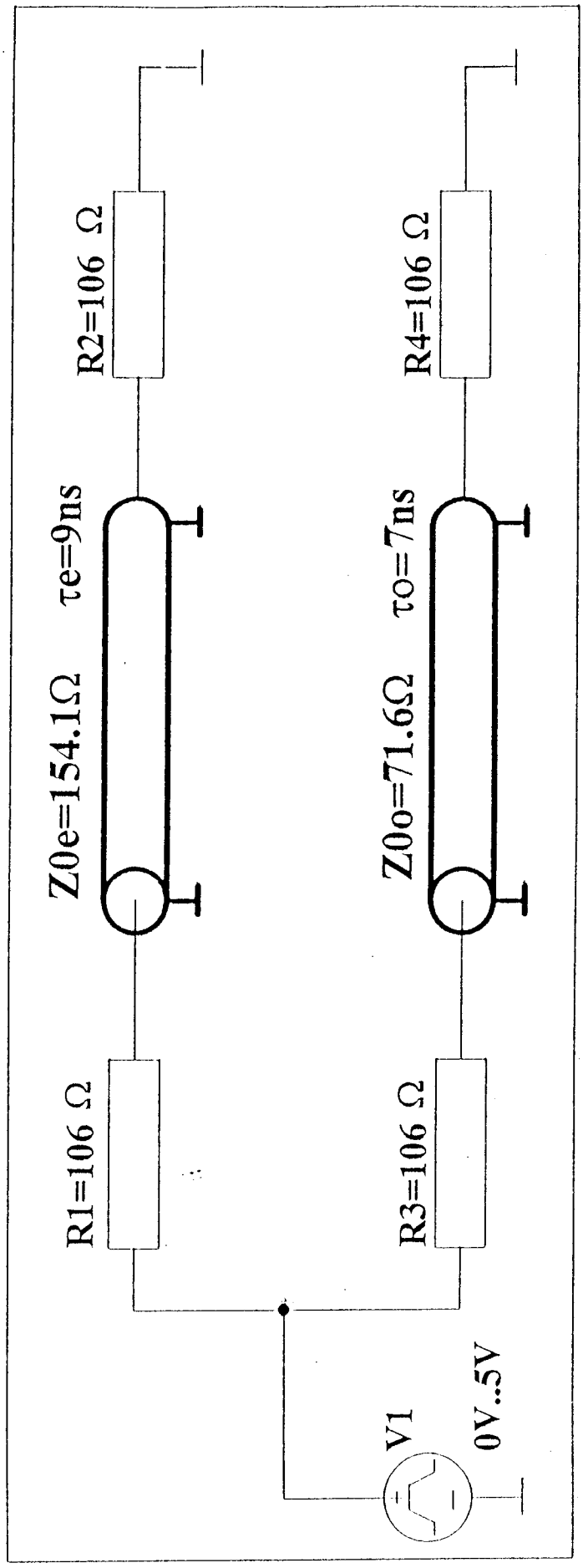
.TLN - File bei gekoppelten Leitungen

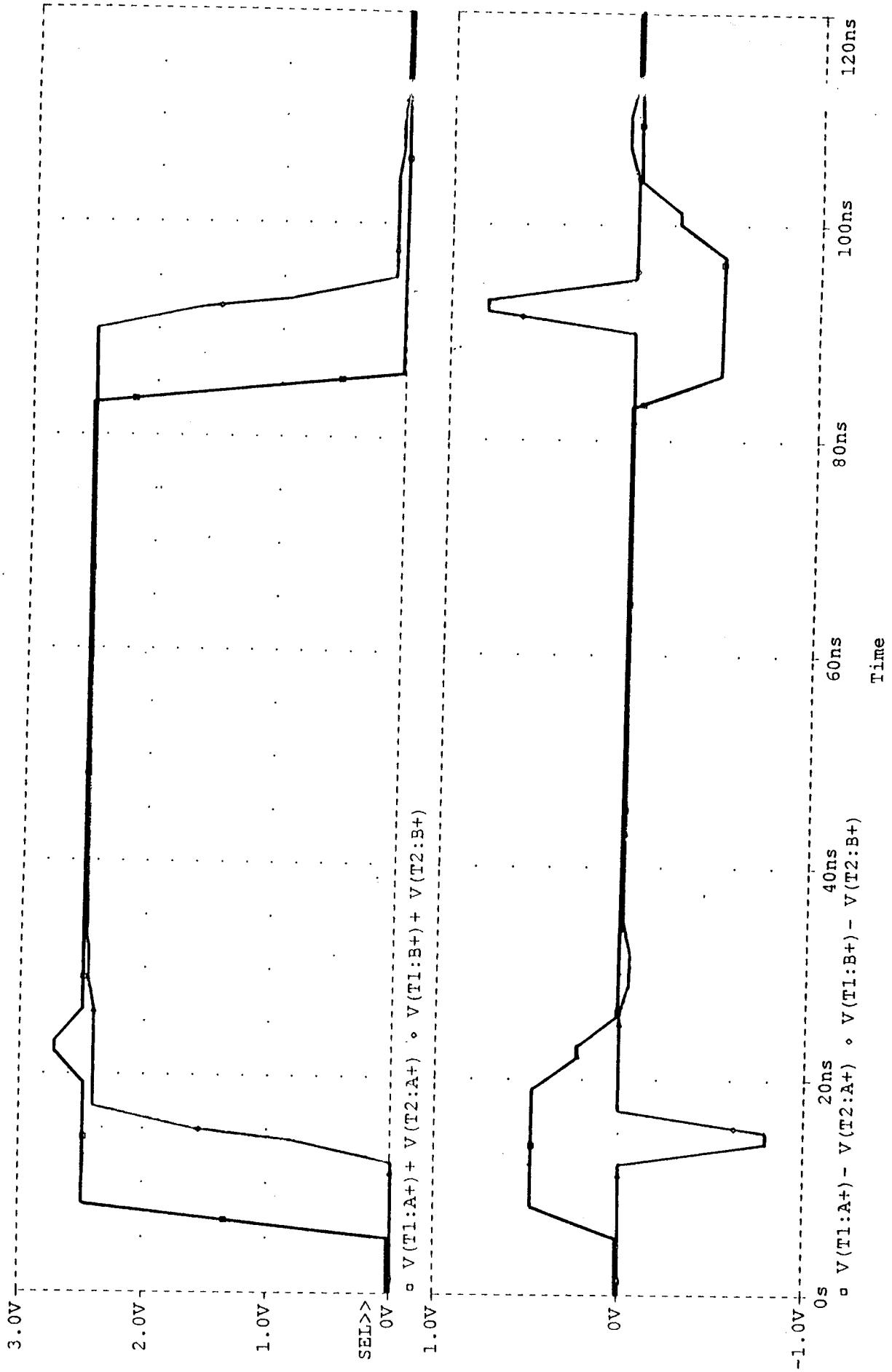
```
*POLARIS SPICE extractor

*AA
T1 1 0 2 0 LEN=0.014097 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T2 3 0 R1.2 0 LEN=0.002642 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T3 R2.1 0 4 0 LEN=0.003048 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T4 5 0 2 0 LEN=0.001041 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T5 6 0 1 0 LEN=0.001041 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T6 3 0 7 0 LEN=5.08e-05 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T7 7 0 5 0 LEN=0.625196 R=1.002258 L=622.148481NH G=0 C=54.392138PF
T8 4 0 6 0 LEN=0.625196 R=1.002258 L=621.674314NH G=0 C=54.855652PF
DEVICE R1.2 NAME=R1.2
DEVICE R2.1 NAME=R2.1
VIA R1.2 C=1PF
VIA R2.1 C=1PF

*BB
T9 8 0 9 0 LEN=0.012141 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T10 R3.2 0 10 0 LEN=0.002972 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T11 11 0 R4.1 0 LEN=0.002642 R=1.002258 L=628.941161NH G=0 C=47.855413PF
T12 10 0 9 0 LEN=0.625196 R=1.002258 L=622.148481NH G=0 C=54.392138PF
T13 11 0 8 0 LEN=0.625196 R=1.002258 L=621.674314NH G=0 C=54.855652PF
DEVICE R4.1 NAME=R4.1
DEVICE R3.2 NAME=R3.2
VIA R4.1 C=1PF
VIA R3.2 C=1PF

*coupling information
KT12T7 T12 T7 Lm=250.443047NH Cm=17.699673PF
KT8T13 T8 T13 Lm=255.641173NH Cm=18.328567PF

*end of SPICE description
```


3.0V
2.0V
1.0V
SEL>>
0V
V(R1:2)
V(R2:1)
EINGANG
AUSGANG
ANGEREGTE LEITUNG
500mV
0V
-500mV
NICHT-ANGEREGTE LEITUNG
0s
20ns
40ns
60ns
80ns
100ns
120ns
V(R3:2)
V(R4:1)
Time

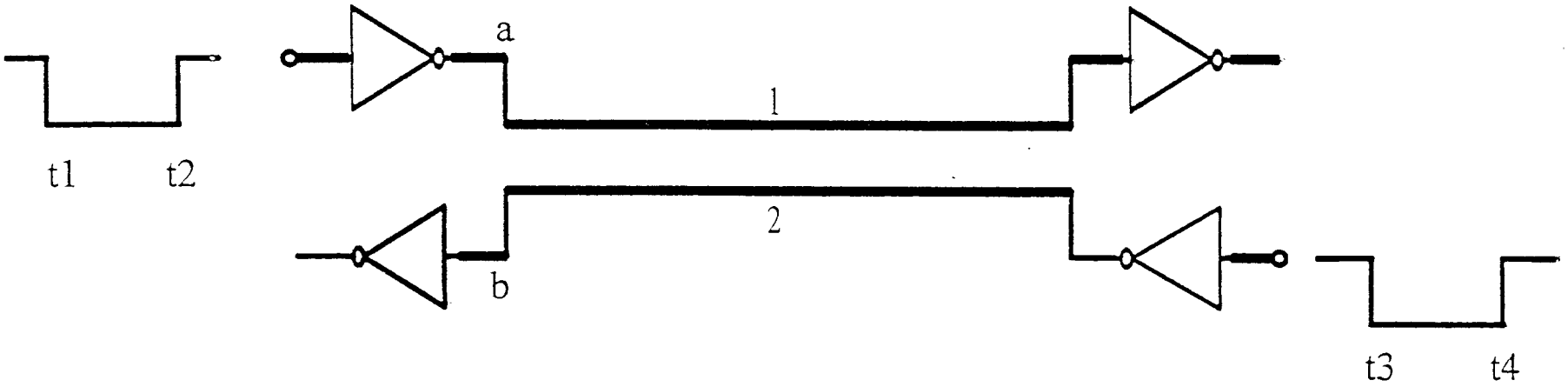
Vorgegebenes Board von Polaris

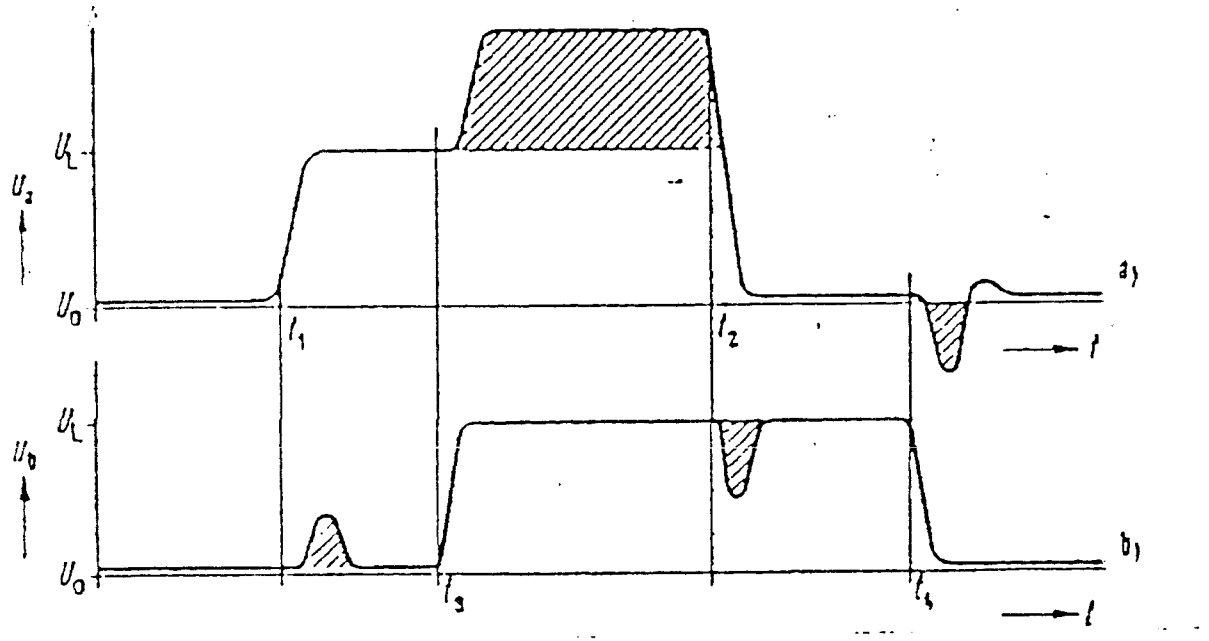
PSPICE Simulation des Even-/Odd-Modes



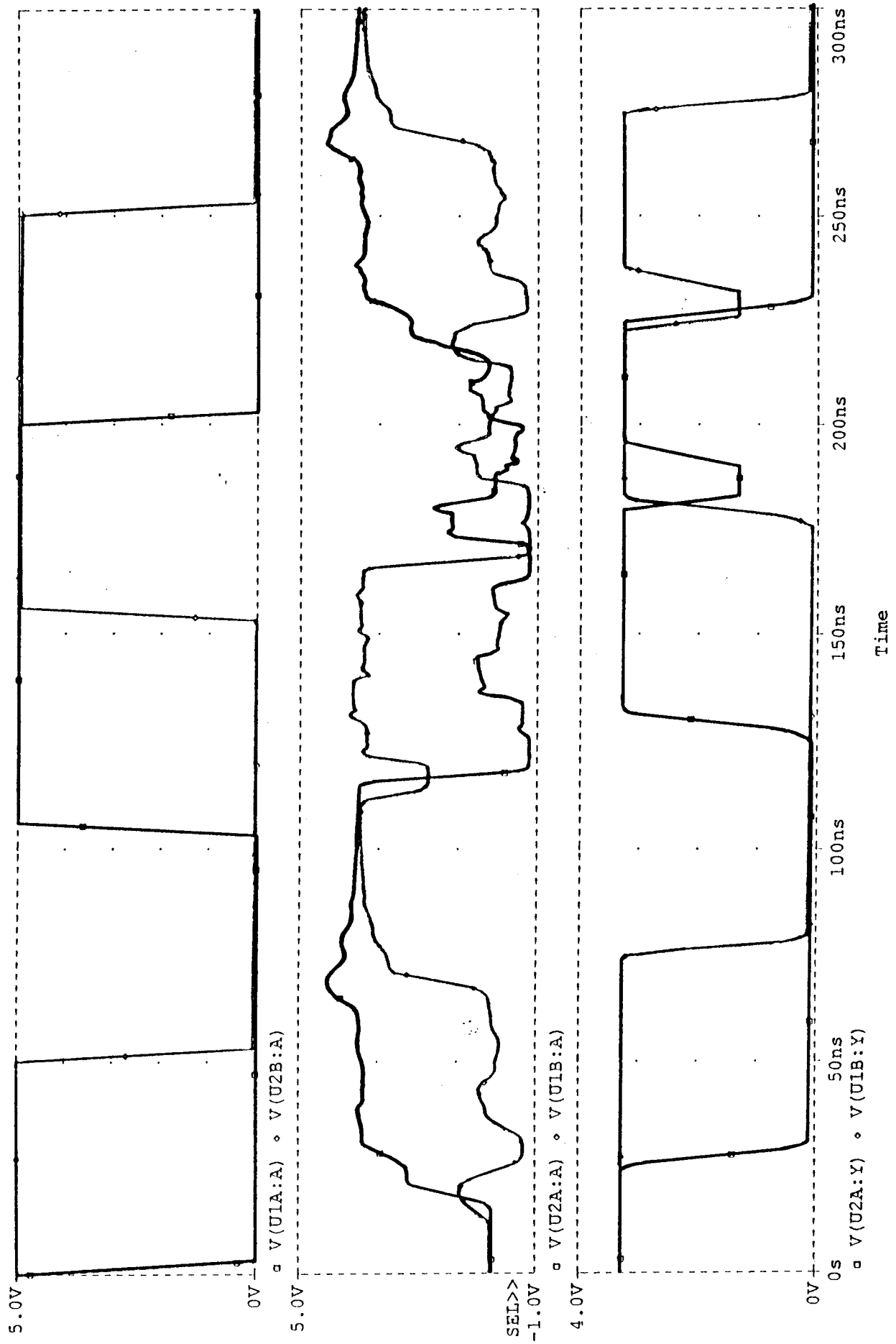

3.0V
2.0V
1.0V
SEL>>
0V
1.0V
0V
-1.0V
0s
20ns
40ns
60ns
80ns
100ns
120ns
Time
V(T1:A+) + V(T2:A+) ∘ V(T1:B+) + V(T2:B+)
V(T1:A+) - V(T2:A+) ∘ V(T1:B+) - V(T2:B+)

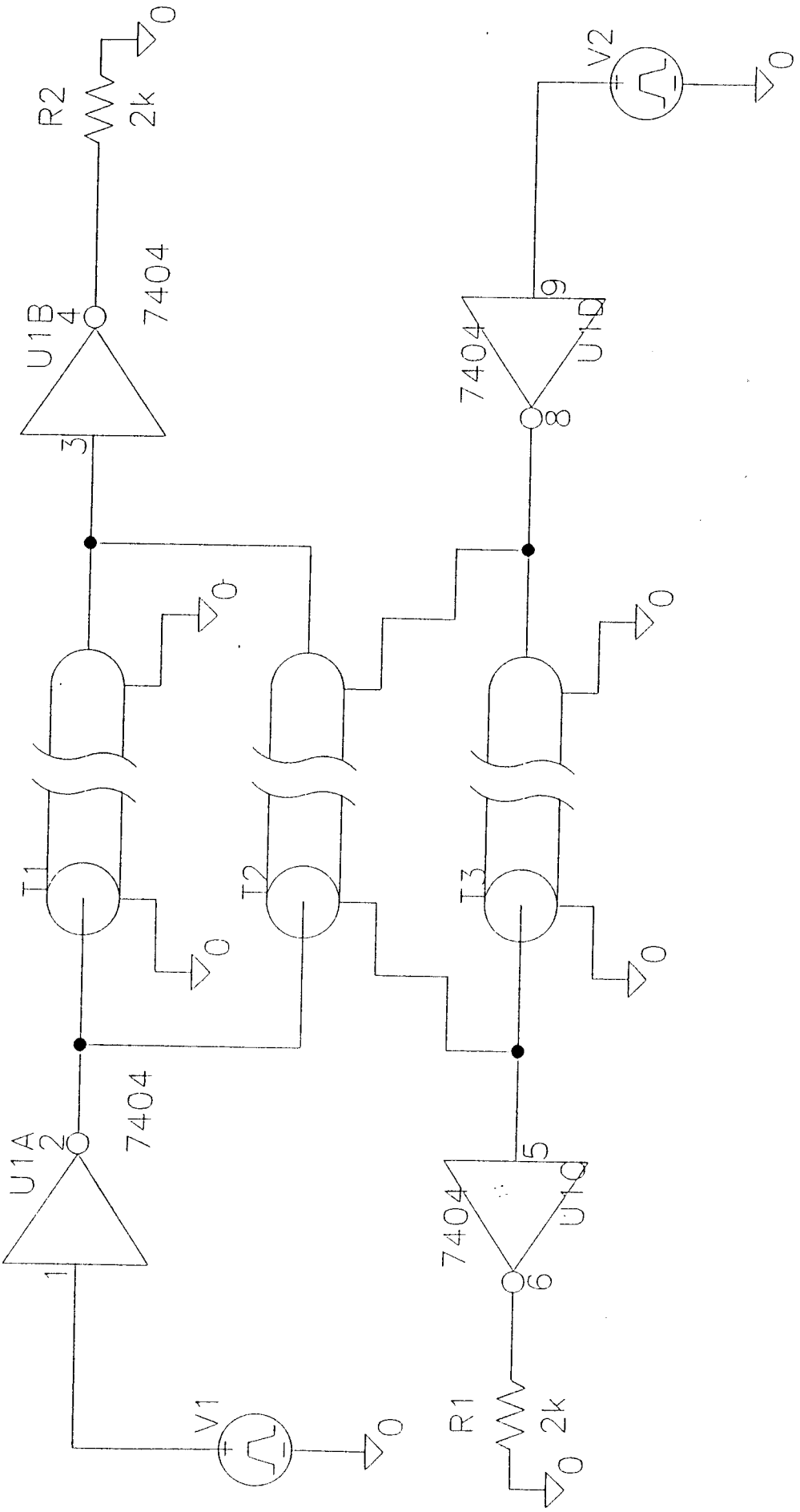
Gegensprechen antiparallel verlaufender Leitungen

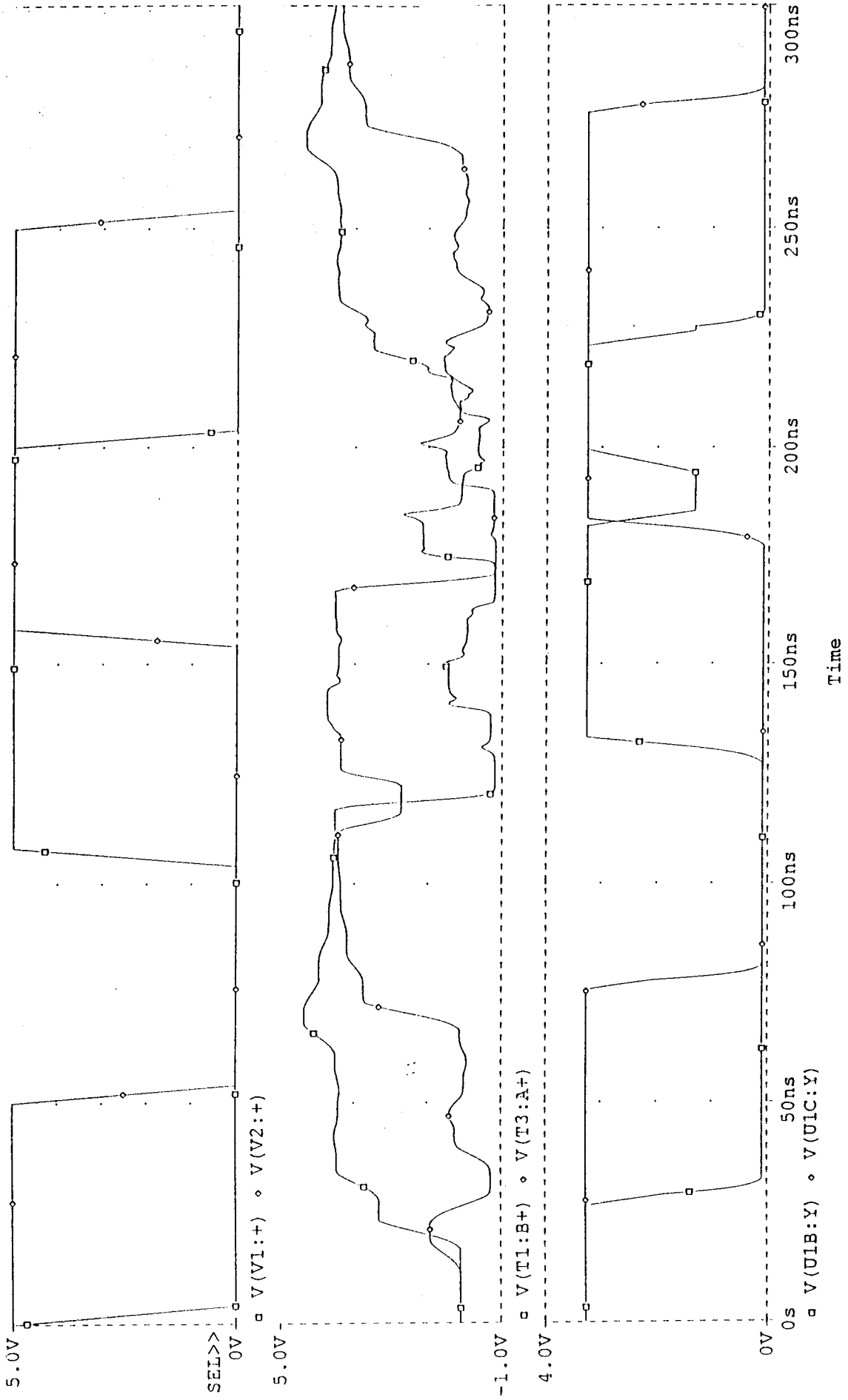




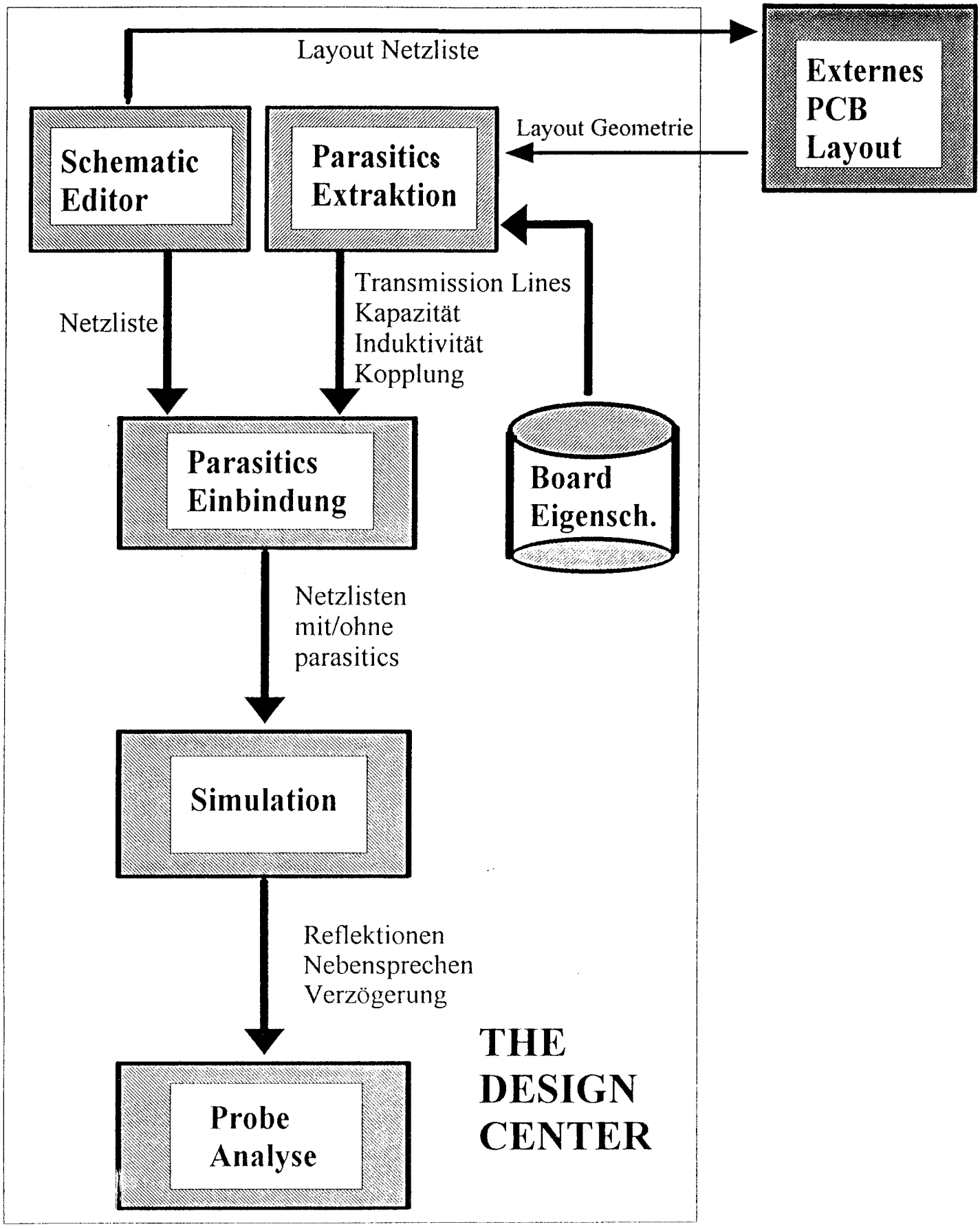
Spannungsverläufe beim antiparallelen Gegensprechen.a)und b) bezieht sich auf die gleich bezeichneten Punkte im obigen Bild


5.0V
0V
V(U1A:A)
V(U2B:A)
5.0V
SEL>>
-1.0V
V(U2A:A)
V(U1B:A)
4.0V
0V
0s
50ns
100ns
150ns
200ns
250ns
300ns
V(U2A:Y)
V(U1B:Y)
Time


R2
2k
0
7404
U1B
4
3
V2
9
U1D
8
T1
T2
T3
U1A
2
1
5
U1C
6
V1
R1
2k


5.0V
SEL>>
0V
V(V1:+)
V(V2:+)
5.0V
-1.0V
V(T1:B+)
V(T3:A+)
4.0V
0V
V(U1B:Y)
V(U1C:Y)
0s
50ns
100ns
150ns
200ns
250ns
300ns
Time

Ablauf einer Simulation mit Polaris



Test eines Signalprozessors

A.Gauckler, W.Rülling
Mikrosystemtechnik / IIT
Fachhochschule Furtwangen

Kurzfassung

Für einen Sea-of-Gates Chip zur Berechnung von Faltungsintegralen werden Testmustersätze erstellt und es wird die erreichbare Fehlerüberdeckungsrate ermittelt. Die prinzipiell nicht entdeckbaren Fehler werden vorgestellt. Schließlich werden verschiedene verwendete Testumgebungen und die mit ihnen erzielten Ergebnisse präsentiert.

1 Einführung

Am Institut für Innovation und Transfer (IIT) der Fachhochschule Furtwangen wurde im Projekt *Algorithmen für intelligente Sensoren* ein Prozessor zur Berechnung von Faltungsintegralen entworfen. Über diese Arbeit wurde bereits mehrfach auf Workshops des MPC-FH Verbundes vorgetragen (siehe [1], [2], [3]). Dabei wurde das zu lösende Anwendungsproblem, das verwendete Lösungskonzept und schließlich die Layoutrealisierung vorgestellt.

Inzwischen wurde ein Prototyp des Prozessors in Sea-of-Gates Technik am IMS (Institut für Mikroelektronik in Stuttgart) gefertigt. Einige technische Daten des Chips und ein Chipfoto sind in [4] wiedergegeben. In der vorliegenden Arbeit soll es nun um den Test der gefertigten Chips gehen. Ein erster (unvollständiger) Funktionstest wurde bereits vor der Auslieferung durch das IMS durchgeführt, um offensichtlich fehlerhafte Exemplare auszusortieren. Die schließlich ausgelieferten Chips wurden dann an der FH Furtwangen wesentlich gründlicheren Tests unterzogen. Diese Testarbeiten sollen im folgenden vorgestellt werden.

Zunächst wird kurz auf die Funktionsweise und auf Besonderheiten des untersuchten Chips eingegangen. Danach wird die Testproblematik erläutert und als erstes die Ermittlung geeigneter Testmuster behandelt. Die verfügbaren automatischen Testmustergeneratoren konnten aufgrund der Komplexität des Chips leider nicht verwendet werden. Anschließend wird die für den gegebenen

Chip erreichte Fehlerüberdeckungsrate diskutiert und es wird vorgestellt, welche Fehler sich prinzipiell nicht nachweisen lassen.

Nach den mehr theoretischen Vorüberlegungen werden verschiedene Testumgebungen und die jeweils mit ihnen erzielten Testergebnisse vorgestellt.

2 Kurzvorstellung des Signalprozessors

Wie bereits in [1] dargestellt, soll der vorliegende Signalprozessor zur Meßwertkorrektur bei miniaturisierten Sensoren verwendet werden. Dabei wird davon ausgegangen, daß das Übertragungsverhalten des Sensors näherungsweise durch ein lineares Netzwerk dargestellt werden kann.

Die Aufgabe des Signalprozessors besteht dann darin, aus den als Sensorausgabe gelieferten Werten $\boldsymbol{y}$ die tatsächlichen physikalischen Eingangsgrößen $\boldsymbol{x}$ durch eine Faltungsintegration zu berechnen. Dazu soll er Prozessor eine Folge $y_0, y_1, y_2 \ldots y_n$ von Meßwerten sequentiell einlesen und jeweils nach dem Wert y_i den Wert

$$x_i = \sum_{j=0}^{i} y_i \cdot g_{i-j}$$

ausgeben. Der Vektor $g_0, g_1, g_2 \ldots g_n$ stellt dabei das inverse Übertragungsverhalten des Sensors dar.

Auf dem Chip werden die Vektoren $\boldsymbol{y}$ und $\boldsymbol{g}$ in zyklischen Schieberegistern gespeichert, so daß sich jeweils die miteinander zu multiplizierenden Werte an einem Rechenwerk treffen. In Abbildung 1 ist dieses Prinzip skizziert.

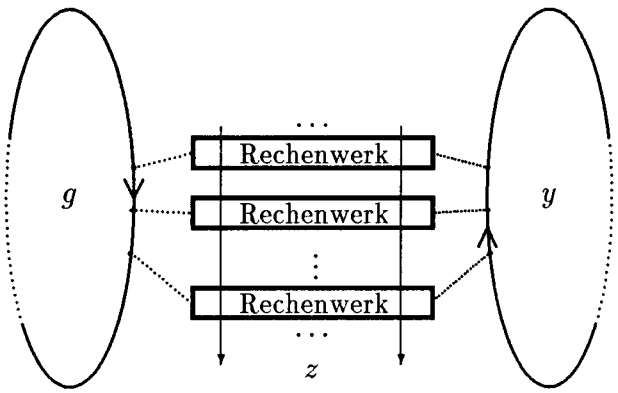


Abbildung 1: Prinzip des Rechenwerks

Die Schieberegister bestehen aus verschiedenen Falten, die wahlweise über Multiplexer zugeschaltet werden können, um verschiedene Registerlängen zu konfigurieren. In der Anwendung wird auf diese Weise die Anzahl der bei der Faltungsintegration zu berücksichtigen Stützstellen eingestellt.

Bei der gewählten Prototyp-Implementierung sind beide Schieberegister für ***g*** und ***y*** seriell realisiert. Ein drittes zyklisches Schieberegister ***z*** wird benutzt, um ein Synchronisationssignal für den Datenzugriff zu erzeugen. Auf diese Weise kann der Chip wahlweise mit einer Datenwortlänge von 8 Bit oder 16 Bit arbeiten. Aus Platzgründen wurden sämtliche Schieberegister statt in statischer Logik in dynamischer Logik realisiert ([2]). Dies ist für die Testproblematik von besonderer Bedeutung. Erstens können die Tests aufgrund der dynamischen Vorgänge nicht beliebig langsam durchgeführt werden und zweitens kann die Testbarkeit der Schaltung nicht wie sonst bei sequentiellen Schaltungen üblich mit Hilfe eines Scan-Path verbessert werden. Auf die Testbarkeit wird im folgenden noch genauer eingegangen.

Das Verhalten des Chips kann außer an den Ergebnisausgaben auch durch drei Testausgänge g_{out}, y_{out} und z_{out} beobachtet werden. Diese Ausgänge entsprechen Abgriffen der drei zyklischen Schieberegister und liefern damit die sequentielle Ausgabe der jeweiligen Schieberegisterinhalte.

3 Testbarkeit des Signalprozessors

Von den erwähnten Testausgängen abgesehen wurden auf dem Chip keine besonderen Vorkehrungen zur Unterstützung von Tests getroffen, da davon ausgegangen wurde, daß der Prozessor aufgrund seiner besonderen Struktur auch ohne Zusatzhardware einfach zu testen sein sollte. Der Test des Rechenwerks kann prinzipiell dadurch erfolgen, daß man verschiedene geeignet gewählte Beispieldaten eingibt und die Korrektheit der berechneten Ergebnisse überprüft. Tatsächlich konnte in [5] gezeigt werden, daß man im wesentlichen nur drei Multiplikationen durchführen muß, um das Rechenwerk fast vollständig zu testen. Da die eingegebenen Daten durch die Schieberegister geshiftet werden, wird gleichzeitig auch deren Funktion überprüft.

Diese Überlegungen zeigen, daß der Chip insofern gut testbar ist, als daß man nur relativ wenige Beispielrechnungen durchführen lassen muß, um seine Korrektheit zu überprüfen. Die erforderlichen Testmusterdaten haben daher eine recht einfache Struktur.

Andererseits ist der Chip insofern schlecht testbar, als daß man aufgrund seiner großen sequentiellen Tiefe mit sehr langen Testmusterfolgen arbeiten muß. Beispielsweise muß eine Schieberegisterfalte der Länge ***n*** dadurch initialisiert werden, daß sie sequentiell mit ***n*** Nullen gefüllt wird. Dies kann allerdings für sämtliche Falten gleichzeitig durchgeführt werden. Da die längste Registerfalte aus 512 Stufen besteht, dauert die Initialisierung des Chips 512 Takte der 2-Phasen-Clock. Einen ähnlichen Zeitaufwand muß man investieren, um die Beispieldaten einzugeben und auf die Ergebnisse zu warten.

4 Testmustergenerierung

Zum Ermitteln von Testmustern geht man von einem idealisierten Fehlermodell aus. Statt alle in der Praxis denkbaren Fehlertypen zu berücksichtigen, testet man die Chips nur auf das Vorkommen von Fehlern aus dem verwendeten Fehlermodell. Typischerweise verwendet man das *Stuck-at-Einzelfehlermodell*, bei dem sich ein Fehler immer so äußert, daß genau eine Leitung der Schaltung auf dem logischen Pegel 0 oder 1 festsitzt. Dies ist zwar eine recht willkürliche Annahme, aber es wird erwartet, daß ein vollständiger Test auf Stuck-at-Einzelfehler so gründlich ist, daß er auch eventuell auftretende andere Fehler entdeckt.

Bei den automatischen Methoden zur Generierung von Testmustern unterscheidet man grundsätzlich zwei verschiedene Typen von Algorithmen.

1. Beim *D-Algorithmus* wird versucht, für einen vorgegebenen einzelnen Fehler des vorgegebenen Fehlermodells geeignete Eingaben zu konstruieren, bei denen sich der Fehler durch fehlerhafte Ausgaben der Schaltung bemerkbar macht. Bereits bei rein kombinatorischen Schaltungen braucht man bei dieser Vorgehensweise im allgemeinen einen exponentiellen Zeitaufwand in Abhängigkeit von der Schaltungsgröße. Bei sequentiellen Schaltungen wächst die Laufzeit zudem exponentiell in der Länge der benötigten Testmusterfolge.

2. Bei der *Fehlersimulation* geht man so vor, daß man für willkürliche Eingaben untersucht, welche Fehler sich durch sie entdecken lassen. Auf diese Weise kann man mit relativ geringem Zeitaufwand viele geeignete Testmuster konstruieren.
   Allerdings erreicht man dabei selbst bei rein kombinatorischen Schaltungen keine besonders hohe Fehlerüberdeckungsrate, weil es im Laufe der Zeit immer unwahrscheinlicher wird, durch zufällige Eingaben neue Testmuster zu finden.

In der Praxis ist es deshalb sinnvoll, zunächst mit Hilfe der Fehlersimulation Testmuster für die "einfach zu entdeckenden"

Fehler zu konstruieren und anschließend mit dem D-Algorithmus Muster für die verbleibenden "schwierigen" Fehler ermitteln zu lassen.

Typischerweise wird die skizzierte Vorgehensweise im ASIC-Entwurf eingesetzt, wenn man die zu untersuchende Schaltung zusätzlich mit Hilfe eines *Scan-Path* testbar gemacht wurde. Der Scan-Path führt dazu, daß die Zustände sämtlicher Speicherelemente extern beobachtbar und setzbar werden. Die Testmustergenerierung kann sich dadurch auf die Betrachtung der kombinatorischen Bestandteile der Schaltung beschränken.
Da im vorliegenden Fall aus Platzgründen auf dem Chip kein Scan-Path realisiert wurde, muß die Testmustergenerierung für eine Schaltung großer sequentieller Tiefe durchgeführt werden. Für die skizzierten Algorithmen bedeutet das eine derart hohe zu erwartende Rechenzeit, daß die Verfahren nicht mehr praktikabel sind.

Im Projekt wurden die Testmuster deshalb ohne spezielle Generatoren ermittelt. Dazu wurden zunächst ohne Rechnereinsatz Vorschläge für sinnvoll scheinende Beispieleingaben des Chips erarbeitet [5]. Diese Beispieldaten wurden zunächst auf einer relativ abstrakten Darstellungsebene formuliert (siehe Abbildung 2) und dann mit Hilfe einer Schaltungssimulation in eine Testmusterdatei überführt.

```
init_registers(2);
set_bit(zfalte,1);
wait_periods(16);
append_g( 32 );          { g[0] := 32 }
append_y( 16 );          { y[0] := 16 }
append_g(  2 );          { g[1] :=  2 }
append_y(  4 );          { y[1] :=  4 }
waiting_for_results(10);
```

Abbildung 2: Beispiel einer PASCAL-Notation

Mit einer anschließenden Fehlersimulation wurde die erreichte Fehlerüberdeckungsrate ermittelt und es wurden die bisher nicht entdeckbaren Fehler protokolliert. Durch manuelle Analyse der verbliebenen Fehlermenge wurden solange zusätzliche Testmuster konstruiert, bis schließlich eine ausreichende Fehlerüberdeckung erreicht wurde. (Abbildung 3)

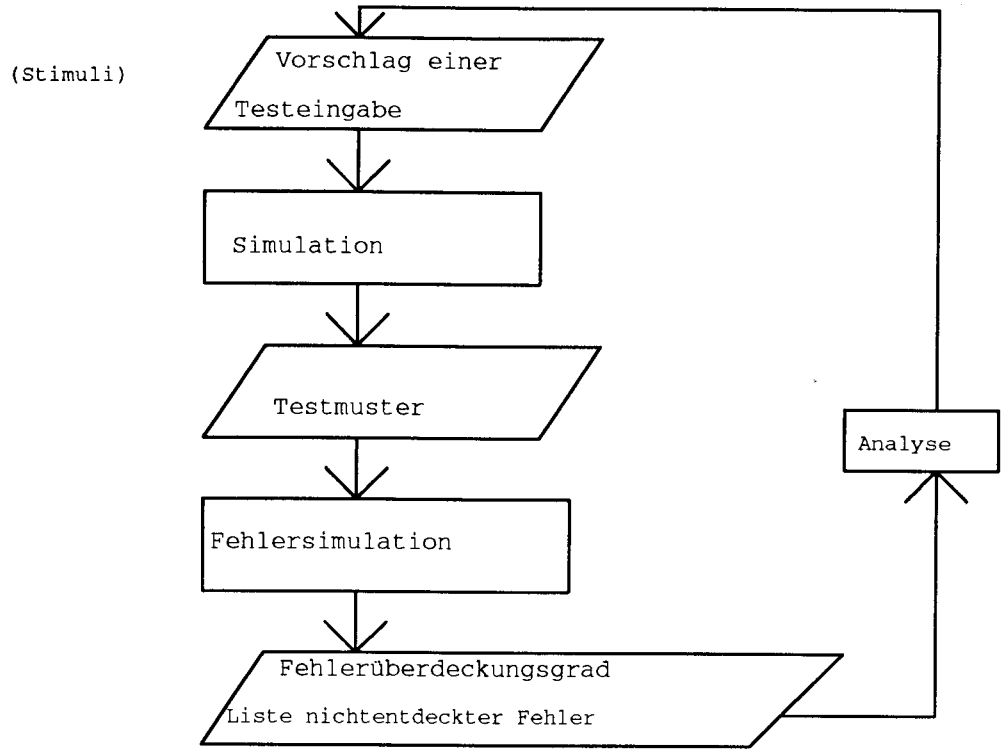


Abbildung 3: Ablaufskizze!

Bei dieser Vorgehensweise wurde die Fehlersimulation also nicht direkt zur Konstruktion von Testmustern verwendet, sondern nur zur Überprüfung bereits vorliegender Testmuster. Für diese Aufgabe wurde auf einer SPARC 10 eine Rechenzeit von 134 Stunden benötigt. Dies bestätigt die ursprüngliche Einschätzung, daß eine vollautomatische Testmustergenerierung nicht durchführbar ist.

Das Ziel der Testmustererzeugung ist, eine 100-prozentige Fehlerüberdeckung zu erreichen. Das bedeutet, das jeder nach dem zugrundeliegenden Fehlermodell mögliche Fehler durch die konstruierten Testeingaben entdeckt wird. In der Praxis gibt es jedoch einige Gründe, warum man sich oft auch mit geringeren Fehlerüberdeckungsraten zufrieden gibt:

1. Die Rechenzeit zur automatischen Generierung einer vollständigen Testmustermenge ist zu hoch.

2. Aufgrund von redundanten Schaltungsteilen, sind manche Fehler grundsätzlich nicht entdeckbar. (Da diese Fehler nie zu fehlerhaften Ergebnissen führen, sollte man sie eigentlich beim Fehlerüberdeckungsgrad nicht mitzählen)

3. Um die Testzeit zu beschränken, sollen keine zu umfangreichen Tests durchgeführt werden.

4. Da sich die Fehlerüberdeckungsrate nur auf das idealisierte Fehlermodell bezieht und nicht auf die in der Praxis tatsächlich auftretenden Fehler, hat man selbst bei einer 100%-igen Fehlerüberdeckung keine Garantie, alle in der Praxis auftretenden Fehler zu erfassen.
Beispielsweise wird im Fehlermodell oftmals nicht zwischen den unterschiedlichen Transistorschaltungen einer NMOS- oder CMOS-Realisierung der Gatter unterschieden. Außerdem werden bei der Sea-of-Gates Technik typischerweise keine Fehler an Transistoren untersucht, die für die Gate-Isolation beschaltet werden, u.s.w.

Die Untersuchung der beim vorliegenden Signalprozessor von den Testmustern nicht abgedeckten Einzel-Stuck-at-Fehler ergab, daß der Prozessor tatsächlich redundante Schaltungsteile enthält, die zu nicht entdeckbaren und damit irrelevanten Fehlermöglichkeiten führen. Von diesen Fehlern abgesehen ist der verwendete Test vollständig. Im folgenden sollen einige Beispiele für beabsichtigte und unbeabsichtigte Schaltungsredundanz des Signalprozessors dargestellt werden.

5 Nicht entdeckbare Fehler

Der am häufigsten auftretende nicht entdeckbare Fehlertyp betrifft dynamischen Schieberegisterzellen. Eine halbe Registerzelle besteht aus einem Transmission-Gate und einem Inverter (siehe Abbildung 4).

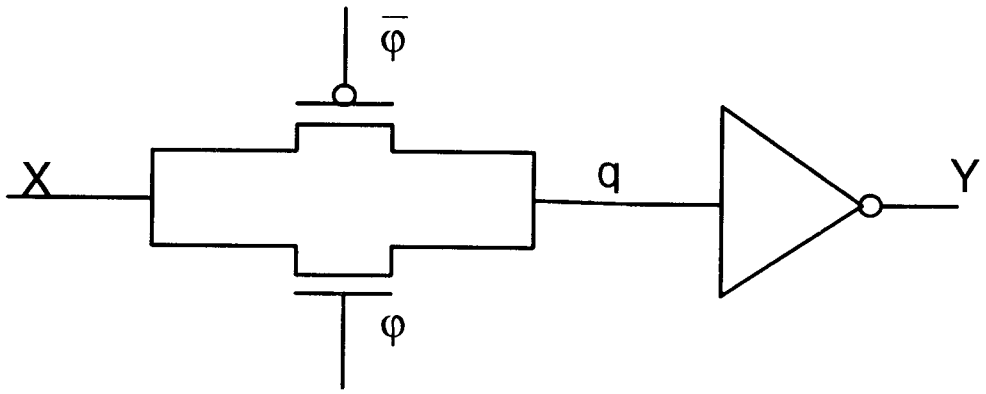


Abbildung 4: Halbe Schieberegister Zelle

Mit dem Signal $\varphi = 1$ bzw. $\bar{\varphi} = 0$ lädt das Eingangssignal x den internen Knoten q auf und man erhält am Ausgang den Wert $y = \bar{x}$.
Im Fall $\varphi = 0$ bzw. $\bar{\varphi} = 1$ sperren die beiden Transistoren des Transmission-Gates und der Ausgangswert y bleibt aufgrund der dynamischen Speicherung am Knoten q noch für einige Zeit erhalten.

Wenn nun einer der beiden Transistoren des Transmission-Gates aufgrund eines Fehlers ständig sperrt, dann führt das dazu, daß der Knoten ***q*** langsamer geladen oder entladen wird, die Schaltung aber ihre prinzipielle Funktion weiterhin erfüllt. Für die für die Anwendung vorgesehenen Taktraten der Schaltung wird sich ein solcher Fehler nicht bemerkbar machen, der Fehler ist also nicht entdeckbar.
Diese Redundanz war bereits beim Schaltungsentwurf bekannt und wurde zur Erhöhung der Zuverlässigkeit der Schieberegisterzellen als sinnvoll erachtet.

Dagegen ist die im folgenden beschriebene Schaltungsredundanz nicht beabsichtigt. Sie wurde erst nachträglich aufgrund der Auswertung der Fehlersimulation entdeckt. Obwohl die korrekte Funktion des Chips nicht beeinträchtigt ist, soll diese Redundanz bei einem Redesign der Schaltung beseitigt werden.

Das Rechenwerk des Chips enthält einen aus 34 Volladdierern bestehenden seriellen Addierer (s.a. Abbildung 5), der die Aufgabe hat, die nacheinander eintreffenden parallelen Datenwörter aufzusummieren. Dazu werden die entstehenden Summenbits und Übertragsbits jeweils mit einer Schieberegisterzelle zwischengespeichert.

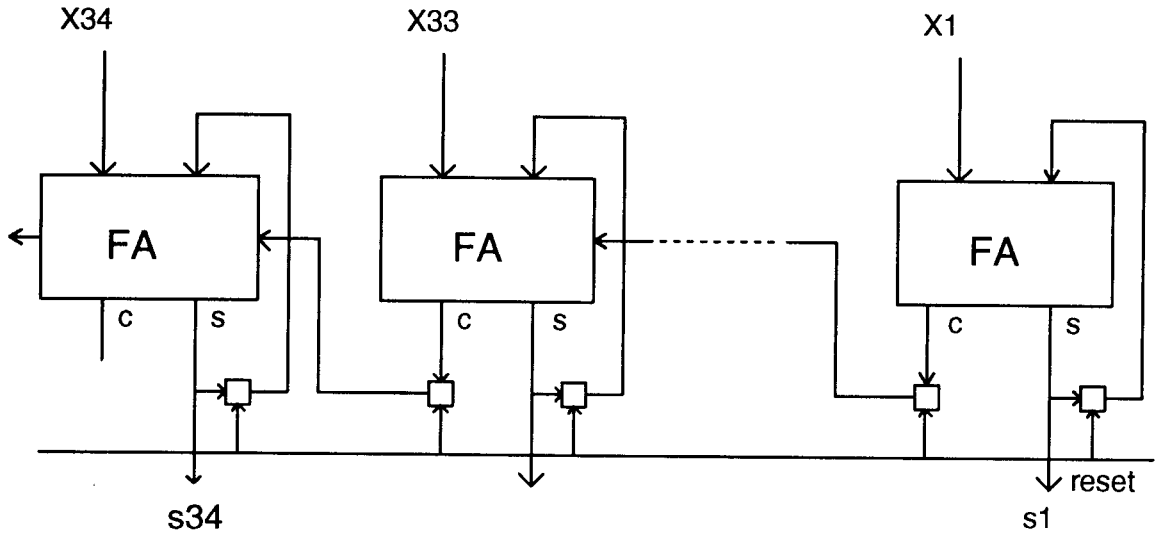


Abbildung 5: Volladdierer

Man beachte, daß der Addierer in jedem Takt einen neuen Summanden aufnehmen kann, die Überträge aber pro Takt jeweils nur um eine Stelle weiterverarbeitet werden. Um den Addierer initialisieren zu können, wurden die Speicherzellen mit einem zusätzlichen ***reset***-Eingang ausgestattet. Auf diese Weise kann man in einem Takt sämtliche Summen- und Übertragsbits auf den Wert 0 setzen.

Die Analyse der nicht entdeckten Fehlertypen hat nun überraschenderweise gezeigt, daß die ***reset***-Eingänge der Übertrags-

speicher nicht auf stuck-at-0 Fehler testbar sind. Der Grund liegt darin, daß der Addierer im Innern eines Multiplizierers verwendet wird und seine Ein- und Ausgänge nicht direkt von außen gesetzt und beobachtet werden können. So sind die einzelnen Summanden beispielsweise die Partialprodukte einer Multiplikation und die Ergebnisdaten werden erst nach Beendigung der vollständigen Multiplikation weitergegeben. Dieser Anwendung entsprechend kann das ***reset***-Signal nur bei Beginn einer neuen Multiplikation und nach Abschluß einer alten Multiplikation ausgelöst werden. Zu diesem Zeitpunkt sind die Überträge jedoch schon vollständig abgearbeitet, so daß ihr Zurücksetzen keine Wirkung hat. Daher kann der Initialisierungsvorgang für die Überträge nie getestet werden und die verwendete Initialisierungshardware ist unnötig.

Bei dem skizzierten Addierer fällt außerdem auf, daß an der niederwertigsten Datenstelle statt des Volladdierers bereits ein Halbaddierer ausreichend gewesen wäre. Natürlich kann der beim Volladdierer konstant auf 0 gelegte Dateneingang nicht auf einen stuck-at-0 Fehler getestet werden. Diese Schaltungsredundanz wurde beim Schaltungsentwurf aus Bequemlichkeit akzeptiert, um kein zusätzliches Gatter entwerfen zu müssen.

Als letztes Beispiel eines nicht-entdeckbaren Fehlers betrachten wir ein einfaches RS-Flipflop. Durch die in Abbildung 6 dargestellte zusätzliche Beschaltung wurde daraus eine Speicherzelle mit Daten- und Enable-Eingang, sowie dem Ausgang q und negiertem Ausgang $\overline{q}$.

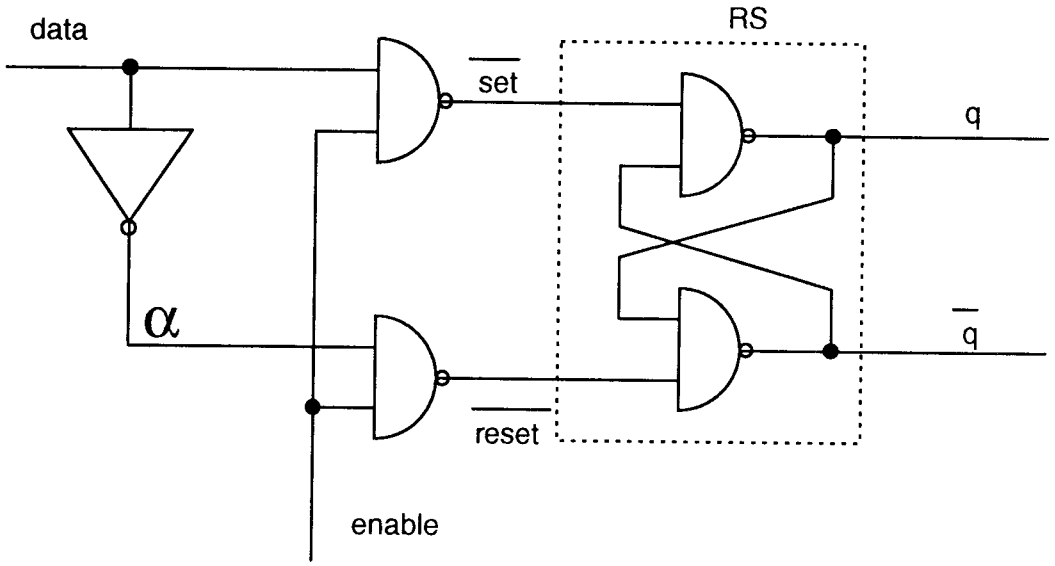


Abbildung 6: RS-FF

Im Fall $enable = 1$ ergibt sich $\overline{set} = \overline{data}$ und $\overline{reset} = data$ so daß am Ausgang $q = data$ und $\overline{q} = \overline{data}$ entsteht.

Setzt man nun $enable = 0$ ergibt sich $\overline{set} = \overline{reset} = 1$ und die Zelle speichert den zuvor eingeschriebenen Wert *data* statisch.

Man sollte denken, daß sich diese Schaltung leicht vollständig testen läßt. Tatsächlich tritt jedoch an der in der Skizze gekennzeichnete Stelle α einen besonderes Phänomen auf. Versucht man nämlich, diese Stelle auf einen stuck-at-1 Fehler zu testen, so muß man $data = 1$ setzen. Im Fehlerfall ergäbe sich dadurch die unzulässige Signalbelegung $\overline{set} = \overline{reset} = 0$ wodurch sowohl q, als auch $\overline{q}$ den Wert 1 annehmen würden.
Man beachte, daß diese Konstellation im fehlerfreien Fall nie auftreten kann. Sofern der Ausgang $\overline{q}$ beobachtet werden kann, wird der Fehler sofort entdeckt. In der vorliegenden Gesamtschaltung wird jedoch nur der nicht-invertierte Ausgang q weiterverwendet, so daß der Fehler nicht direkt beobachtbar ist. Setzt man nun $enable = 0$ um in den Speicherzustand zu schalten, wird $\overline{q}$ negiert auf q zurückgekoppelt und q wird negiert auf $\overline{q}$ zurückgekoppelt.
Lt. Fehlersimulation gerät das Flip-Flop dadurch ins Schwingen, so daß kein definierter Zustand beobachtbar ist. Bei einer separaten Simulation der skizzierten Teilschaltung wird dieses Phänomen als *catastrophic fault* gemeldet. Für die Gesamtschaltung führt dies zu einem angeblich nicht-entdeckten Fehler.

6 Testdurchführung

Wir kommen nun zur praktischen Durchführung des Tests. Bevor die einzelnen verwendeten Testumgebungen vorgestellt werden, soll kurz auf das erzielte Testergebnis eingegangen werden. Wie anfangs bereits erwähnt, wurde ein sehr einfacher von uns konstruierter Funktionstest der 20 ausgelieferten Chips bereits nach der Herstellung beim IMS durchgeführt. Bei Verwendung dieser einfachen Testmuster haben die Chips bei allen Testumgebungen das bereits vom IMS beobachtete Verhalten gezeigt.
Auf Wunsch des IMS sollte dieser erste Test mit möglichst wenigen Testmustern auskommen. Daher wurden nur einige wenige Funktionen des Chips getestet. Insbesondere wurden die Datenregister des Chips nur sehr kurz konfiguriert. Aus diesem Grund ist es wahrscheinlich, daß manche Chips Fertigungsfehler besitzen, die beim IMS noch nicht erkannt werden konnten. Gerade wegen der hohen Wahrscheinlichkeit von Fertigungsfehlern ist das Testen von komplexen Chips ja besonders wichtig.

Tatsächlich stellte sich heraus, daß 6 der 20 Chips nicht einwandfrei funktionieren. Unter den fehlerhaften Chips sind jedoch einige, die eingeschränkt verwendet werden können, da sich der Fehler nur auf eine bestimmte Schieberegisterfalte bezieht, die

dann bei der Konfigurierung des Chips natürlich nicht aktiviert werden darf.
Insgesamt können die Testergebnisse als Erfolg für das durchgeführte Chipentwurfsprojekt gewertet werden. Der Nachweis der Funktionsfähigkeit des Signalprozessors ist gleichzeitig ein Beleg für die Korrektheit der im Projekt entworfenen speziellen Schaltungs- und Layoutgeneratoren, für die manuellen Plazierungs- und Verdrahtungsarbeiten und schließlich für die Realisierbarkeit des Gesamtkonzepts.

Offen ist lediglich noch die Frage, nach der maximal zulässigen Taktrate des Signalprozessors. Bisher wurden bis zu einer Taktrate von 15 MHz korrekte Ergebnisse erzielt. Für die bei höheren Taktraten auftretenden Probleme ist noch nicht geklärt, ob sie vom Signalprozessor oder aber von den jeweiligen Testumgebungen verursacht werden.

7 Testumgebungen

7.1 Logic-Analyzer

Der am einfachsten zu realisierende Test war wurde unter Zuhilfenahme eines Logic-Analyzers (Tektronix DAS 9200) durchgeführt. Dazu mußten lediglich die zuvor berechneten Testmuster geeignet konvertiert werden und es mußte eine Testplatine für den zu testenden Chip erstellt werden. Um mit dem verfügbaren Testmusterspeicher (8 KBit/Pin) auszukommen mußten die Tests in verschiedene nacheinander auszuführende Teiltests (mit jeweils neuer Initialisierung des Chips) zerlegt werden. Der Takt wurde zunächst wie die Testmusterdaten über den Logic-Analyzer an den Chip angelegt. Da der Chip jedoch eine nicht überlappende 2-Phasen Clock benötigt, benötigte jeder Takt 4 aufeinanderfolgende Testeingaben. Der dadurch verbundene Speicheraufwand und die um den Faktor ¼ reduzierte reale Taktrate haben dazu geführt, daß schließlich eine extern generierte Clock verwendet wurde, mit der der Logik-Analyzer synchronisiert wurde.

7.2 HP-Digitaltester

Um den Signalprozessor auf dem an der FH Furtwangen verfügbaren HP 81810S Tester zu untersuchen, wurde eine spezielle Adapterplatine entworfen und verdrahtet [6]. Das größte Problem war dabei, daß der Tester über nicht genug Kanäle verfügt, um sämtliche relevanten Datenausgaben des Chips gleichzeitig zu beobachten und alle Eingänge zu setzen. Deshalb mußten selten benötigte Konfigurationseingänge mit Hilfe von DIP-Schaltern angesteuert werden und die Ausgabepins wurden gruppenweise über Multiplexer zusammengefaßt.

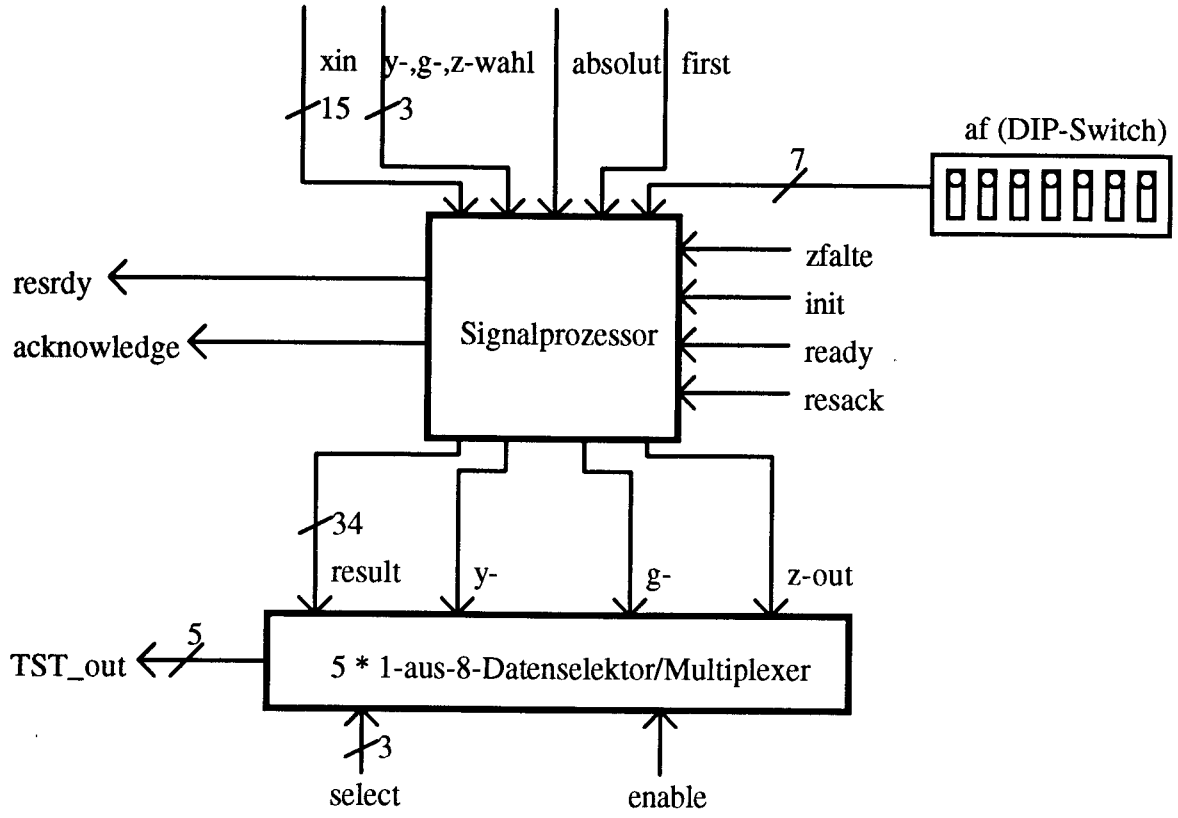


Abbildung 7: PIN-Multiplexing

Abbildung 7 zeigt das Blockschaltbild der PIN-Auswahl beim hp-Tester. Von den 34-Result-Ausgängen und den 3 Schieberegisterabgriffen (g,y, und z) werden jeweils Gruppen a 5 Pins an den Tester weitergereicht (TST_out). Mit 8 Testdurchläufen können somit alle Ausgänge beobachtet werden.

7.3 PC-Umgebung

Als dritte Testmethode wurde der Signalprozessor als Co-Prozessor eines PCs betrieben. Dazu wurde eine spezielle Karte mit einem lokalen Taktgenerator und Ein-Ausgabekanälen realisiert. Auf dem PC wurde eine spezielle Software implementiert, über die sich der Signalprozessor konfigurieren und mit Daten versorgen läßt. Zur Vereinfachung der Versuche wurde in dieser Software die gleiche Schnittstelle wie bei der Schaltungssimulation des Signalprozessors realisiert. Damit können die bereits zur Simulation des Prozessors verwendeten Programme auch zum direkten Test des Prozessors eingesetzt werden.

Gegenüber den bisher dargestellten Testumgebungen gibt es bei der PC-Umgebung einen wesentlichen Unterschied. Erstmals wurde der Signalprozessors asynchron zur Testumgebung betrieben. Statt sämtliche Ausgabesignale einschließlich der Testausgaben in

jedem Takt mit den simulierten Werten zu vergleichen, werden bei der PC-Umgebung lediglich die vom Prozessor gelieferten Endergebnisse mit Hilfe von Handshake-Signalen übernommen und ausgewertet. Hier wird also kein exaktes Timing überprüft, sondern es wird lediglich getestet, ob die gelieferten Endergebnisse stimmen.

Bei dieser Methode wurden bis zu einer Taktrate von 15MHz korrekte Ergebnisse erzielt. Bei höheren Taktraten wird es bei der verwendeten Schnittstellenkarte bereits schwierig, die Ergebnisdaten schnell genug vom PC aus zu lesen. Deshalb ist auch bei diesen Versuchen unklar, wo die Leistungsgrenzen des Signalprozessors liegen.

8 Leistungsaufnahme der Chips

Bei der Durchführung der Tests wurde auch die Leistungsaufnahme der Chips in Abhängigkeit von der Taktrate ermittelt. In Abbildung 8 sind die Ergebnisse für einen der funktionierenden Chips dargestellt.

Chip Nr.	25kHz	0.25MHZ	1MHz	2.5MHz	5MHz
1	0.93mA	2.3mA	6.5mA	16.3mA	32.7mA

Abbildung 8: Stromaufnahme in Abhängigkeit der Frequenz

Bei einem der defekten Chips konnte eine erhöhte Stromaufnahme von etwa 3mA über den gesamten Frequenzbereich beobachtet werden. Die anderen als fehlerhaft nachgewiesenen Chips zeigten bei dieser Messung jedoch keine signifikanten Abweichungen.

9 Zusammenfassung und Ausblick

Für einen an der FH Furtwangen entwickelten und beim IMS gefertigten Signalprozessor wurden Testmuster bezgl. des Stuck-at-Fehlermodells hergeleitet. Abgesehen von einigen grundsätzlich wegen Schaltungsredundanzen nicht beobachtbaren Fehlermöglichkeiten konnte eine 100%-ige Fehlerüberdeckungsrate erreicht werden. Mit den Testmustern wurden 6 der 20 ausgelieferten Chips als defekt erkannt.

Die Testergebnisse können als Erfolg des durchgeführten Chipentwurfsprojekts gewertet werden. Eine genaue Untersuchung der maximal verwendbaren Taktrate steht jedoch noch aus.

Literatur

[1] W. Rülling: „Ein VLSI-Chip zur schnellen Berechnung von Faltungsintegralen", MPC-FH Workshop in Göppingen, 1991
[2] P. Leber: „Layoutentwurf für ein Schieberegister variabler Länge in der Sea-of-Gates Technologie des IMS", MPC-FH Workshop in Aalen, Januar 1991
[3] M. Faas, P. Leber, W. Rülling: „Realisierung großer Netze", MPC-FH Workshop in Ravensburg-Weingarten, 1993
[4] W. Rülling: „Automatische Meßwertkorrektur für intelligente Sensoren", MPC-FH Workshop in Ulm, 1994
[5] M. Schwarz: „Testmustergenerierung für Schaltungen in dynamischer Logik", Diplomarbeit , FH Furtwangen, 1993
[6] R. Niederhüfner: „Test eines Signalprozessors", Diplomarbeit, FH Furtwangen, 1994

Selbsttest eines Solarladereglers

Prof. Führer (FH Ulm)
Stöckle Ralf (FH Ulm)

Einleitung

Bei der Integration eines bereits bestehenden, diskreten Solarladeregles sind weitere Funktionen mit aufgenommen worden. Eine dieser zusätzlichen Funktionen ist der „Selbsttest". Um die Aufgaben des Selbsttestes besser zu verstehen, ist es nötig, auf das Umfeld des Solarladereglers genauer einzugehen.

Ein Laderegeler ist immer dort nötig, wo ein Speicher im Solarsystem benötigt wird. Der momentan billigste und gebräuchlichst Speicher ist der Bleiakkumulator.

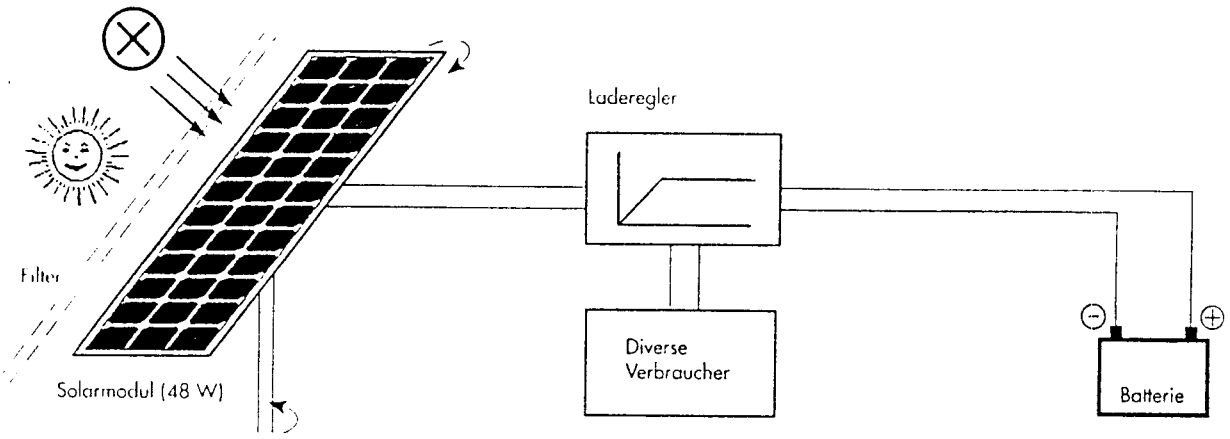


Bild 1 12V oder 24V Solarsystem mit Speicher

Ein Bleiakkumulator erfordert spezielle Funktionen des Ladereglers:

- Laden nach IU-Kennlinie mit Pulsweitenmodulation (PWM)
- Überladeschutz
- Gasungsregelung
- Tiefentladeschutz

Laden nach IU-Kennlinie

Das Ziel ist es, den Akku möglichst schnell zu laden und damit die zu Verfügung stehende Sonnenenergie möglichst effektiv auszunutzen. Ein Problem dabei ergibt sich aus dem Umstand, daß

der Akku noch nicht vollständig geladen ist, wenn die Akkuklemmenspannung die Ladeendspannung erreicht hat. Der Grund dafür ist die hohe Säurekonzentration an den Platten des Akkus beim Ladevorgang, was sich in einer hohen Klemmenspannung wiederspiegelt. Unterschiedliche Säurekonzentrationen gleichen sich durch Diffusion aus. Aber dieser Vorgang läuft nur mit einer endlichen Geschwindigkeit ab. Deshalb muß der Laderegler den Akku auf der Ladeendspannung (wegen Gasung) konstant halten, aber trotzdem die dazu notwendigen Ladeströme weiter zulassen. Dies wird auch als IU-Laden (Bild 2) bezeichnet. Realisiert wird dieser Ladevorgang mithilfe der Pulsweitenmodulation (PWM), wobei das Tastverhältnis der Schalter, des Serienstellgliedes(Bild 3)oder des Parallelstellgliedes (Bild 4), den Ladestrom einstellt.

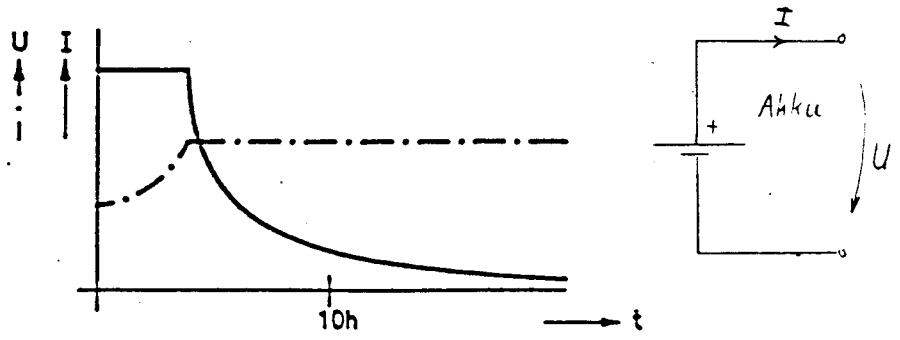


Bild 2 Verlauf der Akkuklemmenspannung und des Ladestromes bei IU-Laden

Überladeschutz

Erreicht ein Bleiakkumulator beim Laden die Ladeendspannung, so führt ein Teil des Ladestromes zur sogenannten "Gasung". Dabei wird das im Elektrolyten enthaltene Wasser in Wasserstoff und Sauerstoff gespaltet. Neben diesem Wasserverbrauch, der natürlich ausgeglichen werden muß, werden bei starker Gasung auch Partikel der aktiven Masse aus den Platten des Akkus ausgeschwemmt. Dies hat einen Kapazitätsverlust oder sogar die Zerstörung des Akkus zur Folge. Es werden zwei Schaltungsvarianten zum Überladeschutz eingesetzt:

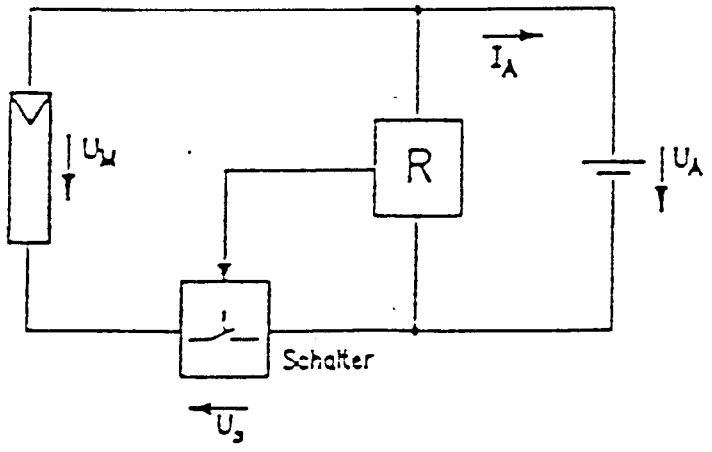


Bild 3 Serienstellglied

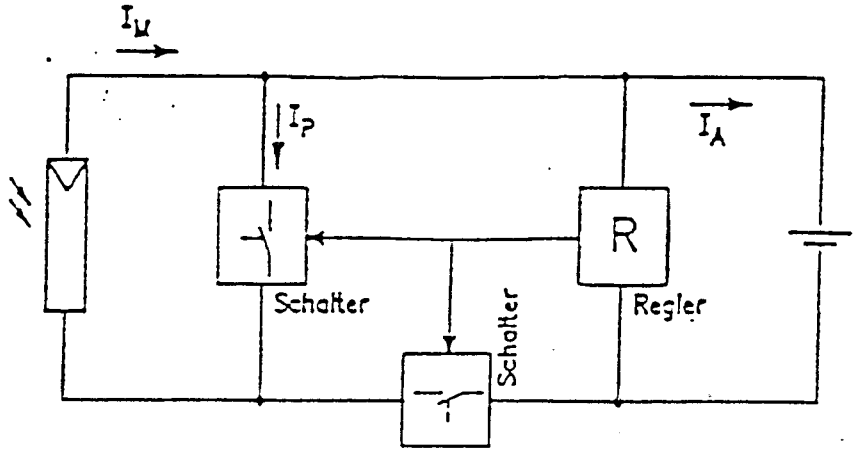


Bild 4 Parallelstellgied

Beim Serienstellglied wird der Ladevorgang dadurch unterbrochen, daß der Schalter der in Reihe zum Solargenerator liegt geöffnet wird. Dagegen wird beim Parallelstellglied der Ladestrom über den parallelen Schalter kurzgeschlossen. Dies kann man bedenkenlos tun, da sich der Solargenerator prinzipiell wie eine Stromquelle verhält. Der zweite Schalter ist nun dazu da, damit beim Kurzschließen des Solargerators nicht auch noch der Akku kurzgeschlossen wird. Die Folge wäre die Zerstörung der beiden Schalter und des Akkus.

Gasungsregelung

Eine gelegentliche, geregelte Gasung bringt hingegen Vorteile mit sich. Durch die aufsteigenden Gasblasen wird der Elektrolyt umgewälzt (bei großen Akkubanken wird dies mit einem speziellen Pumpsystem ausgeführt). Dadurch wird die Säureschichtung, die vor allem beim Laden auftritt, verhindert. Wichtigster Schaden der durch die Säureschichtung eintreten kann, ist wiederum ein Kapazitätsverlust des Akkus. Die Gasungsregelung wird aktiviert, wenn die Akkuklemmenspannung unter einen bestimmten Wert absinkt. Dabei wird die Ladeendspannung durch die höhere Gasungsendspannung ausgetauscht (beide Spannungen sind sehr stark temperaturabhängig: -4mV/K/Zelle; im 12V-System sind es 6 Zellen ==> -24mV/K).

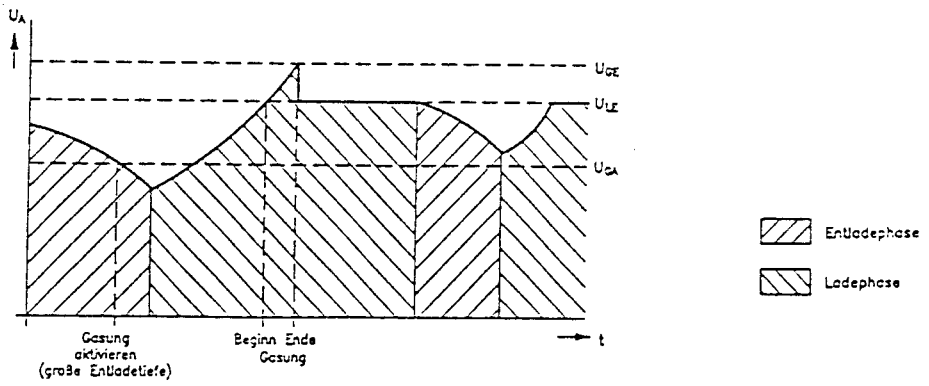


Bild 5 Aktiveren der Gasungsregelung

Tiefentladeschutz

Bei tiefer Entladung des Bleiakkumulators kommt es zur sogenannten Sulfatierung. Dabei Entstehen kleine Kristalle an den Plattenoberfläche, die sich beim Laden nicht mehr vollständig auflösen. Das Kristallwachstum ist um so größer, je größer die Entladetiefe ist. Deshalb wird der Verbraucher bei unterschreiten der sogenannten Entladeendspannung abgeschaltet.

Die Bestimmung des Ladezustandes ist sehr schwierig. Am Genauesten ist die Ermittlung dann, wenn man die Akkumulatorspannung und den Entladestrom als Berechnungsgrößen zur Abschaltung des Verbrauchers heranzieht. Damit erhält man zwei Varianten des Tiefentladeschutzes. Variante eins ist die „feste" Entladeendspannung und Variante zwei ist die „variable" Entladeendspannung.

Den Vorteil der variablen gegenüber der festen Entladeendspannung erkennt man im nachfolgenden Diagramm, das die nichtlinearen Kennlinien der Akkumulatorspannung in Abhängigkeit vom Entladestrom zeigt. Der Parameter der Kennlinienschar ist dabei die Entladetiefe. Man stellt fest, daß bei kleinen Entladeströmen der Akku, mit fester Entladeendspannung, sehr tief entladen werden kann. Besser dagegen ist eine laststromabhängig Entladeendspannung, die sogenannte "variable" Entladeendspannung.

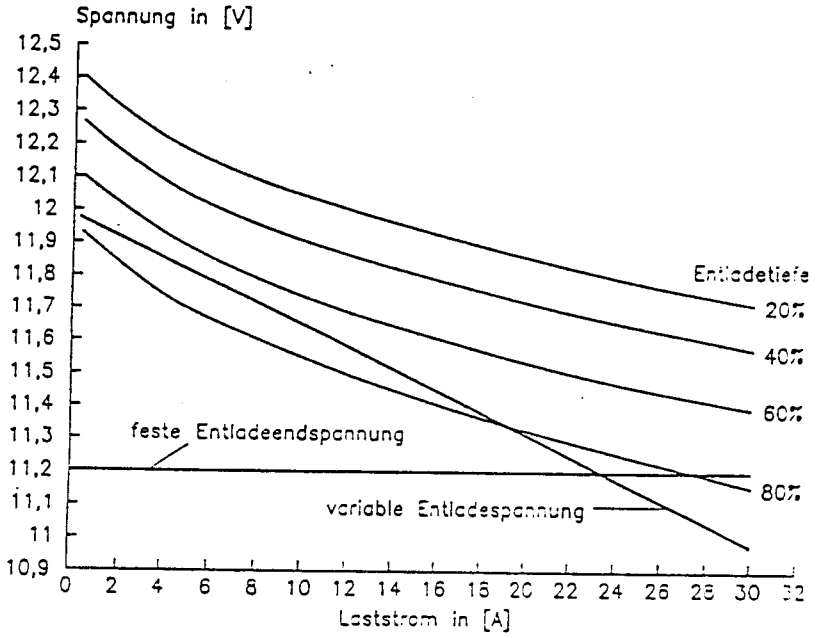


Bild 6 Feste und variable Entladeendspannung

Zusammenfassend kann man darauf hinweisen, daß ein Bleiakkumulator nur dann eine lange Lebensdauer erreicht, wenn er vor Überladung und Tiefentladung geschützt wird. Diese Funktionen sind die minimale Anforderung an einen Laderegler.

Selbsttest

Ziel des Selbsttestes ist es, die Aufgaben des Prüffeldes zu unterstützen. Die mitintegrierte Funktion „Selbsttest" stellt nun fest, ob der Regler die zuvor aufgeführten Funktionen ausführen kann. Dabei werden externe und interne Bereiche überprüft.

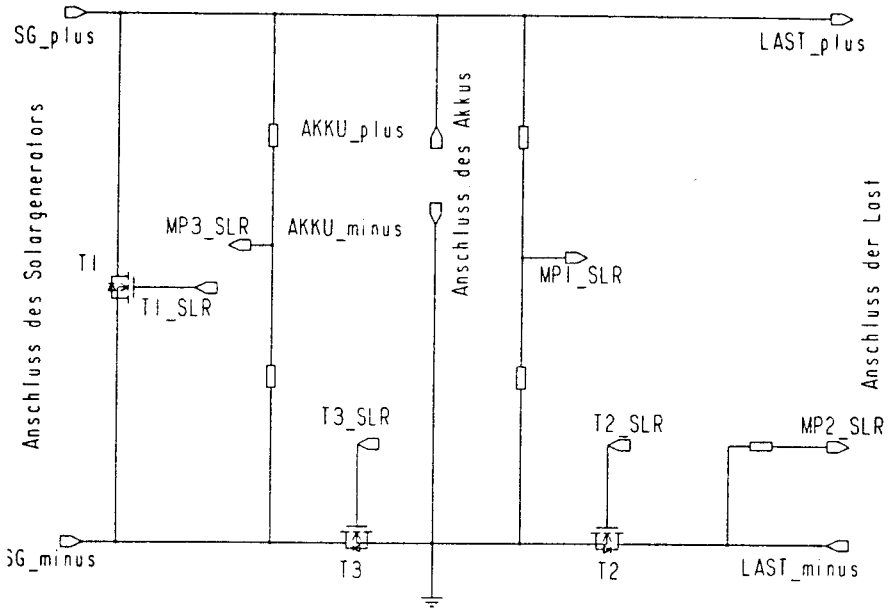


Bild 7 Externe Beschaltung des IC (Stellglieder)

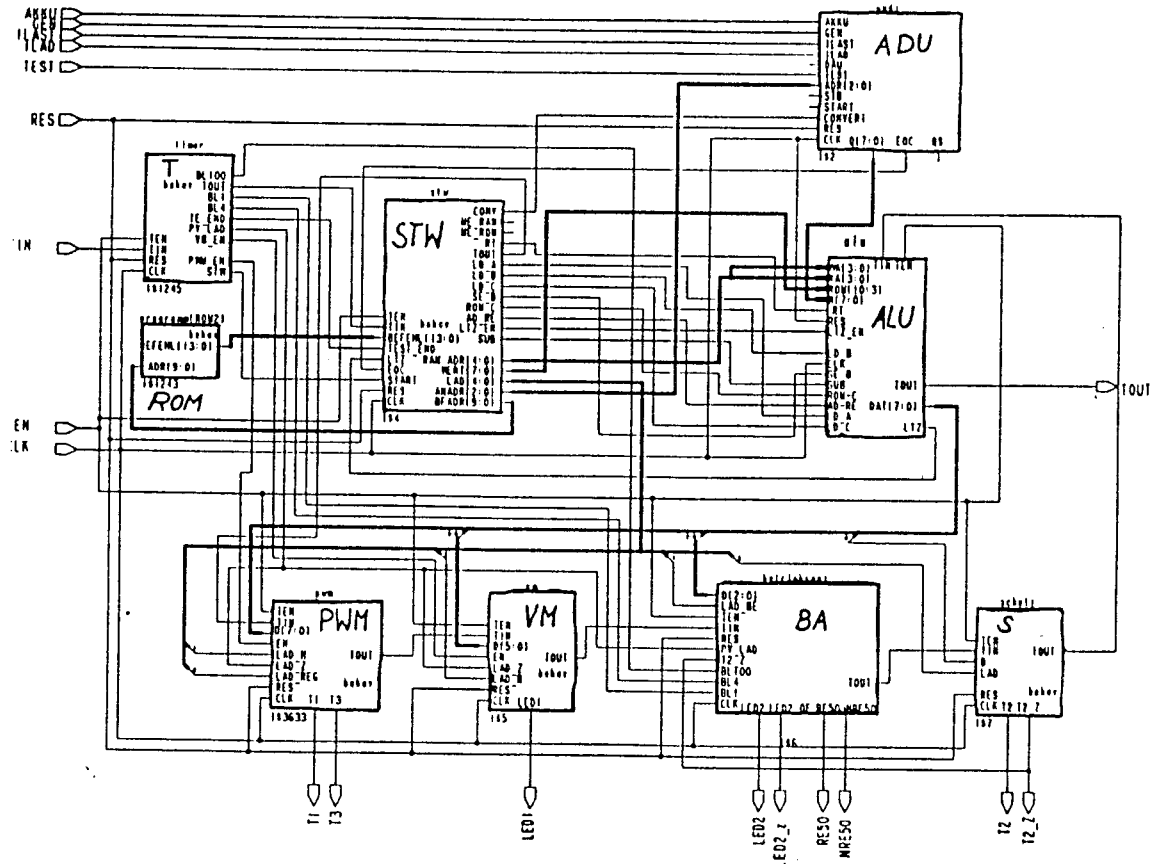


Bild 8 Interne Blockstruktur des Ladereglers

Intern (Blöcke)

In Bild 8 sieht man das Blockschaltbild des integrierten Solarladereglers. Im Gegensatz zum bisherigen diskreten und analog arbeitenden Laderegler ist dies ein integrierter und digital arbeitender Laderegler. Die Ausführung der Funktionen ist in einem Programm im ROM hinterlegt.

Um alle Funktionen ausführen zu können, sind 16 Befehle notwendig. Ein eigens dafür geschriebener Compiler übersetzt das Programm. Diese Übersetzung ist die Grundlage für die Daten im ROM.

Das Programm im ROM wird mit dem BIST-Test (Buid In Self Test) überprüft. Dazu stellt der verwendete ES2 - Prozeß einen speziellen ROM-Generator zur Verfügung, der alle notwendigen Gatter zur Realisierung des BIST automatisch generiert. Das Ergebnis des Testdurchlaufes ist eine Ein-Bit-Information darüber, ob die Daten, die bei der Herstellung in das ROM geschrieben wurden, immer noch richtig sind.

Beim Ablauf des Selbsttestes (besser Selbsttest-Unterprogramm), angestoßen durch einen Power-On-Reset, wird zuerst die ALU überprüft. Dazu wird eine Befehlssequenz mit einem Startwert durchlaufen und am Ende das Ergebnis auf Richtigkeit abgeprüft. Natürlich werden die Gatter des IC bei der Herstellung getestet, aber der Umgang mit CMOS-Bausteinen ist nicht unproblematisch. Deshalb wird bei der Endkontrolle im Prüffeld der Selbsttest gebraucht.

Wenn die ALU in Ordnung ist (auch das Steuerwerk, das die Befehle ausdekodiert hat, und das ROM), dann wird der ADU auf plausibilität getestet. Dazu stellt man einen definierten Betriebszustand am komplett zusammengebauten Regler ein. Zum Beispiel eine Akkuspannung von 12V mit 0.1% Toleranz. Der IST- und der SOLL-ADU-Wert werden miteinander programmtechnisch verglichen. Zu große Abweichungen führen zu einer Fehlermeldung, die optisch mit LED angezeigt wird. Bei dieser Überprüfung werden gleich die Analogeingänge und die vorgeschalteten Spannungsteiler mit kontrolliert.

Wenn der interne Teil in Ordnung ist, dann wird mit dem externen fortgefahren.

Extern (außerhalb des IC)

Die Stellglieder (MOS-Transistoren T1, T2 und T3 in Bild 7) werden dadurch überprüft, daß die Potentiale an den Meßpunkten (MP1, MP2, MP3 in Bild 7) im eingeschalteten und ausgeschalteten Zustand miteinander verglichen werden. Abweichungen führen zu Fehlermeldungen, die auf die Fehlerursache hindeuten. Gleichzeitig werden auch die Blöcke PWM, VM (Voltmeter), BA (Betriebsanzeige) und SCHUTZ getestet.

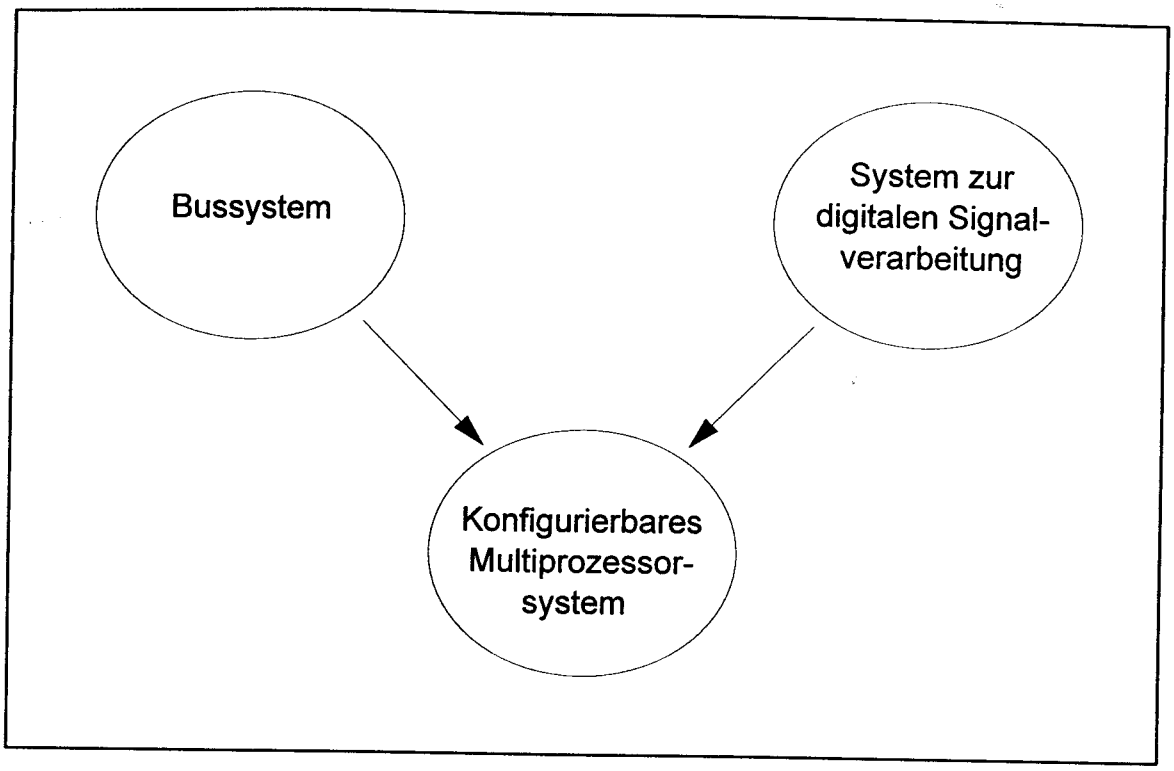
Wenn der Selbsttest durchlaufen ist, kann man mit Sicherheit aussagen, ob der komplette Regler richtig arbeitet. Somit ist durch zusätzliche Resourcen, die eine Integration mit sich gebracht hat, eine Möglichkeit geschaffen worden, den Endtest im Prüffeld mit aufzunehmen. Die Einsparungen an Prüfperepherie rechtfertigen diese Maßnahme entscheidend.

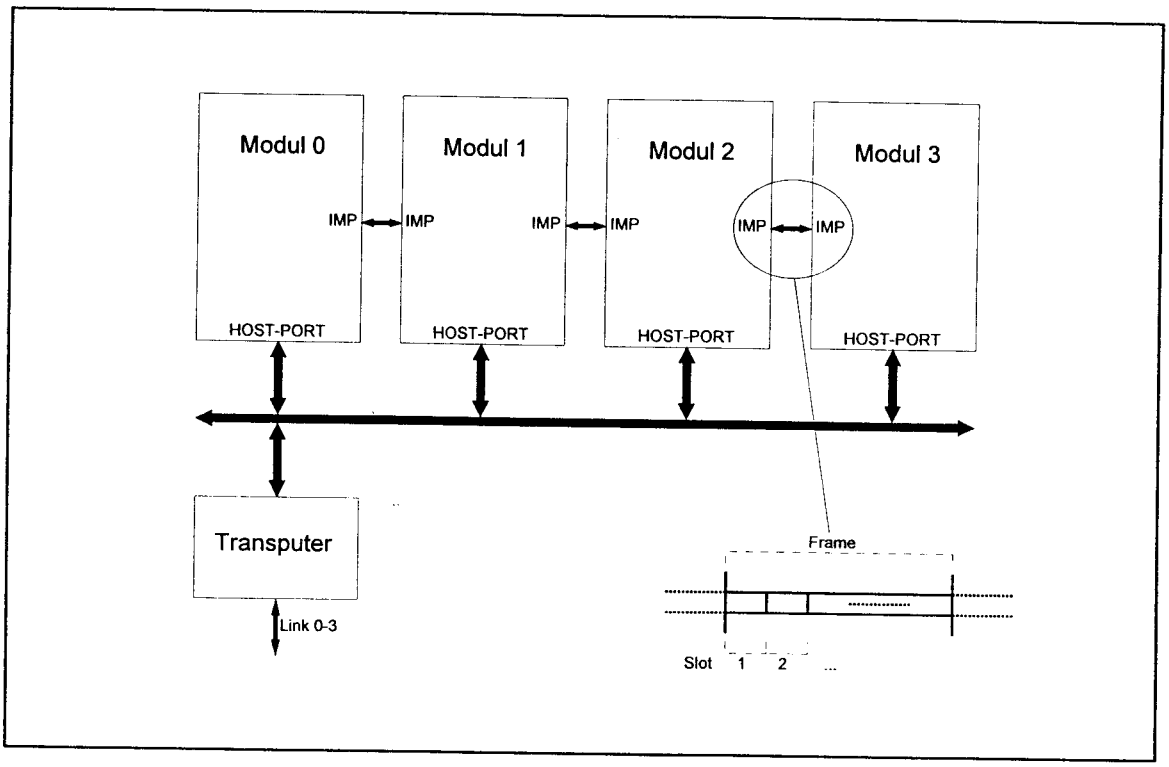
MPC-Workshop SS94

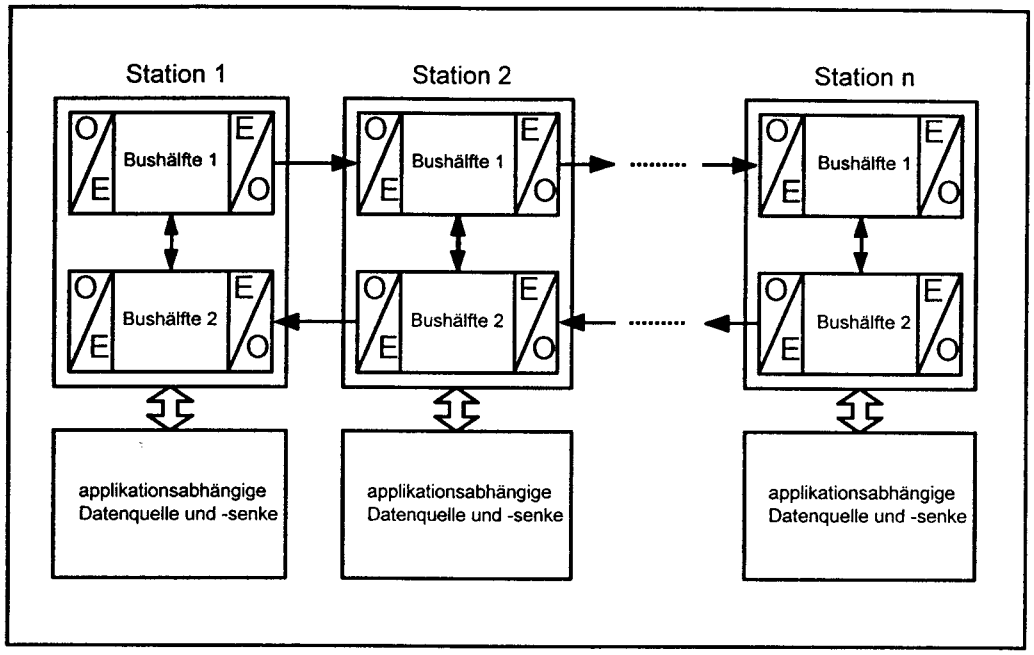
M. Gaiser, N. Höptner
Fachhochschule Pforzheim

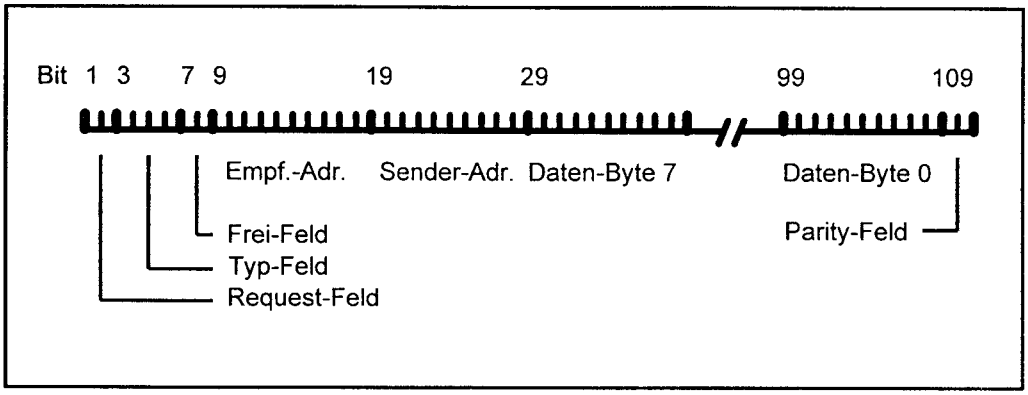
VHDL-gestütztes LCA-Design eines Schnittstellenwandlers

- ☞ Einführung
- ☞ Entwurf
- ☞ Entwicklungsumgebung
- ☞ Bewertung und Ausblick


Bussystem
System zur digitalen Signal-verarbeitung
Konfigurierbares Multiprozessor-system

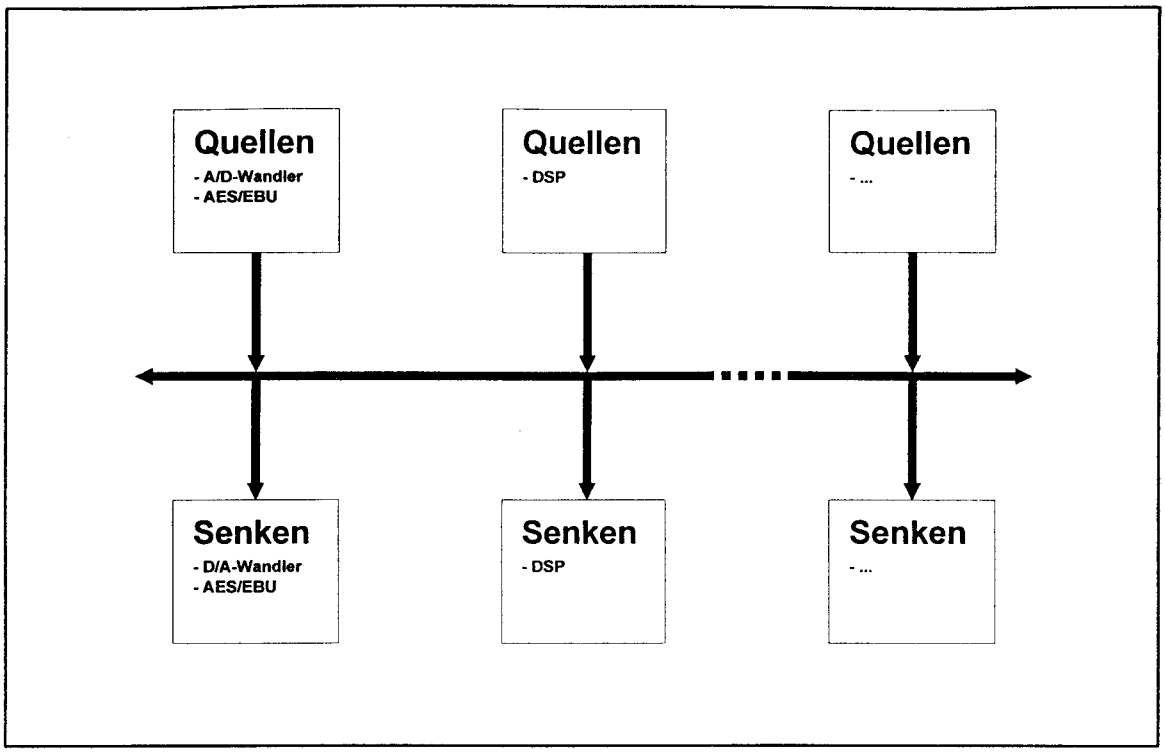

Modul 0
Modul 1
Modul 2
Modul 3
IMP
IMP
IMP
IMP
IMP
IMP
HOST-PORT
HOST-PORT
HOST-PORT
HOST-PORT
Transputer
Link 0-3
Frame
Slot
1
2
...

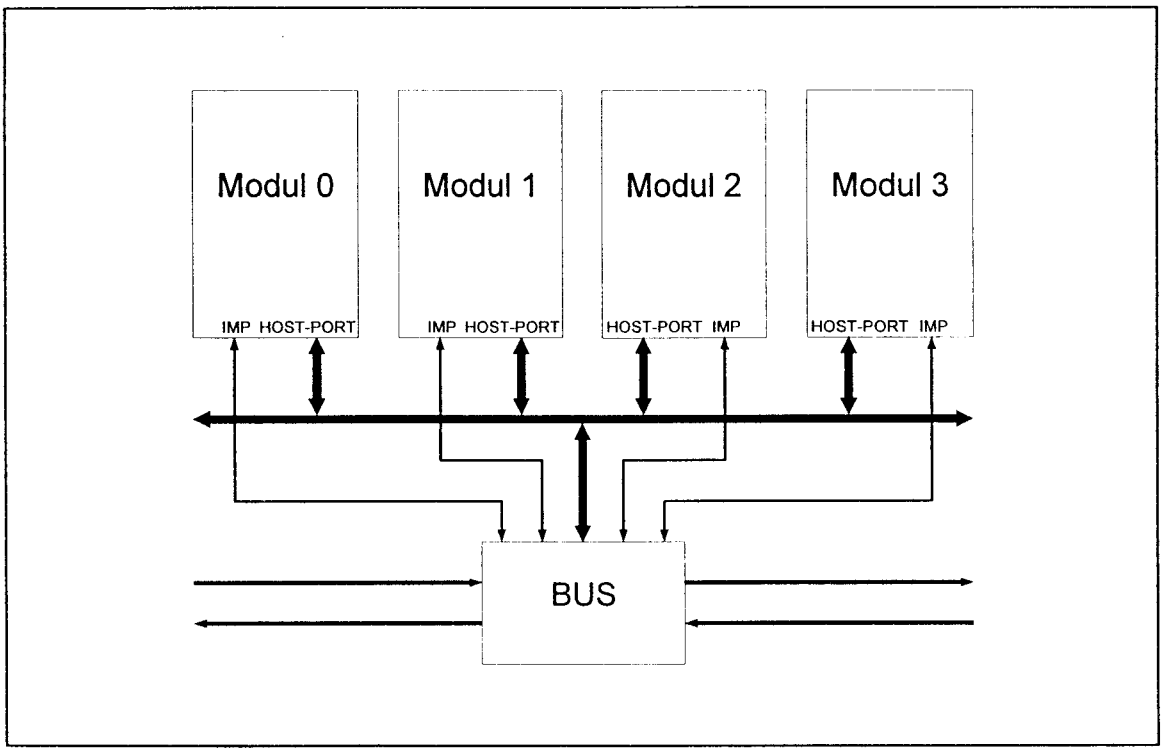

Station 1
Station 2
Station n
O/E
E/O
Bushälfte 1
Bushälfte 2
applikationsabhängige Datenquelle und -senke

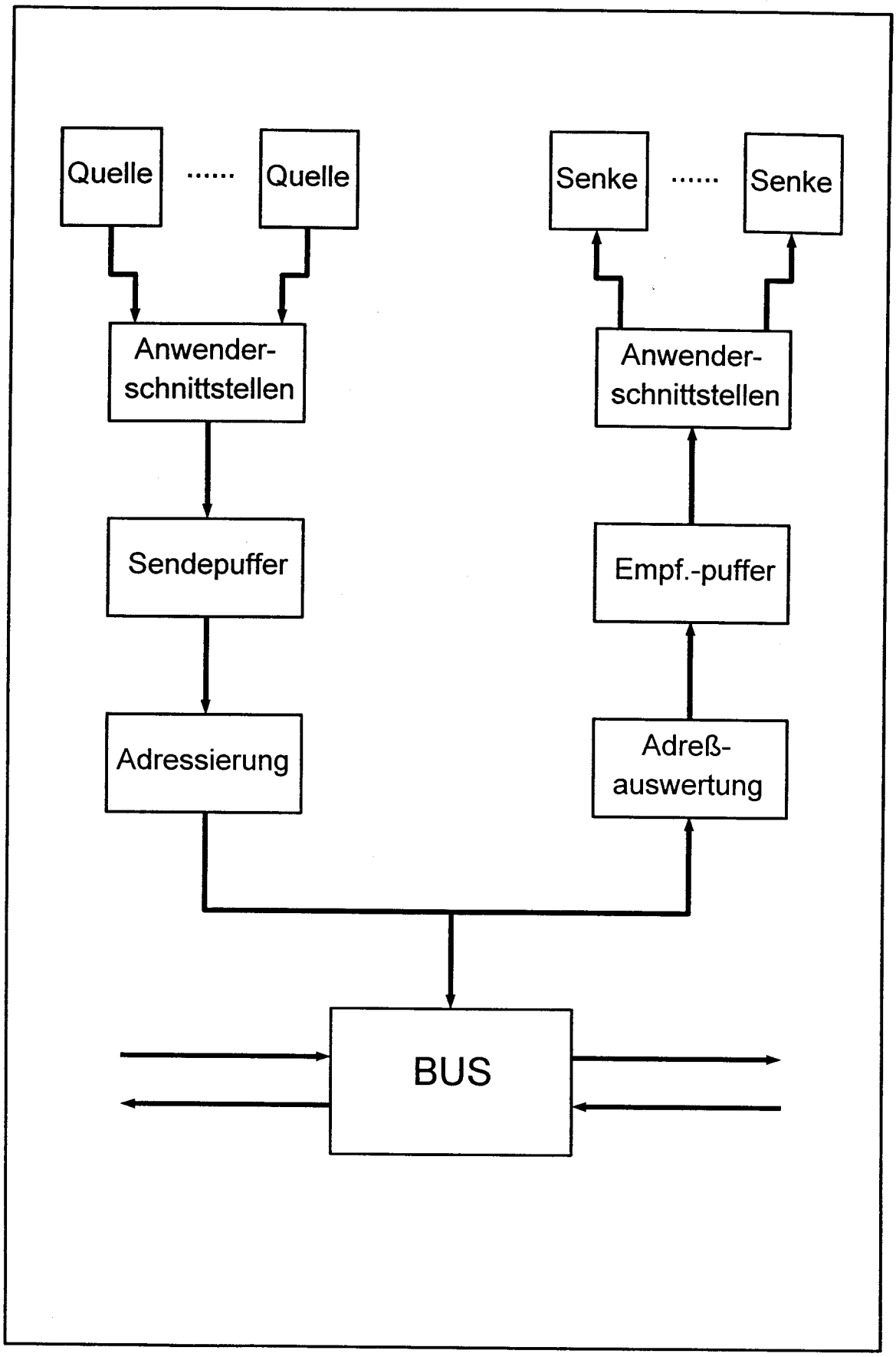

Bit 1 3 7 9 19 29 99 109
Empf.-Adr.
Sender-Adr.
Daten-Byte 7
Daten-Byte 0
Frei-Feld
Typ-Feld
Request-Feld
Parity-Feld

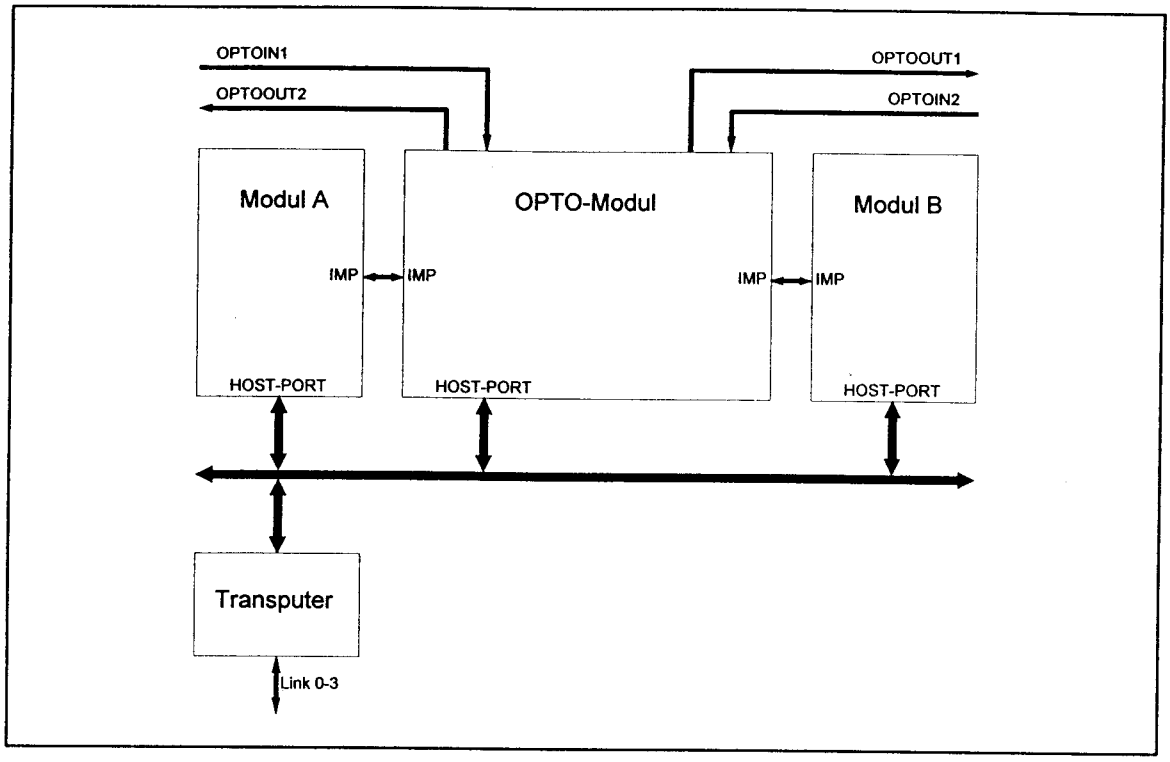
Anforderungen an das Bussystem

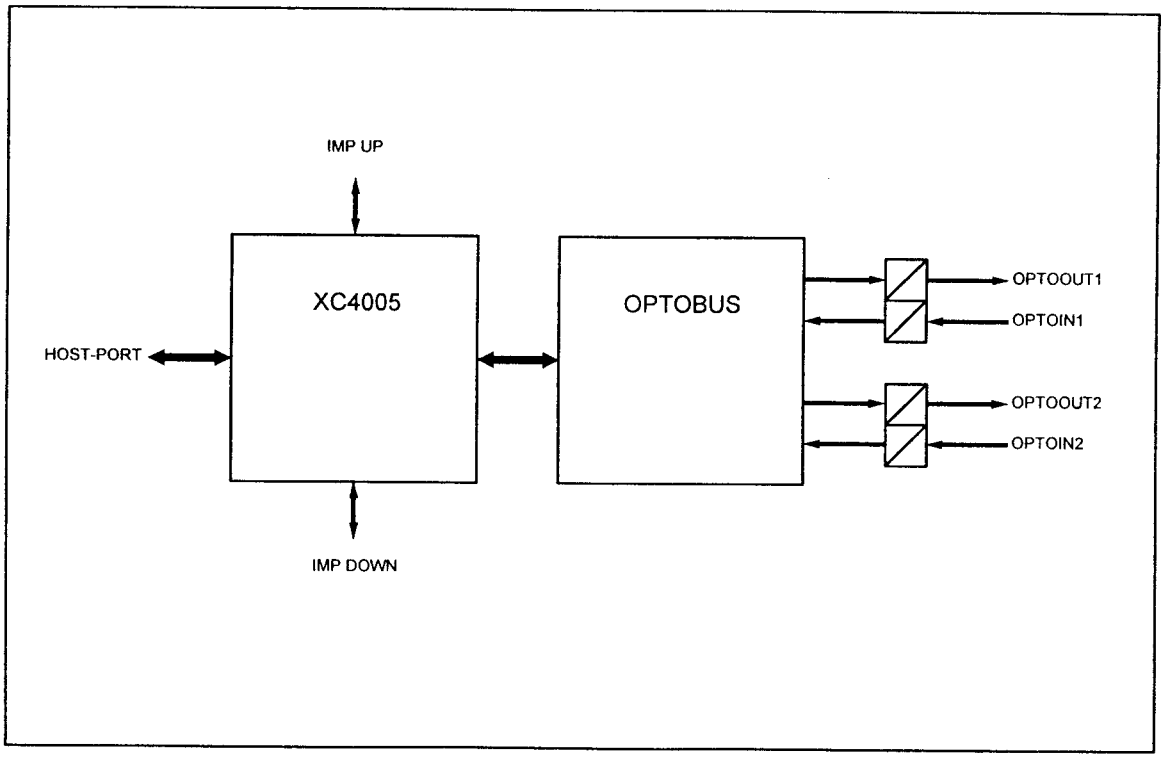
- Hohe Übertragungsrate
- Echtzeitfähigkeit
- Bus als Kreuzschiene
- Konvertierung unterschiedlicher Datenformate

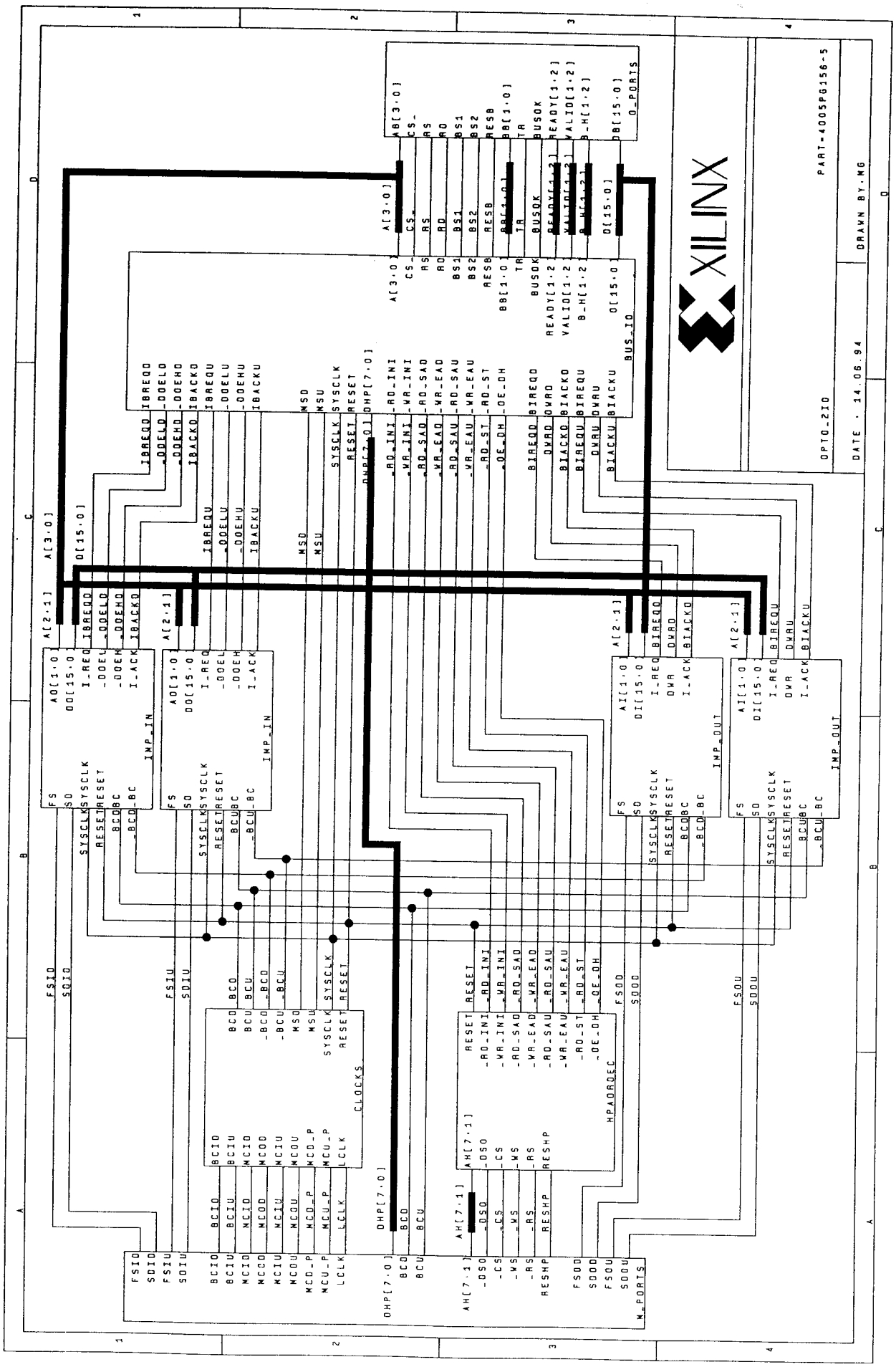

Quellen
- A/D-Wandler
- AES/EBU
Quellen
- DSP
Quellen
- ...
Senken
- D/A-Wandler
- AES/EBU
Senken
- DSP
Senken
- ...

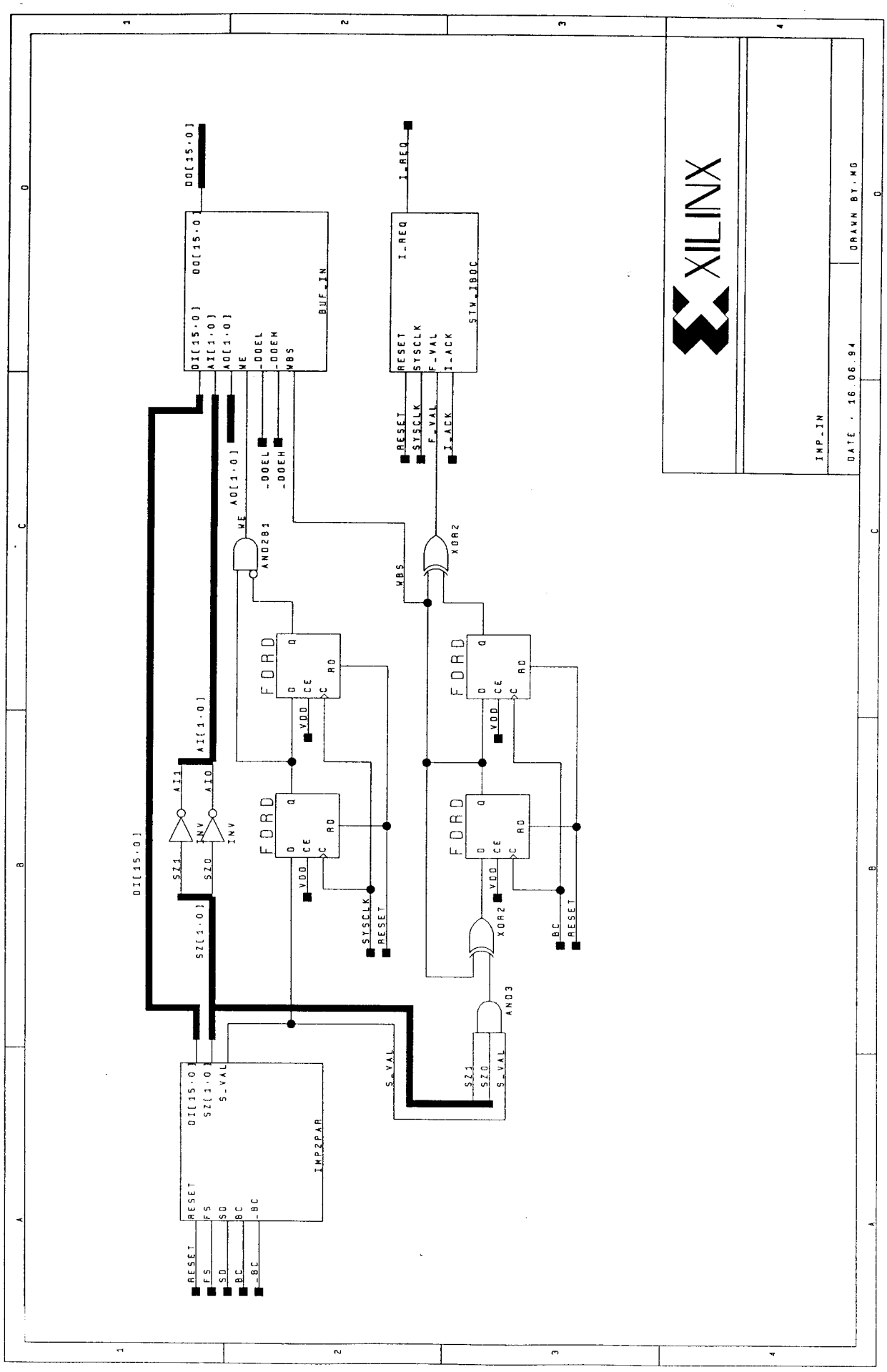

Modul 0
IMP HOST-PORT
Modul 1
IMP HOST-PORT
Modul 2
HOST-PORT IMP
Modul 3
HOST-PORT IMP
BUS

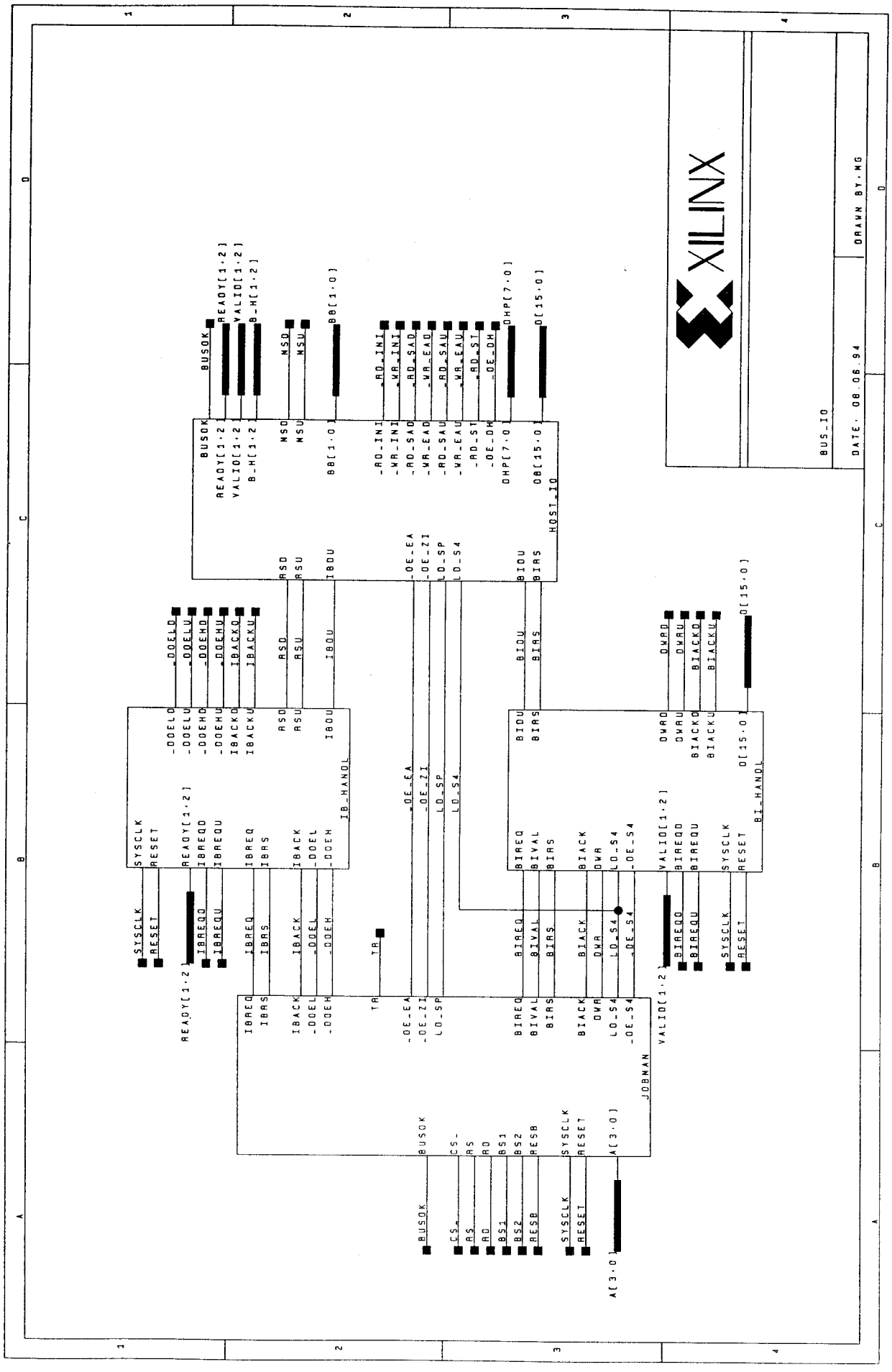

Quelle
.....
Quelle
Senke
.....
Senke
Anwender-
schnittstellen
Anwender-
schnittstellen
Sendepuffer
Empf.-puffer
Adressierung
Adreß-
auswertung
BUS


OPTOIN1
OPTOOUT2
OPTOOUT1
OPTOIN2
Modul A
OPTO-Modul
Modul B
IMP
IMP
IMP
IMP
HOST-PORT
HOST-PORT
HOST-PORT
Transputer
Link 0-3


IMP UP
XC4005
OPTOBUS
HOST-PORT
OPTOOUT1
OPTOIN1
OPTOOUT2
OPTOIN2
IMP DOWN

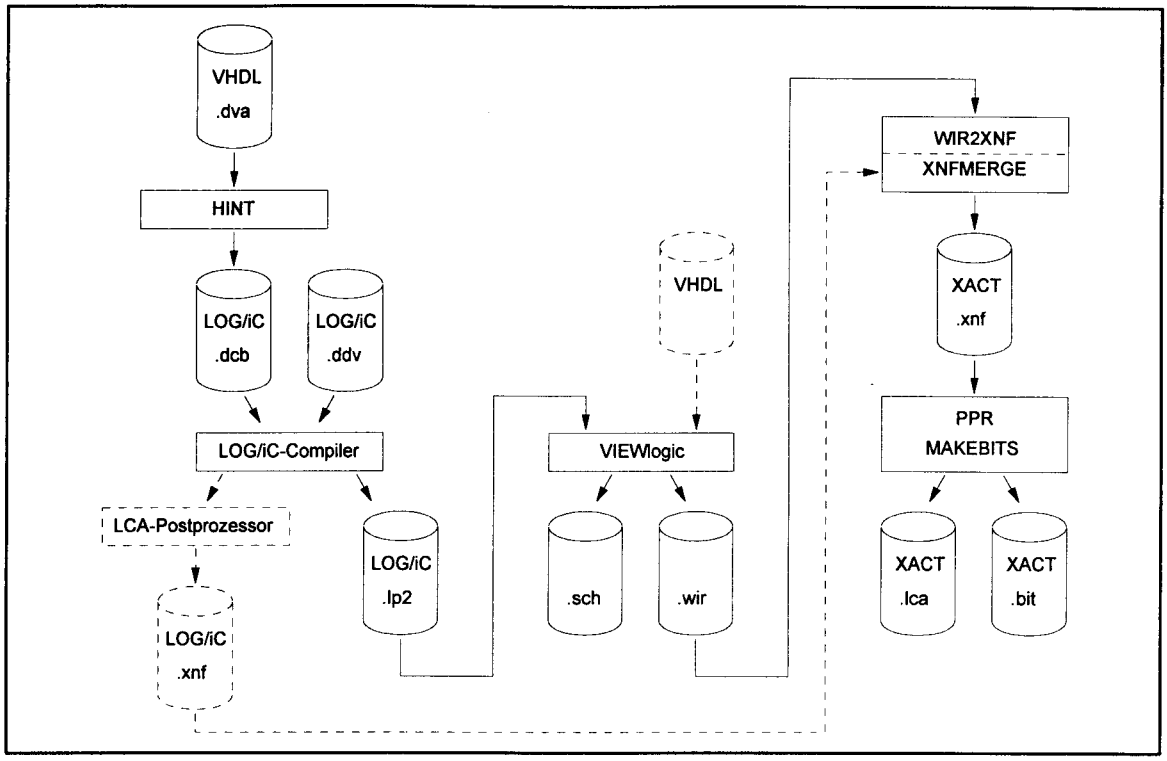

XILINX
PART=4005PG156-5
DRAWN BY:MG
DATE : 14.06.94
OPTO_ZIO
BUS_IO
M_PORTS
CLOCKS
IMP_IN
IMP_OUT
MPADDEC


XILINX
IMP_IN
DATE : 16.06.94
DRAWN BY:MG
DO[15:0]
I_REQ
BUF_IN
STW_IBOC
DI[15:0]
AI[1:0]
AO[1:0]
WE
_DOEL
_DOEH
WBS
RESET
SYSCLK
F_VAL
I_ACK
AND2B1
XOR2
FDRD
VDD
INV
SZ1
SZ0
SZ[1:0]
S_VAL
AND3
BC
IMPZPAR
FS
SD


XILINX
BUS_IO
DATE: 08.06.94
DRAWN BY: MG
HOST_IO
IB_HANDL
BI_HANDL
JOBMAN
BUSOK
READY[1:2]
VALID[1:2]
B_H[1:2]
MSO
MSU
BB[1:0]
AD_INI
WR_INI
AD_SAO
WR_EAO
AD_SAU
WR_EAU
AD_ST
OE_DH
DHP[7:0]
D[15:0]
ASO
ASU
IBOU
BIDU
BIAS
OE_EA
OE_ZI
LO_SP
LO_S4
SYSCLK
RESET
IBREQ
IBACK
IBVAL
IBRAS
IBREQO
IBREQU
IBACKO
IBACKU
DWRO
DWRU
BIREQ
BIVAL
BIRAS
BIACK
DWR
OE_S4
TA
CS_
AS
AD
BS1
BS2
RESB
A[3:0]

Entwicklungsumgebung

- Hardware
  - 486 DX2, 66MHz
- Software
  - LOG/iC
  - HINT
    (VHDL-Präprozessor für LOG/iC)
  - Viewlogic
  - XACT
    (LCA-Designtools)


VHDL
.dva
HINT
LOG/iC
.dcb
LOG/iC
.ddv
LOG/iC-Compiler
LCA-Postprozessor
LOG/iC
.xnf
LOG/iC
.lp2
VHDL
VIEWlogic
.sch
.wir
WIR2XNF
XNFMERGE
XACT
.xnf
PPR
MAKEBITS
XACT
.lca
XACT
.bit

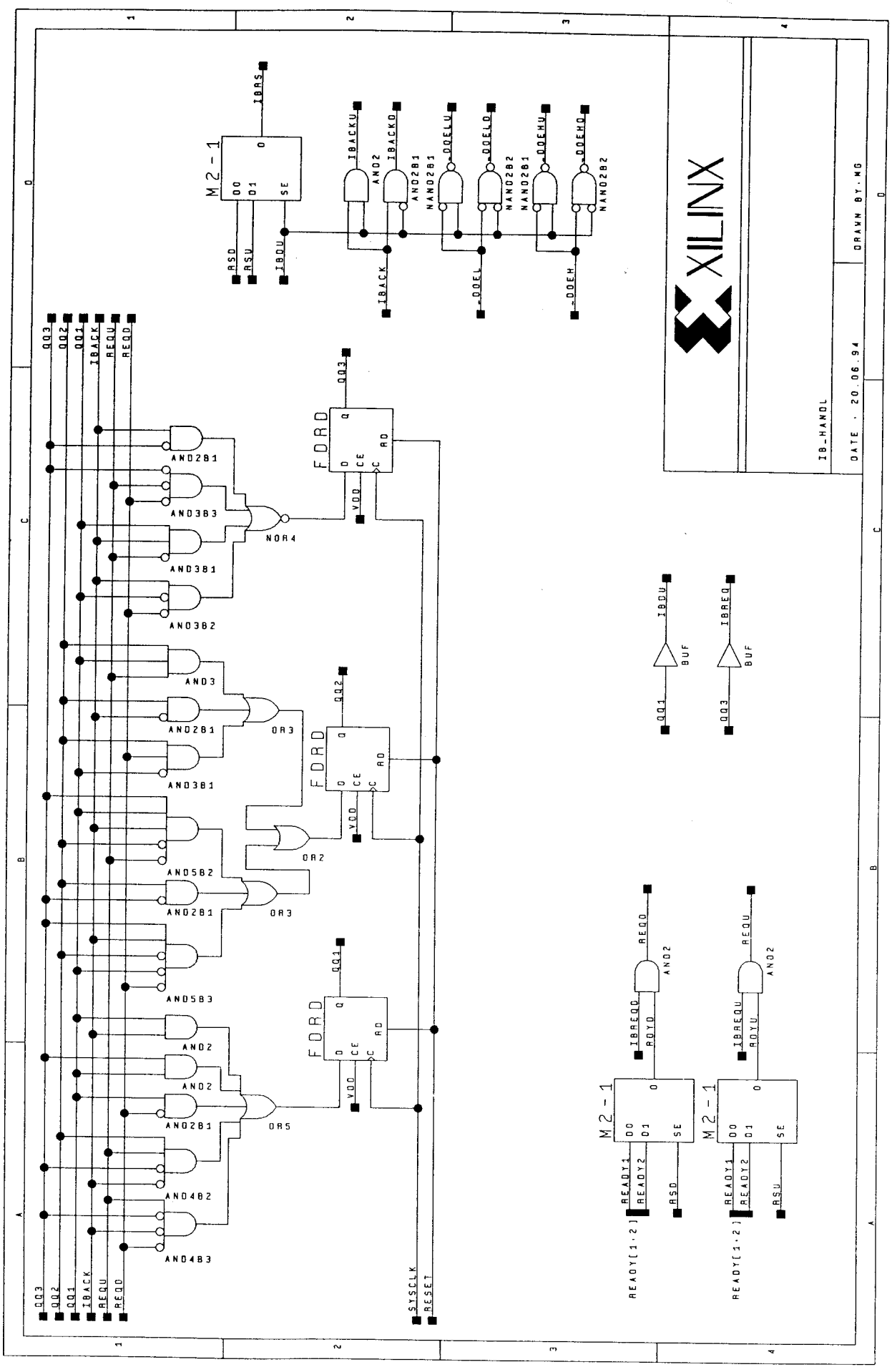
Einige Einschränkungen von HINT

- Keine gemischte Modellierung von Struktur und Verhalten innerhalb einer *architecture*
- Keine Libraries
- Modellierung des zeitlichen Verhaltens wird nicht unterstützt
- Maximal zwei Hierarchieebenen
- Maximal ein *process* pro *architecture*
- Maximal eine Typdeklaration pro *architecture*, reseviert für FSMs

```
ENTITY ib_handl IS
  PORT (clock          : IN hbit;
        reqd           : IN hbit;
        requ           : IN hbit;
        iback          : IN hbit;

        ibdu           : OUT hbit;
        ibreq          : OUT hbit);
END ib_handl;


ARCHITECTURE behave_ib_handl OF ib_handl IS

  TYPE statetype IS (WAD_D, RQD_D, WAD_U, RQD_U, WAU_D, RQU_D, WAU_U, RQU_U);
  SIGNAL state, nextstate: statetype;

BEGIN
  PROCESS(state, reqd, requ, iback)
  BEGIN
    CASE state IS
      WHEN WAD_D =>   ibdu  <= '0';
                      ibreq <= '0';
                      IF    (reqd='1') and (iback='0') THEN                 nextstate <= RQD_D;
                      ELSIF (reqd='0') and (requ='1') and (iback='0') THEN nextstate <= RQU_D;
                      ELSE                                                  nextstate <= WAD_D;
                      END IF;
      WHEN RQD_D =>   ibdu  <= '0';
                      ibreq <= '1';
                      IF (reqd='0') and (iback='1') THEN                    nextstate <= WAD_U;
                      ELSE                                                  nextstate <= RQD_D;
                      END IF;
      WHEN WAD_U =>   ibdu  <= '0';
                      ibreq <= '0';
                      IF    (requ='1') and (iback='0') THEN                 nextstate <= RQU_U;
                      ELSIF (reqd='1') and (requ='0') and (iback='0') THEN nextstate <= RQD_U;
                      ELSE                                                  nextstate <= WAD_U;
                      END IF;
      WHEN RQD_U =>   ibdu  <= '0';
                      ibreq <= '1';
                      IF (reqd='0') and (iback='1') THEN                    nextstate <= WAD_D;
                      ELSE                                                  nextstate <= RQD_U;
                      END IF;
      WHEN WAU_D =>   ibdu  <= '1';
                      ibreq <= '0';
                      IF    (reqd='1') and (iback='0') THEN                 nextstate <= RQD_D;
                      ELSIF (reqd='0') and (requ='1') and (iback='0') THEN nextstate <= RQU_D;
                      ELSE                                                  nextstate <= WAU_D;
                      END IF;
      WHEN RQU_D =>   ibdu  <= '1';
                      ibreq <= '1';
                      IF (requ='0') and (iback='1') THEN                    nextstate <= WAU_U;
                      ELSE                                                  nextstate <= RQU_D;
                      END IF;
      WHEN WAU_U =>   ibdu  <= '1';
                      ibreq <= '0';
                      IF    (requ='1') and (iback='0') THEN                 nextstate <= RQU_U;
                      ELSIF (reqd='1') and (requ='0') and (iback='0') THEN nextstate <= RQD_U;
                      ELSE                                                  nextstate <= WAU_U;
                      END IF;
      WHEN RQU_U =>   ibdu  <= '1';
                      ibreq <= '1';
                      IF (requ='0') and (iback='1') THEN                    nextstate <= WAU_D;
                      ELSE                                                  nextstate <= RQU_U;
                      END IF;
    END CASE;

  END PROCESS;

  state <= nextstate WHEN rise(clock) ELSE state;

END behave_ib_handl;
```

```
;*** HINT release 3.00, s/n: 11665, date: 1994/06/23, 10:24:13
;*** Copyright (c) 1990-1993, HARDI Electronics AB, Lund Sweden
*X-NAMES
 CLOCK, REQD, REQU, IBACK;

*Y-NAMES
 IBDU, IBREQ;
 QQ[1..3];

*LOCAL
 ;

*FLOW-TABLE
 $HEADER: X[ REQD, REQU, IBACK] : Y[ IBDU, IBREQ];
 S1:X1-0:Y00:F2;
 S1:X010:Y00:F6;
 S1:X00-:Y00:F1;
 S1:X--1:Y00:F1;
 S2:X0-1:Y01:F3;
 S2:X1--:Y01:F2;
 S2:X--0:Y01:F2;
 S3:X-10:Y00:F8;
 S3:X100:Y00:F4;
 S3:X00-:Y00:F3;
 S3:X--1:Y00:F3;
 S4:X0-1:Y01:F1;
 S4:X1--:Y01:F4;
 S4:X--0:Y01:F4;
 S5:X1-0:Y10:F2;
 S5:X010:Y10:F6;
 S5:X00-:Y10:F5;
 S5:X--1:Y10:F5;
 S6:X-01:Y11:F7;
 S6:X-1-:Y11:F6;
 S6:X--0:Y11:F6;
 S7:X-10:Y10:F8;
 S7:X100:Y10:F4;
 S7:X00-:Y10:F7;
 S7:X--1:Y10:F7;
 S8:X-01:Y11:F5;
 S8:X-1-:Y11:F8;
 S8:X--0:Y11:F8;

 $HEADER: Q[ QQ[1..3]];
 S[1..8]: Q $BINARY;

*BOOLEAN-EQUATIONS
 $QALL.CLK= CLOCK;

*END
```

```
   IBDU    =  QQ1    ;

   IBREQ   =  QQ3    ;

   QQ1.D   := /REQD   &  REQU   & /IBACK  & /QQ3
            +  REQU   & /IBACK  &  QQ2    & /QQ3
            + /REQD   &  QQ1
            +  IBACK  &  QQ1
            +  QQ1    &  QQ3    ;

   QQ2.D   := /REQD   &  IBACK  & /QQ1    & /QQ2    &  QQ3
            +  QQ2    & /QQ3
            + /REQU   &  IBACK  &  QQ1    & /QQ2    &  QQ3
            +  REQD   & /QQ1    &  QQ2
            + /IBACK  &  QQ2
            +  REQU   &  QQ1    &  QQ2    ;

  /QQ3.D   := /REQD   &  IBACK  & /QQ1
            + /REQU   &  IBACK  &  QQ1
            + /REQD   & /REQU   & /QQ3
            +  IBACK  & /QQ3    ;
```


XILINX
IB_HANDL
DATE : 20.06.94
DRAWN BY:MG
M2-1
IBRS
RSD
RSU
IBQU
IBACKU
IBACKO
IBACK
DOELU
DOELO
-DOEL
DOEHU
DOEHO
-DOEH
AND2
AND2B1
NAND2B1
NAND2B2
FDRD
Q
D
CE
C
RD
VDD
Q03
Q02
Q01
IBACK
REQU
REQD
AND2B1
AND3B3
NOR4
AND3B1
AND3B2
AND3
OR3
AND5B2
OR2
AND5B3
OR5
AND4B2
AND4B3
BUF
IBREQD
IBREQU
RQYD
RQYU
READY1
READY2
READY[1:2]
SYSCLK
RESET

Bewertung

- ☞ Ergebnisse
- ☞ Entwicklungsumgebung

Ausblick

- ☞ Fortführung des Projekts
- ☞ Anwendung

VHDL-Entwurf und Synthese einer PLL-Schaltung zur digitalen Lagemessung

Michael Kröner, Andreas Meybohm
Gerald Kampe, Werner Zimmermann
Fachhochschule für Technik Esslingen

1. Einleitung

Zur Lagemessung bei linearen oder rotatorischen Bewegungen, z.B. in Computer-Mäusen oder in Werkzeugmaschinen, werden heute bevorzugt Inkrementalgeber eingesetzt. Dabei kommen sowohl optische Geber mit 'Strichscheiben' und Lichtschranken als auch induktive Geber mit Zahnrädern zum Einsatz. Um mit diesen Gebern neben der Position auch die Bewegungsrichtung erfassen zu können, liefern die Geber in der Regel zwei um eine viertel Periode gegeneinander versetzte Signale.

Die von den Sensoren gelieferten Signale sind näherungsweise sinusförmig, werden aber wegen der einfacheren Signalverarbeitung häufig in Rechtecksignale umgeformt. Die gewünschte Lagemessung kann dann sehr einfach durch digitale Vor/Rückwärtszähler erfolgen. Da die Inkrementzahl der Geber aus mechanischen Gründen begrenzt ist, kann eine erheblich höhere Auflösung des Meßwerts nur erreicht werden, wenn die analogen sinusförmigen Signale direkt ausgewertet werden, anstatt lediglich deren Nulldurchgänge zu verwenden, wie dies bei der Umformung dieser Signale in Rechtecksignale geschieht. Die Bereitstellung eines digitalen Lagemeßwerts wird dabei allerdings erheblich aufwendiger.

Ein mögliches Meßprinzip arbeitet mit einer PLL-Schaltung (Bild 1), bei der über einen Regelkreis das von einem spannungsgesteuerten Oszillator erzeugte Quadratursignal $\sin\varphi$, $\cos\varphi$ dem Winkelgebersignal $\sin\alpha$, $\cos\alpha$ phasengetreu nachgeführt wird. Die 'Regeldifferenz' $u_M = \sin\alpha \cdot \cos\varphi - \sin\varphi \cdot \cos\alpha = \sin(\alpha-\varphi)$ wird dabei in einem Quadraturphasenvergleicher gebildet. Bei geeigneter Auslegung erzwingt der Regler $\alpha \approx \varphi$, auch wenn $\varphi = \varphi(t)$ ein zeitlich veränderlicher Wert ist. Der Phasenwinkel φ des erzeugten Quadratursignals stellt damit den gewünschten Meßwert für den Phasenwinkel α dar.

Dieser Phasenregelkreis war im Rahmen von zwei Diplomarbeiten [1,2] in VHDL zu beschreiben, zu simulieren und zu optimieren sowie in eine digitale Schaltung umzusetzen. Als Werkzeuge wurden die VHDL-Umgebung und die Simulations- und Syntheseprogramme des MENTOR-CAE-Systems eingesetzt. Die gestellten Anforderungen (Bild 1) entsprechen einem Winkelmeßsystem mit einem Geber mit 1024 Inkrementen je Umdrehung bei einer maximalen

Drehzahl von 10000 1/min. Als Winkelauflösung des Meßsystems waren $32 \cdot 1024$ Schritte je Umdrehung bei Maximaldrehzahl (=max. Eingangsfrequenz der sinusförmigen Signale) schrittweise zunehmend bis zu $1024 \cdot 1024$ Schritte bei sehr niedrigen Drehzahlen gefordert. Bei Verarbeitung von Rechtecksignalen wäre dagegen lediglich eine Auflösung von $4 \cdot 1024$ Schritten je Umdrehung möglich.

2. Digitaler Phasenregelkreis

Im Phasenvergleicher müssen die Produkte $\sin\alpha \cdot \cos\varphi$ und $\sin\varphi \cdot \cos\alpha$ der beiden Quadratursignale $\sin\alpha$, $\cos\alpha$ und $\sin\varphi$, $\cos\varphi$ sowie deren Summe (bzw. Differenz) gebildet werden (Bild 2). Der digitale Schaltungsaufwand für die beiden Multiplikationen wird deutlich reduziert, wenn lediglich die Signale $\sin\varphi$, $\cos\varphi$ als n bit breite digitale Parallelworte vorliegen und die Sensorsignale $\sin\alpha$, $\cos\alpha$ als 1 bit breite serielle Datenströme (Bitstrom) verarbeitet werden (Bild 3). Dies vereinfacht auch die Analog-Digitalumsetzung der ursprünglich analogen Sensorsignale. Statt schneller paralleler A/D-Wandler sind lediglich 1-bit-Wandler erforderlich, wie sie in Form von Sigma-Delta-(Σ/Δ)-Wandlern auch in digitalen Schaltkreistechnologien verhältnismässig einfach hergestellt werden können. (Bemerkung: Die Sigma-Delta-Wandler wurden als Verhaltensmodell in VHDL beschrieben, aber nicht synthetisiert).

Der steuerbare Oszillator wird durch einen Vor-Rückwärtszähler mit fester Taktfrequenz realisiert, der ein Sinus-Cosinus-ROM ansteuert (Bild 2). Die Zählrichtung wird durch einen Zweipunktregler 'moduliert' und dadurch die (mittlere) Zählfrequenz eingestellt. Die Regeldifferenz $\sin(\alpha-\varphi)$ am Eingang des Zweipunktreglers muß aufgrund der '1-bit-Multiplikationen' noch tiefpaßgefiltert werden.

Der Zählerstand φ des Vor-Rückwärtszählers stellt gleichzeitig den gewünschten digitalen Meßwert für die Winkelposition α dar.

3. Verhaltenssimulation

Der Regelkreis nach Bild 2 wurde in VHDL beschrieben und mit Mentor Quicksim simuliert. Bild 4 zeigt einen simulierten Einschwingvorgang des Meßsystems bei einer niedrigen Eingangsfrequenz. Dabei ist im oberen Signal auch die in Bild 2 nicht dargestellte automatische Anpassung der Meßauflösung an die Eingangsfrequenz zu sehen. Die Wortbreite der einzelnen Datenpfade, die Größe der ROM-Tabelle sowie die Struktur und die Koeffizienten des Tiefpaßfilters wurden mit Hilfe der Simulation optimiert.

4. Schaltungssynthese

Ausgehend von der Verhaltensbeschreibung wurde die Schaltung schrittweise verfeinert, in eine Datenflußbeschreibung umgesetzt und anschließend blockweise synthetisiert und optimiert. Am Beispiel des Addierers aus der Phasenvergleicherstufe in Bild 2 läßt sich gut nachvollziehen, wie die VHDL-Beschreibung (Bild 5) durch Autologic in eine Schaltung umgesetzt wird (Bild 6). Die Auffächerung der Wortbreite, der eigentliche Additionsbefehl (in Bild 6 nur als Block dargestellt) sowie die Anbindung an das Taktsignal innerhalb des Prozeßstatements sind im Schaltbild unmittelbar wiederzuerkennen.

Die Parametereinstellung des Schaltungsoptimierers hat erheblichen Einfluß auf das Ergebnis. Mit den standardmäßigen Voreinstellungen liefert die Optimierung der Additionsstufe eine Schaltung mit 138 Gattern (Bild 7 oben), die allerdings eine Durchlaufverzögerung von fast 100ns aufweist. Da für den gesamten Regelkreis eine Taktperiode von 100ns gefordert war, mußte die Verzögerungszeit durch Verringerung der sequentiellen Tiefe der Schaltung reduziert werden. Durch Wahl eines 'Look ahead' von 18bit (statt 2bit) konnte die Verzögerungszeit auf 30ns reduziert werden (Bild 7 unten), allerdings vervierfachte sich die Gatterzahl mit 456 nahezu (Bild 8).

Der Einfluß der VHDL-Beschreibung auf das Syntheseergebnis soll an einem Beispiel gezeigt werden. Im Bild 9 unten ist eine Multiplikationsschaltung in VHDL beschrieben, bei der ein 17bit breiter Multiplikand mit einem 1bit breiten Multiplikator multipliziert wird, der die Werte ± 1 (entsprechen logisch 0 bzw. 1) annehmen kann. Da der Multiplikand in Betrag-Vorzeichendarstellung vorliegt, muß lediglich das Vorzeichenbit in Abhängigkeit des Multiplikators invertiert werden (case-Befehl), während die 16 Betragsbits unverändert übernommen werden können (for-Befehl). Das Syntheseergebnis weist erwartungsgemäß einen Inverter, einen 2-zu-1-Multiplexer und 16 direkte Leitungsverbindungen auf (Bild 9 mitte links). Verlegt man dagegen die Lage der for-Schleife in den case-Befehl, so ergibt sich zwar eine exakt identische logische Funktion, die Schaltungssynthese liefert jetzt allerdings 16 weitere 2-zu-1-Multiplexer (Bild 9 mitte rechts). Erst nach einem anschließenden Optimierungslauf erkennt Autologic, daß die 16 zusätzlichen Multiplexer eigentlich unnötig sind und entfernt sie wieder.

5. Zusammenfassung

Im Rahmen von zwei Diplomarbeiten wurde das Schaltungsprinzip eines digitalen Lagemeßsystems auf Basis eines phasenstarren Regelkreises entworfen. Dabei zeigte sich, daß sich Verhaltensbeschreibungen in VHDL und Simulationen mit MENTOR Quicksim nicht nur für die klassische digitale Schaltungstechnik sondern auch für regelungstechnische Untersuchungen

quasi-analoger Vorgänge eignen. Die Verhaltensbeschreibung wurde schrittweise verfeinert, die Funktionsgruppen synthetisiert und optimiert. Die geforderten Zielwerte konnten in der Simulation mit einem Aufwand von ca. 2500 Gattern auf Basis der IMS 1,2μ-Gate-Forest-Bibliothek erreicht werden. Die Durchlaufzeit der Schaltung ohne das ROM betrug 95ns.

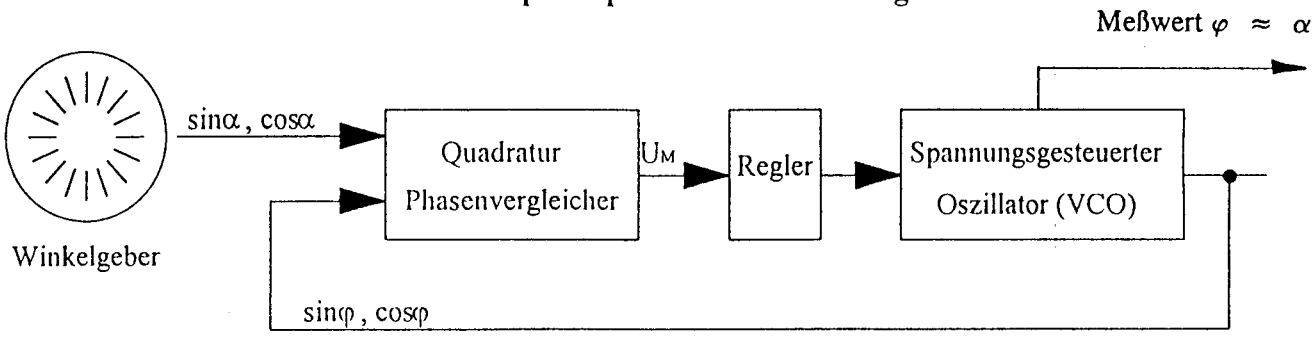
Ziel der weiterführenden Arbeiten wird es sein, die Schaltung um ein mikroprozessorfähiges Businterface zu ergänzen, die ROM-Baugruppe zu eliminieren und die Genauigkeit der Schaltung noch zu steigern, bevor die Schaltung für die Fertigung am IMS freigegeben wird.

Literatur

[1] Kröner, Michael: Schaltungsentwurf und Synthese einer digitalen PLL mit Hilfe von VHDL. Diplomarbeit, Fachhochschule für Technik Esslingen, 1994

[2] Meybohm, Andreas: IC-Synthese und Optimierung eines Schaltungsentwurfs in VHDL. Diplomarbeit, Fachhochschule für Technik Esslingen, 1994

Bild 1 **Grundprinzip der Winkelmessung**



Anforderungen:

Frequenz der sinusförmigen Eingangssignale:
f_I = 0..170 kHz

Taktfrequenz:
10 MHz

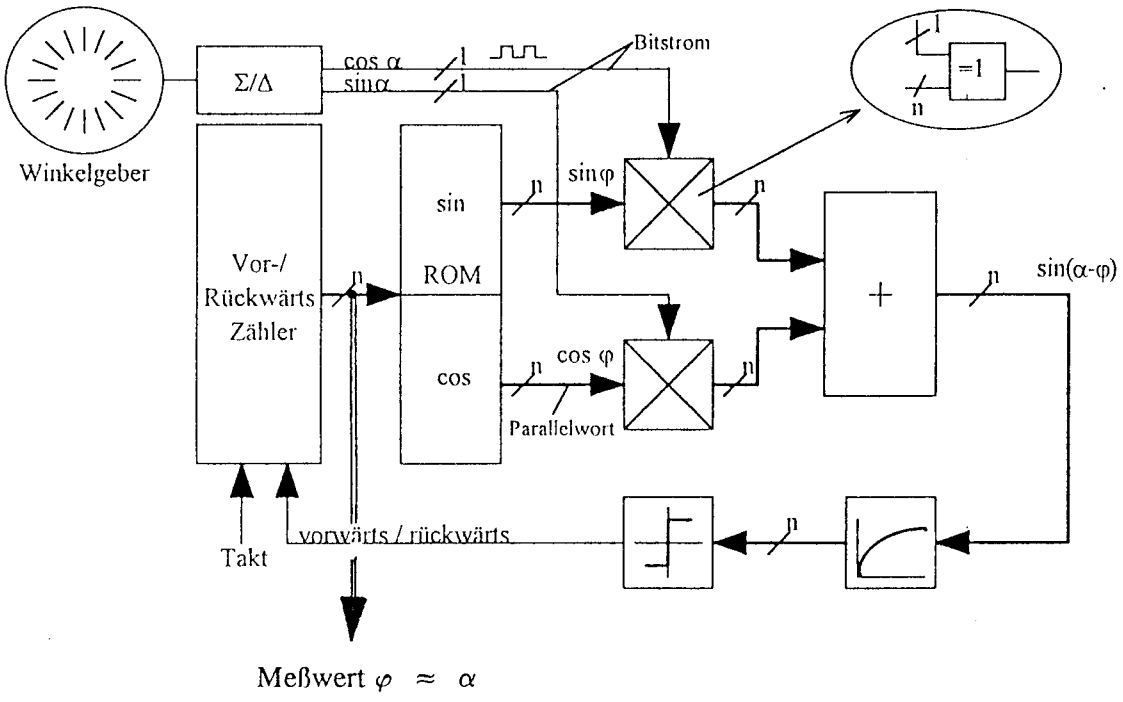
Auflösung:
16..32 Schritte je Periode bei f_I = 170 kHz
. . .
1024 Schritte je Periode bei f_I = 0 kHz

Zieltechnologie:
IMS Gate-Forest 1,2µ

im eingeschwungenen Zustand:

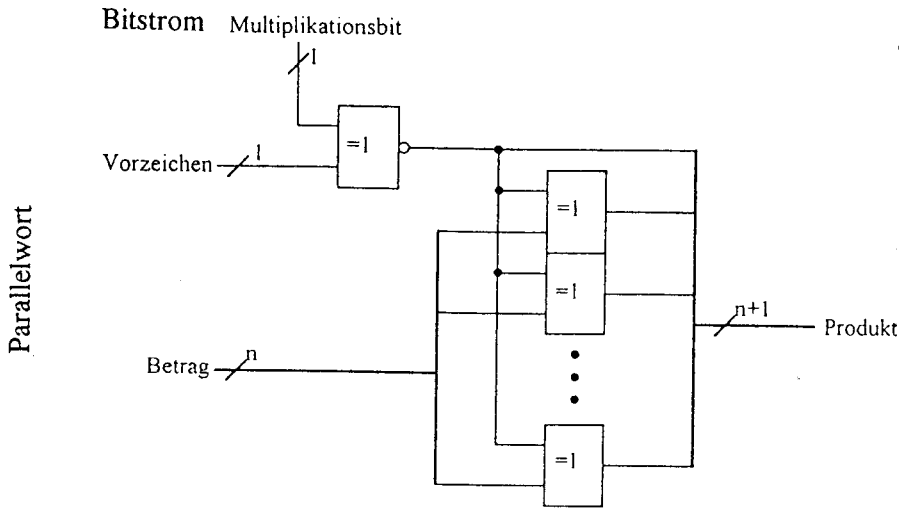
$$U_M = \sin\alpha\cdot\cos\varphi - \sin\varphi\cdot\cos\alpha = \sin(\alpha-\varphi) = 0 \quad \rightarrow \quad \alpha = \varphi$$

Bild 2 **Schaltung der digitalen PLL**



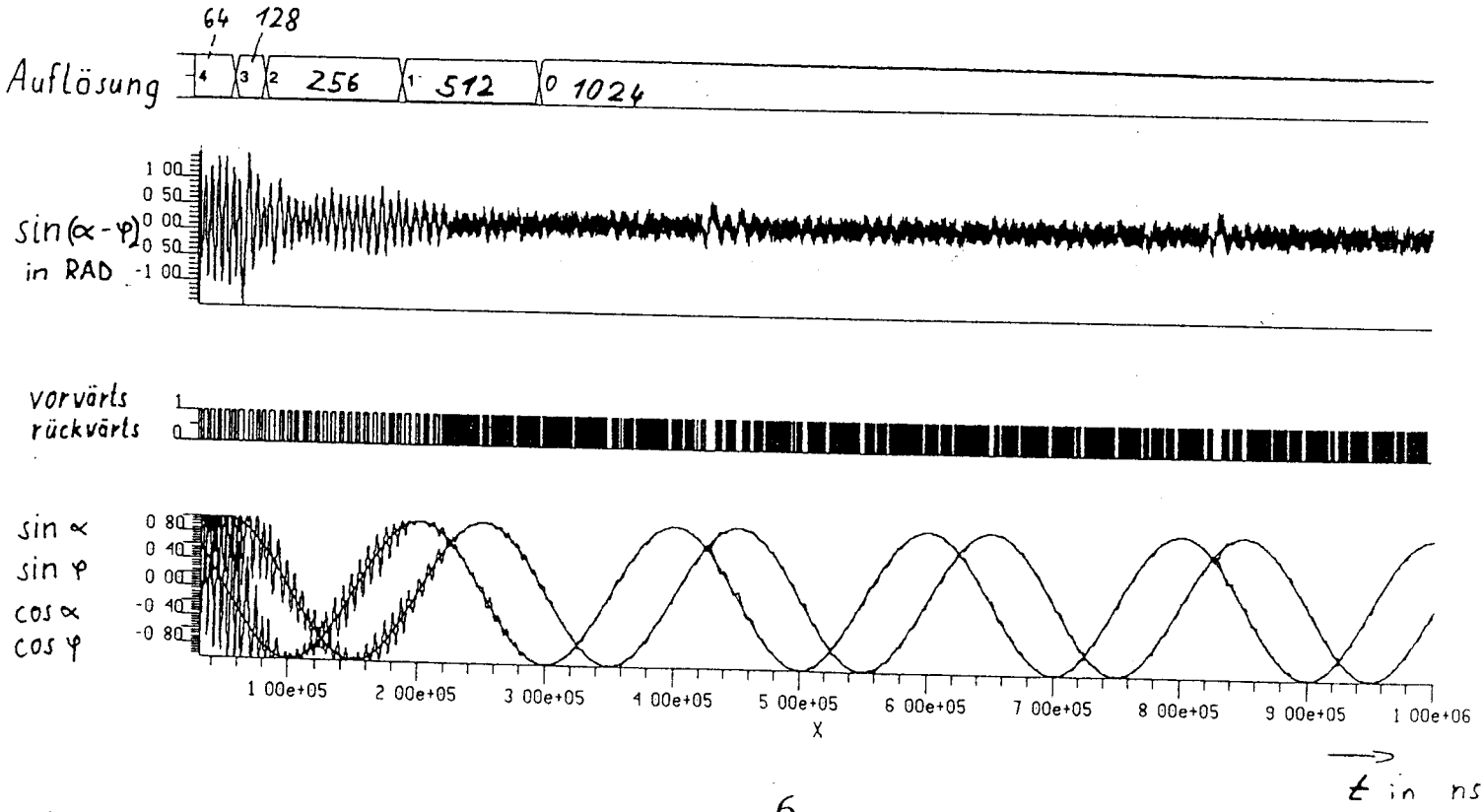
digitale Bereitstellung des Winkelmeßwertes

Bild 3 Realisierung des digitalen Multiplizierers



Werte aus dem ROM bestehen nur aus Betrag
Vorzeichen wird vom Zähler geliefert, wechselt nach jedem Durchlauf

Bild 4 Simulationsergebnis



```
------------------------------------------------------------------
--
--  Modellbeschreibung:
--    Addition von zwei 17 Bit breiten Dualzahlen
--
------------------------------------------------------------------
--
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;

ENTITY add IS
  PORT (clk      : IN  qsim_state;
        summand1 : IN  qsim_state_vector(16 DOWNTO 0);
        summand2 : IN  qsim_state_vector(16 DOWNTO 0);
        addout   : OUT qsim_state_vector(17 DOWNTO 0));
END add;


ARCHITECTURE behav OF add IS
  SIGNAL sum1, sum2 : qsim_state_vector(17 DOWNTO 0);
BEGIN
  sum1(17)           <= summand1(16);
  sum1(16 DOWNTO 0)  <= summand1;
  sum2(17)           <= summand2(16);
  sum2(16 DOWNTO 0)  <= summand2;
  PROCESS
  BEGIN
    WAIT UNTIL (clk'EVENT AND clk'LAST_VALUE = '0' AND clk = '1');
      addout <= (sum1+sum2);
  END PROCESS;
END behav;
```

Auffächerung

Speicherung

Addierer

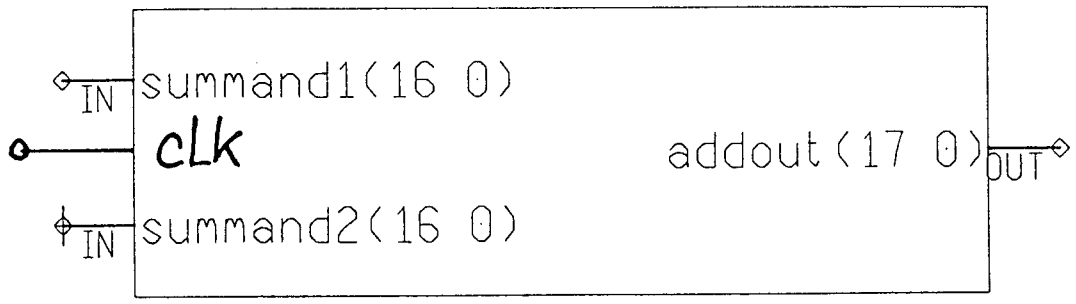
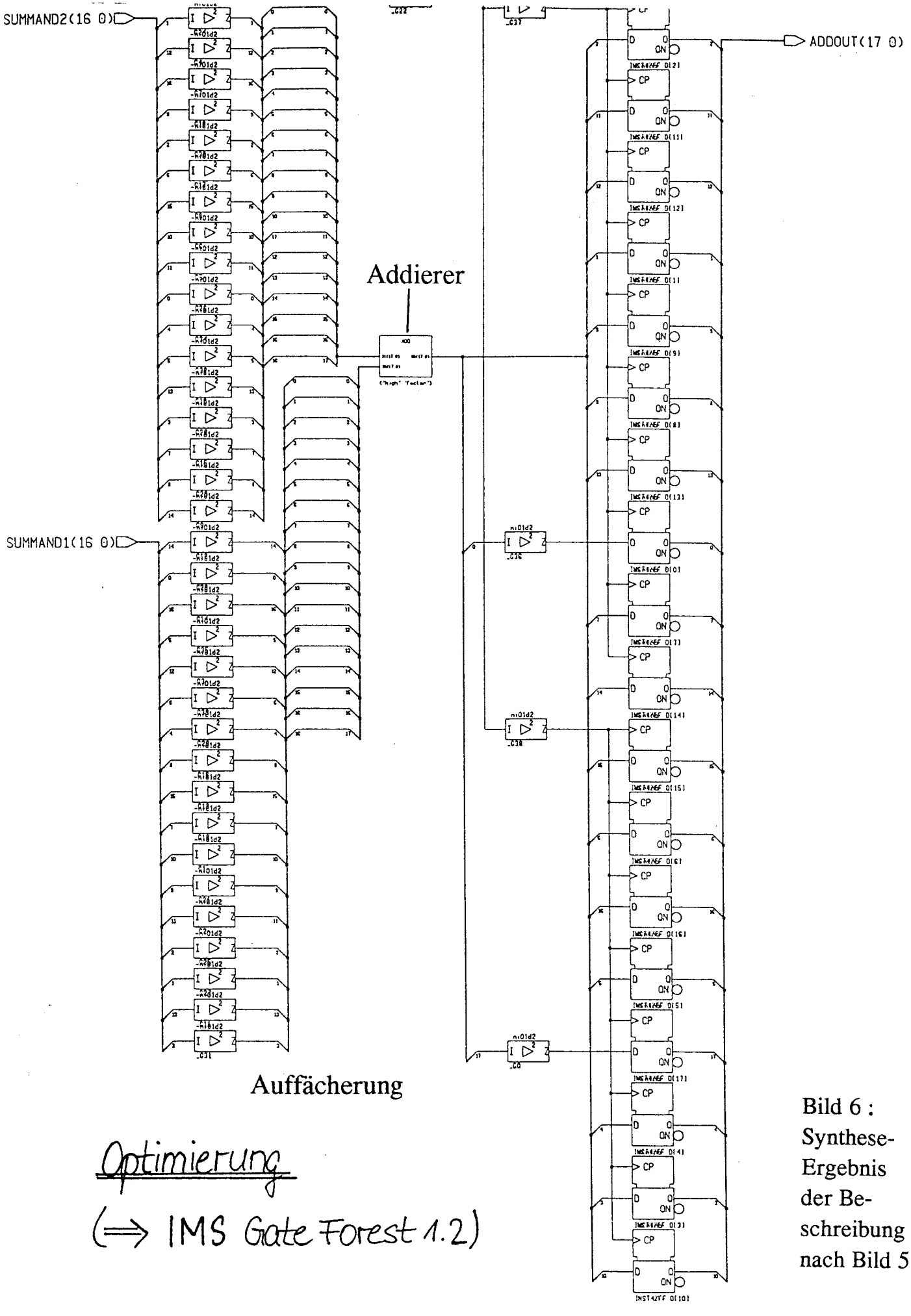


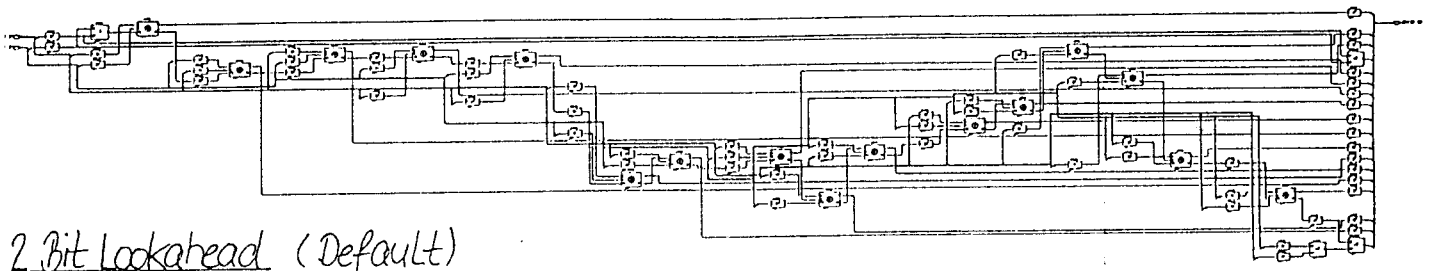
Bild 5 VHDL-Beschreibung des Addierers



Optimierung

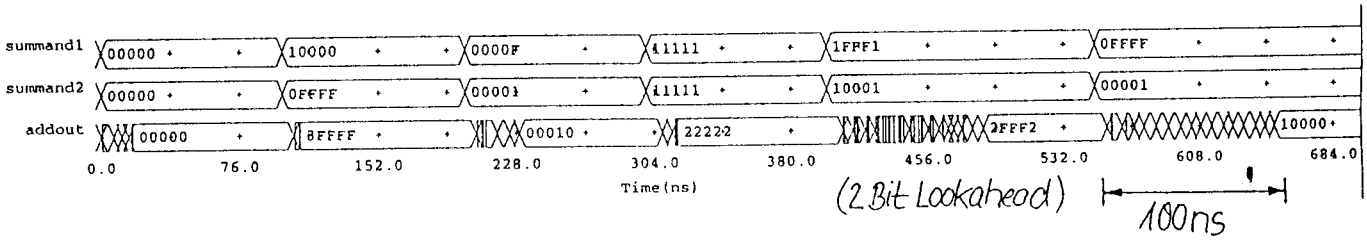
(⟹ IMS Gate Forest 1.2)

Bild 6 : Synthese-Ergebnis der Beschreibung nach Bild 5



2 Bit Lookahead (Default)

(⟹ IMS Gate Forest 1.2) ; 138 Gatter



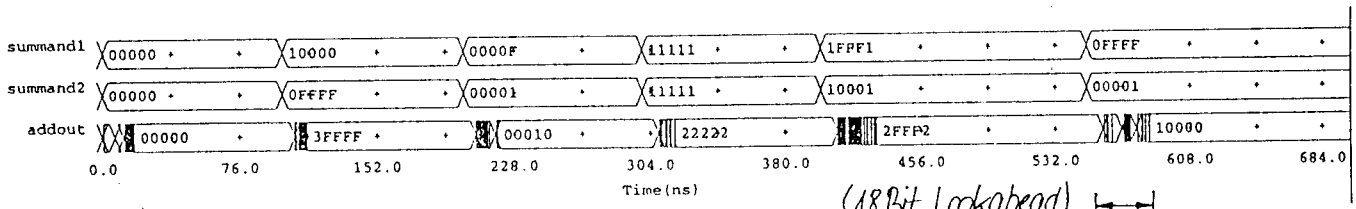
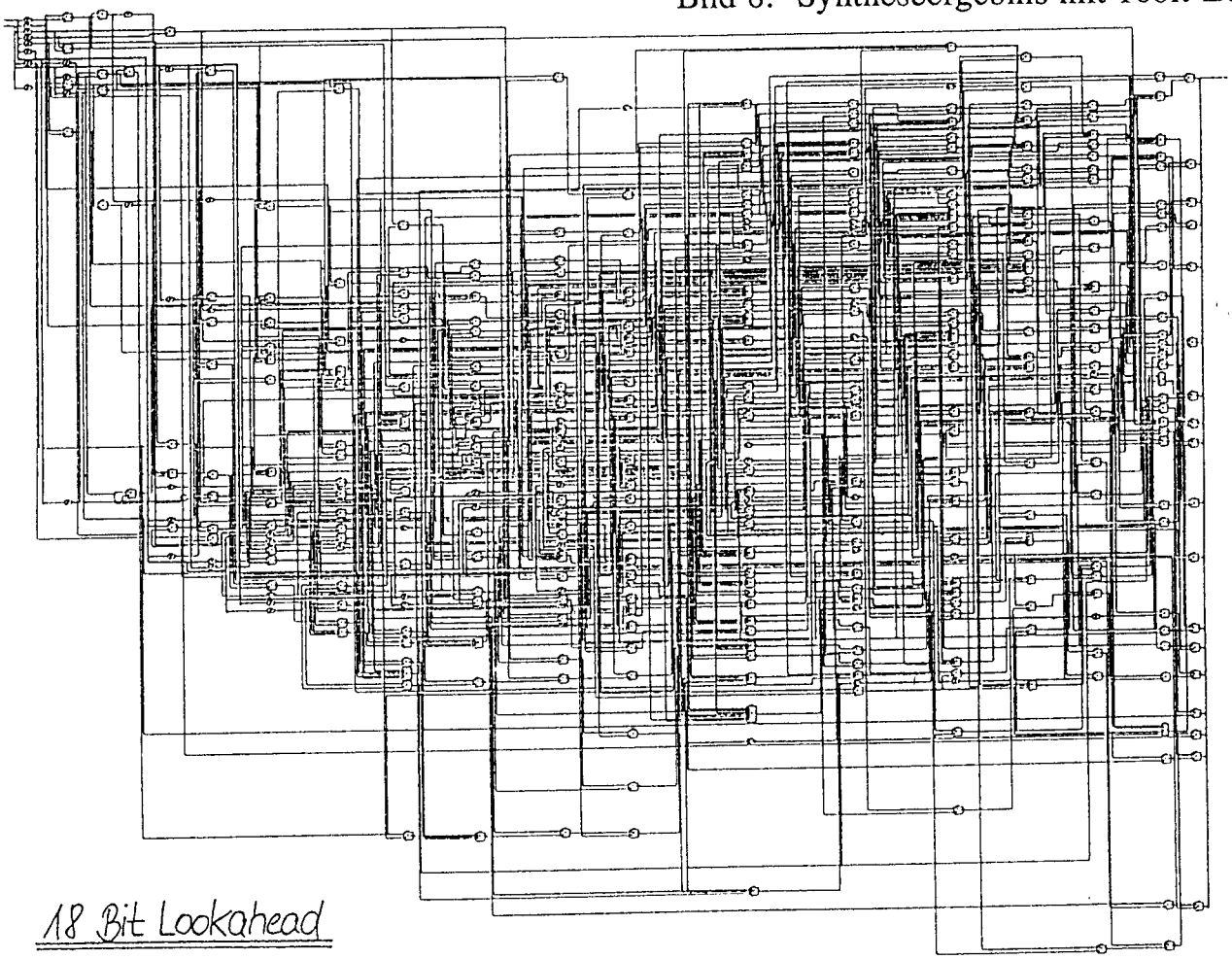


Bild 7: Laufzeitvergleich

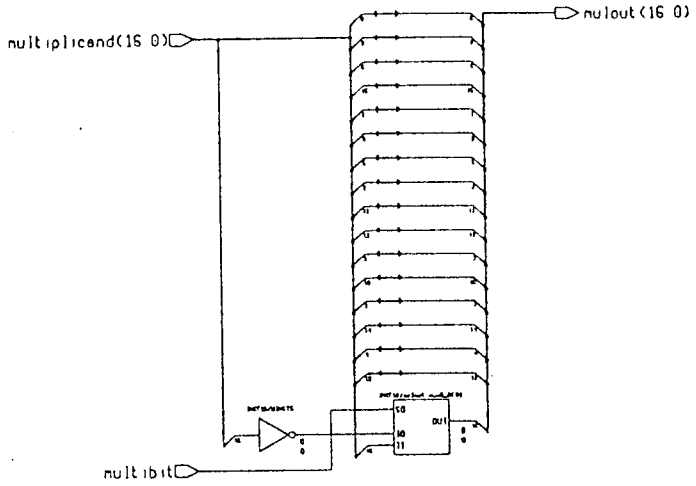
Bild 8: Syntheseergebnis mit 18bit-Look ahead

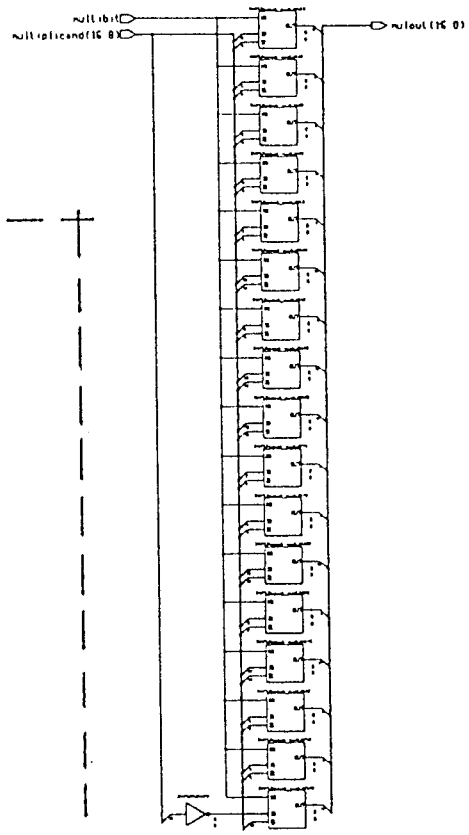


18 Bit Lookahead

(⟹ IMS Gate Forest 1.2) ; 456 Gatter

```
ARCHITECTURE behav OF mulvb IS
BEGIN
  PROCESS (multibit,multiplicand)
    VARIABLE i : INTEGER RANGE 0 TO 15 := 0;
  BEGIN
    CASE multibit IS
      WHEN '1' => mulout      <= multiplicand;
      WHEN '0' => mulout(16) <= NOT multiplicand(16);
        FOR i IN 0 TO 15 LOOP
          mulout(i) <= multiplicand(i);
        END LOOP;
      WHEN OTHERS => NULL;
    END CASE;
  END PROCESS;
END behav;
```





```
ARCHITECTURE behav OF mulvb IS
BEGIN
  PROCESS (multibit,multiplicand)
    VARIABLE i : INTEGER RANGE 0 TO 15 := 0;
  BEGIN
    CASE multibit IS
      WHEN '1' => mulout(16) <= multiplicand(16);
      WHEN '0' => mulout(16) <= NOT multiplicand(16);
      WHEN OTHERS => NULL;
    END CASE;
    FOR i IN 0 TO 15 LOOP
      mulout(i) <= multiplicand(i);
    END LOOP;
  END PROCESS;
END behav;
```

Bild 9:
Syntheseergebnis in Abhängigkeit von der VHDL-Beschreibung

FACHHOCHSCHULE FÜR WIRTSCHAFT UND TECHNIK REUTLINGEN
-FACHBEREICH ELEKTRONIK-

WORKSHOP DER MPC-GRUPPE SS94

Realisierung einer Multi-Ported-Memory-Schaltung mit VHDL auf einem FPGA der Firma ACTEL

Carsten Sigwarth
Trottäcker 16
79713 Bad Säckingen

1.0 Einleitung

1.1 Aufgabenstellung

In dieser Diplomarbeit wird der Ablauf von einem VHDL-Code zu einem FPGA an einer vorgegebenen Schaltung durchgeführt und beschrieben. Hierfür steht ein VHDL-Code zur Verfügung, der eine diskret aufgebaute Multi-Port-Memory-Schaltung (MPM) der Firma Phoenix Contact beschreibt. Diese Schaltung soll in ein FPGA integriert werden.

Besonderen Wert wird in dieser Arbeit dabei auf den Schritt von der synthetisierten Schaltung bis zum fertigen FPGA gelegt. Der Schritt vom VHDL-Code zur synthetisierten Schaltung wurde schon in der Diplomarbeit von Uwe Feist [2] ausführlich beschrieben und kann vom interessierten Leser dort nachgelesen werden.

Für die Realisierung dieser Arbeit stehen folgende Hilfsmittel zur Verfügung:

HP 700 Workstations

Mentor Graphics Version 8.2_5

ACTEL-Soft- und Hardware

Schaltpläne der Firma Phoenix Contact

2.0 Das MPM

2.1 Prinzipielle Funktionsweise

Das MPM ist eine Schaltung, die es vier Teilnehmern ermöglicht, Daten über einen Speicher auszutauschen. Voraussetzung ist, daß die Teilnehmer über einen 8 oder 16 Bit Datenbus verfügen und daß die Zugriffe der Teilnehmer verzögert werden können.

Das MPM verwaltet die Zugriffe der einzelnen Teilnehmer. Hat ein Teilnehmer den Zugriff auf das MPM erhalten, so sperrt das MPM die Zugriffe der anderen Teilnehmer, bis der momentan aktive Teilnehmer seine Zugriffe beendet hat. Fordern mehrere Teilnehmer gleichzeitig den Zugriff auf das MPM, so erhalten die Teilnehmer die Zugriffsrechte nach Prioritäten verteilt.

Jeder Teilnehmer kann, je nach Bussystem, verschieden große Zugriffe auf das MPM machen, ohne dabei von anderen Teilnehmern unterbrochen zu werden. Für Word-Teilnehmer sind dies Einfach- Zweifach- und Dreifachzugriffe, Byteteilnehmer dagegen haben die Möglichkeit Einfach-, Vierfach- und Sechsfachzugriffe durchzuführen. Führt ein Teilnehmer nicht alle seine angemeldeten Zugriffe durch, so würde er das MPM für die restlichen Teilnehmer blockieren. Um dies zu verhindern, wird nach einer festgelegten Zeitspanne ein TIMEOUT-Signal erzeugt, welches das MPM dann wieder freigibt. Die Zeitdauer kann über vier Jumper eingestellt werden.

Zusätzlich bietet das MPM den angeschlossenen Teilnehmern die Möglichkeit, untereinander quittierte Nachrichten auszutauschen. Hierzu können sie im Handshakeregister einzelne Bits setzen. Jeder Teilnehmer hat eine bestimmte Anzahl von Handshakebits, mit denen er zum einen anderen Teilnehmern mitteilen kann, daß für sie eine Nachricht im MPM steht oder den Erhalt einer Nachricht quittieren kann.

Den Status der angeschlossenen Teilnehmer und des MPM's kann jeder Teilnehmer im Statusregister nachlesen.

Ausführlichere Informationen über die Funktion des MPM's erhält der interessierte Leser in der Diplomarbeit von Andreas Hölz [3], sowie in der Entwicklungsdokumentation der Firma Phoenix Contact [1].

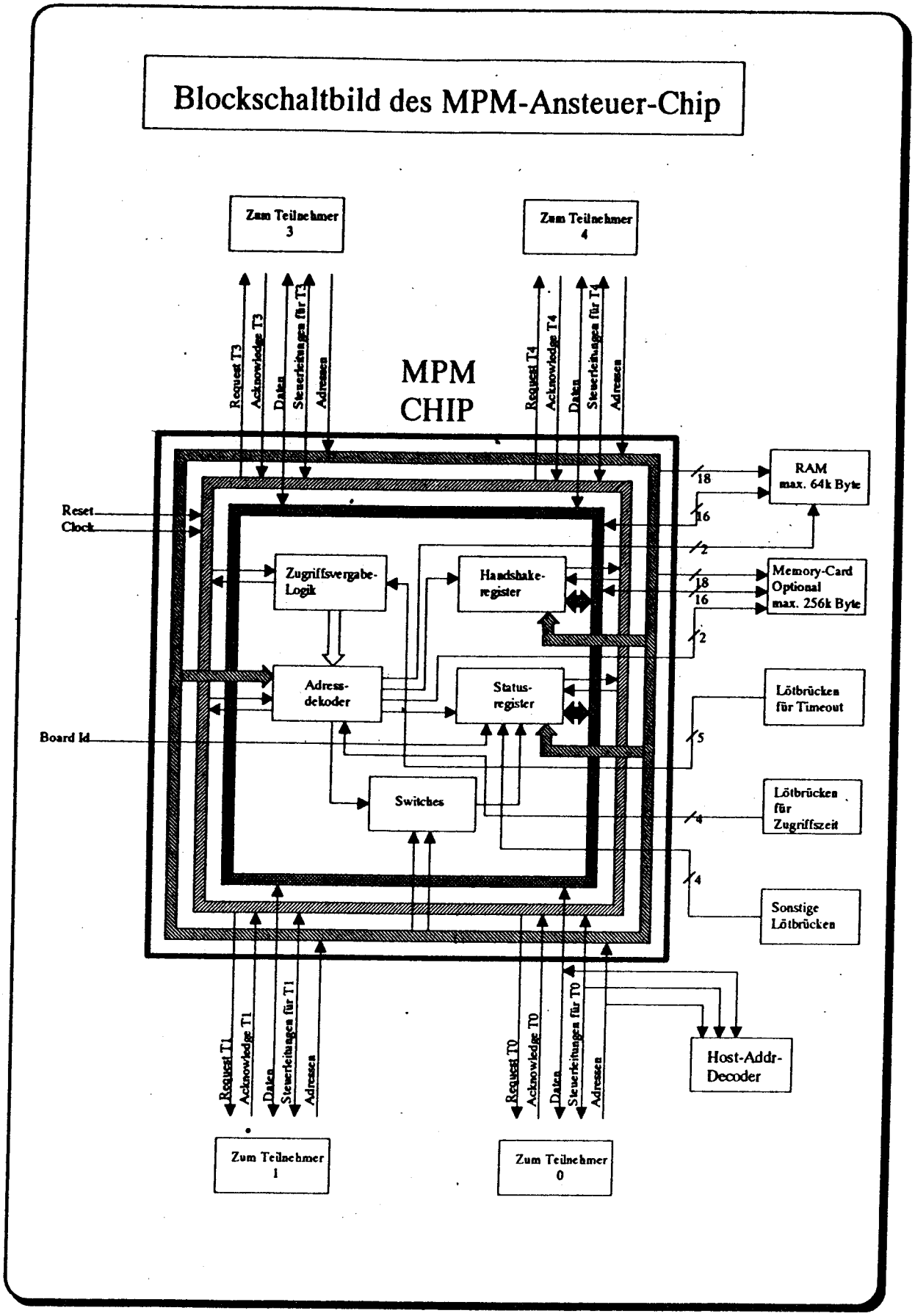


ABBILDUNG 1. Blockschaltbild des MPM

3.0 Ablauf vom VHDL-Code zum FPGA

3.1 Allgemeiner Ablauf

Die Entwicklung eines FPGA's der Firma ACTEL erfolgt in zwei verschiedenen Entwicklungsumgebungen, die durch definierte Schnittstellen miteinander verknüpft sind. In Abbildung 2 ist der allgemeine Ablauf vom Entwurf bis zum fertigen FPGA zu sehen.

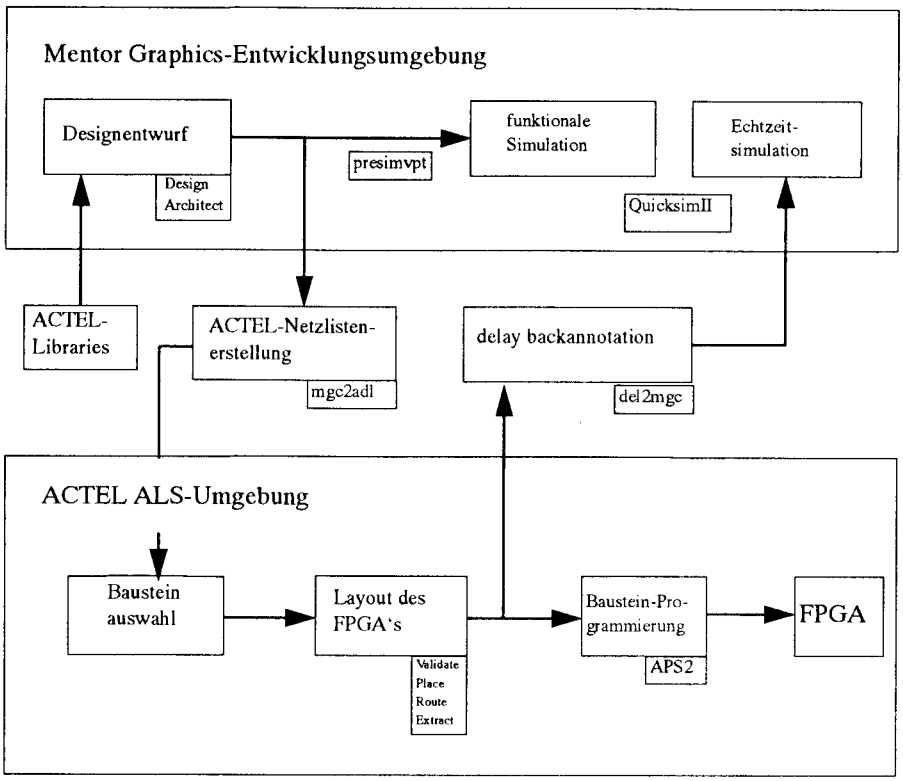


ABBILDUNG 2. Designfluß beim FPGA-Entwurf

Der eigentliche Designentwurf, sowie die Simulation der Schaltung erfolgen in einer Designentwicklungsumgebung, in diesem Fall mit den Entwicklungstools von **Mentor Graphics**. In ihr wird die Schaltung entwickelt und auf ihre korrekte Funktion getestet. Die Eingabe der Schaltung kann auf unterschiedliche Weise erfolgen, direkt als Schematic mit den ACTEL-Bausteinen, als VHDL-Code oder als bool'sche Gleichung, um nur ein paar Möglichkeiten zu nennen. Die Umsetzung auf die Zieltechnologie erfolgt durch das Softwaretool **Autologic** von **Mentor Graphics**. Die Optimierung auf Fläche und Geschwindigkeit kann auch durch das Tool **ACTmap FPGA Fitter** der Firma ACTEL erfolgen.

Ist die Schaltung funktional komplett entwickelt, wird sie über eine vordefinierte Schnittstelle in die Sprache übersetzt, die in der ALS-Umgebung (ACTION LOGIC SYSTEM) von ACTEL verwendet wird.

In dieser Umgebung kann das FPGA dann plaziert und verdrahtet werden. Ebenso werden hier die Daten für die Echtzeitsimulation und das Schießen des FPGA's erzeugt.

Über eine weitere Schnittstelle können die Daten für die Echtzeitsimulation wieder in die Mentor-spezifische Sprache übersetzt werden, so daß dort dann eine Echtzeitsimulation durchgeführt werden kann.

In den nachfolgenden Kapiteln wird der Ablauf eines FPGA-Entwurf mit VHDL ausführlich beschrieben.

3.2 Die Mentor Graphics-Entwicklungsumgebung

Für die Entwicklung einer Schaltung stehen in der Mentor Graphics-Entwicklungsumgebung mehrere Tools zur Verfügung.

- **Design Architect**: Eingabe der Schaltung
- **Quicksim**: Simulation der Schaltung
- **Autologic**: Synthese einer Schaltung aus einer Beschreibungssprache oder Optimieren einer Schaltung auf Fläche und Geschwindigkeit

Alle Softwaretools zur Entwicklung einer Schaltung werden vom Design Manager aus gestartet. Ebenso werden hier die Dateien erstellt und verwaltet. Dieses Tool sollte daher als erstes aufgerufen werden, nachdem man den Rechner eingeschaltet hat.

Hierzu drückt man die rechte Maustaste und hält sie gedrückt. Daraufhin öffnet das WORKSPACE MENU. Hier wählt man dann **MGC Applications > Design Manager** und läßt die Maustaste wieder los. Anschließend öffnet sich der Design Manager. (Abbildung 3)

Der Design Manager hat zwei Fenster, das Toolfenster, in dem alle Programme aufgeführt sind und das Navigatorfenster, in dem die einzelnen Dateien angezeigt werden.

Um die einzelnen Tools für die Entwicklung einer Schaltung aufzurufen, gibt es zwei Möglichkeiten. Zum einen kann man die einzelnen Programme im Toolfenster durch doppelklicken mit der linken Maustaste direkt ausrufen. Zum anderen kann man die Programme im Navigatorfenster auch direkt vom jeweiligen Arbeitsverzeichnis aus aufrufen. Hierzu markiert man das entsprechende Arbeitsverzeichnis. Mit der rechten Maustaste kann man dann das entsprechende Programm mit **Open > *Programm*** aufrufen.

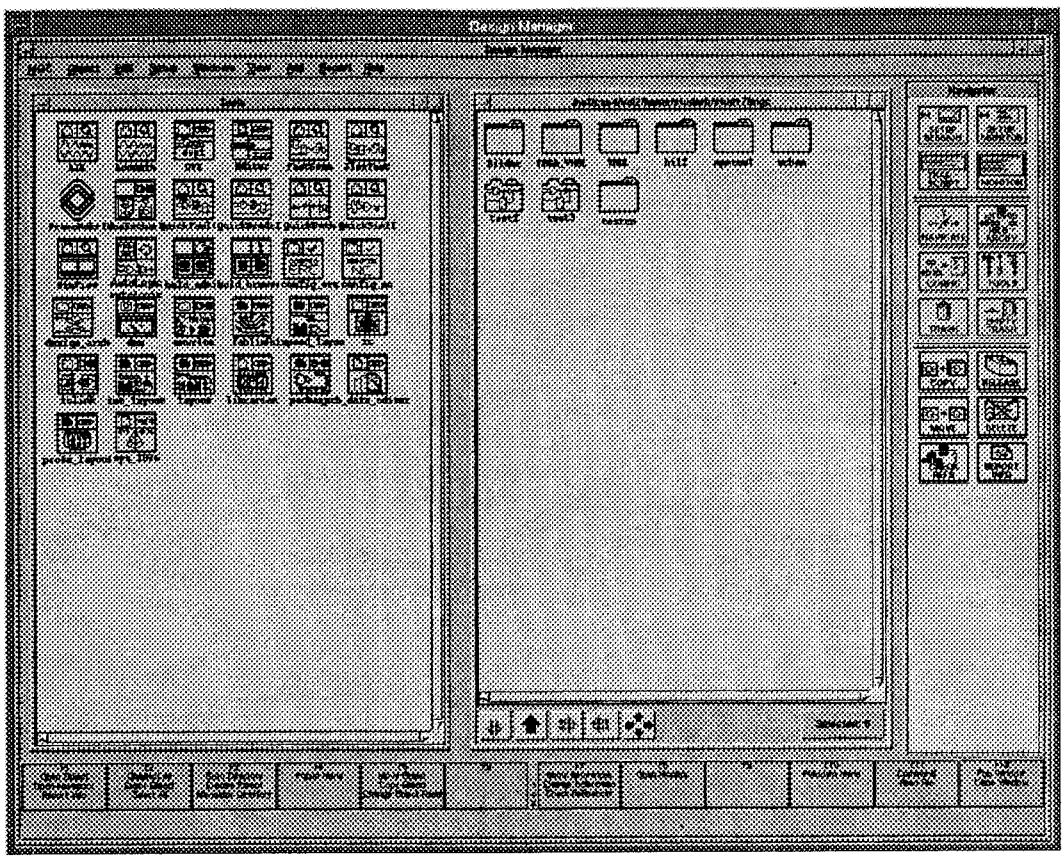


ABBILDUNG 3. Design Manager

3.3 Designentwurf

Abbildung4 zeigt den vollständigen Ablauf eines Designentwurf für ein ACTEL-Design in Mentor Graphics. Er unterscheidet sich nur durch das zusätzliche Erstellen eines Designviewpoints für die Simulation einer ACTEL-Schaltung von dem normalen Vorgehen bei einem Designentwurf.

Die weiteren Besonderheiten, die beim Entwurf eines ACTEL-Designs zu beachten sind, werden in den nachfolgenden Unterkapiteln jeweils aufgeführt.

Auf die einzelnen Schritte zum Designentwurf wird nur stichwortartig eingegangen. Eine genaue Beschreibung der Funktion der Mentortools findet der interessierte Leser in der Diplomarbeit von Uwe Feist [2].

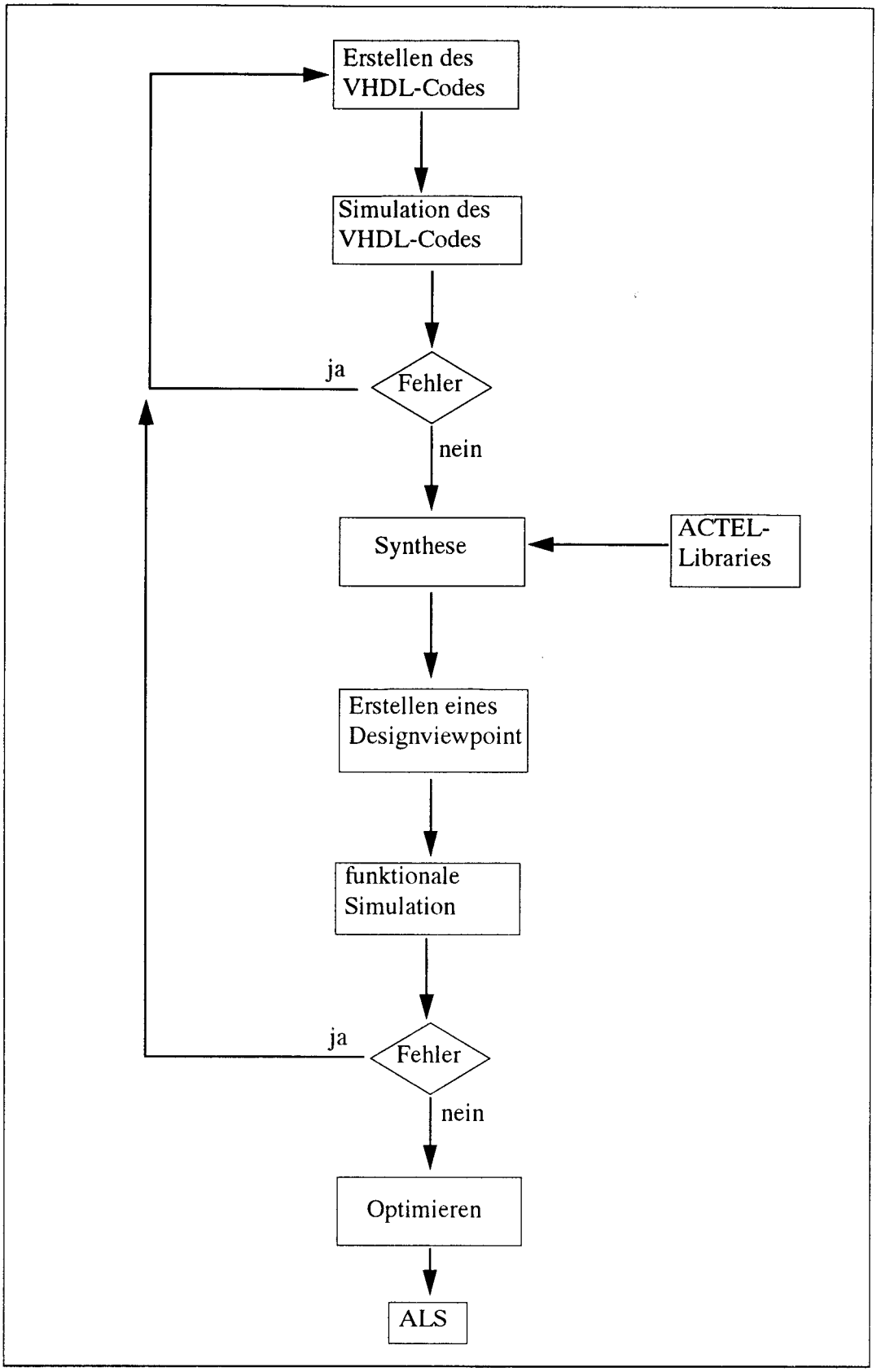


ABBILDUNG 4. Designentwurf

3.3.1 Erstellen des VHDL-Codes

Der VHDL-Code wird im Designarchitect erstellt. Hierzu öffnet man den Design Architect aus dem Design Manager.

Setzen des Arbeitsverzeichnisses

Nachdem der Design Architect geöffnet ist, setzt man mit **MGC > Location Map > Set Working Directory** das Verzeichnis als Arbeitsverzeichnis, in das man seinen VHDL-Code speichern will.

Öffnen des VHDL-Editors

Anschließend kann man durch Drücken des **IKON Open VHDL** in der Session Palette den VHDL-Editor öffnen. In der Dialogbox, die sich dann öffnet, muß man nur noch den Namen für den Sourcecode eingeben und OK drücken (Abbildung 5). In dem VHDL-Editor kann man nun den Code eingeben.

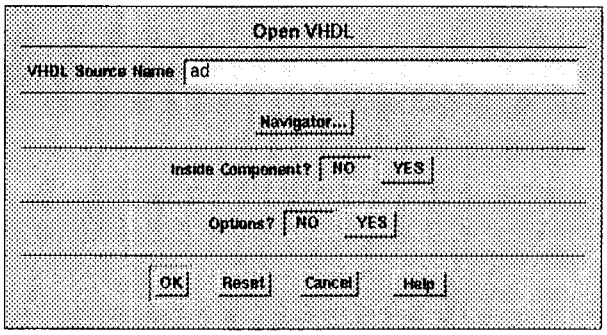


ABBILDUNG 5. Dialogbox Open VHDL

Compilieren des VHDL-Codes

Nachdem der Code im Editor eingegeben wurde, muß er compiliert werden. Zuerst muß mit **Compile > Set Options** in der Dialogbox (Abbildung 6) die Optionen für den Compiler gesetzt werden.

Will man den Code später synthetisieren, so muß man den Schalter **Synthesis** setzen. In der Zeile **Work Lib** muß der Name des aktuellen Arbeitsverzeichnisses stehen. Anschließend kann mit **Compile > Compile** der VHDL-Code compiliert werden.

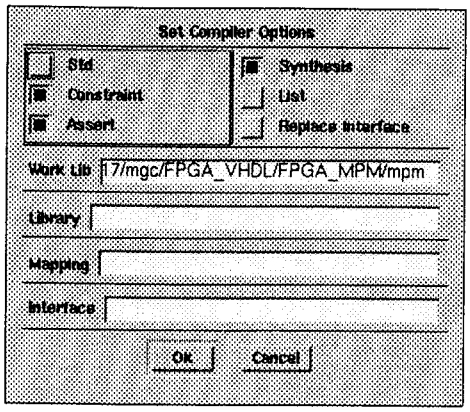


ABBILDUNG 6. Einstellung der Compiler Options

Besonderheiten bei ACTEL-Designs

Will man mit einem VHDL-Code ein ACTEL-Design entwerfen, so muß man zwei Besonderheiten beachten.

1. In der ACTEL-Library gibt es keine Tristatetreiber, die in der Schaltung, z.B. zum Ansteuern von Bussen, verwendet werden können. Hat man in seinem Schaltungsentwurf Tristatetreiber vorgesehen, so muß man im VHDL-Code die Tristatestrukturen durch Multiplexerstrukturen ersetzen.

   Berücksichtigt man dies beim Designentwurf nicht, so werden spätestens beim Synthetisieren der Schaltung die Tristatestrukturen durch Multiplexerstrukturen ersetzt.

2. Die Ansteuersignale für die Padtypen, die man in der Schaltung verwenden will, müssen im VHDL-Code auch schon berücksichtigt werden. In Abbildung 7 ist eine kleine Auswahl der zur Verfügung stehenden Pads zu sehen.

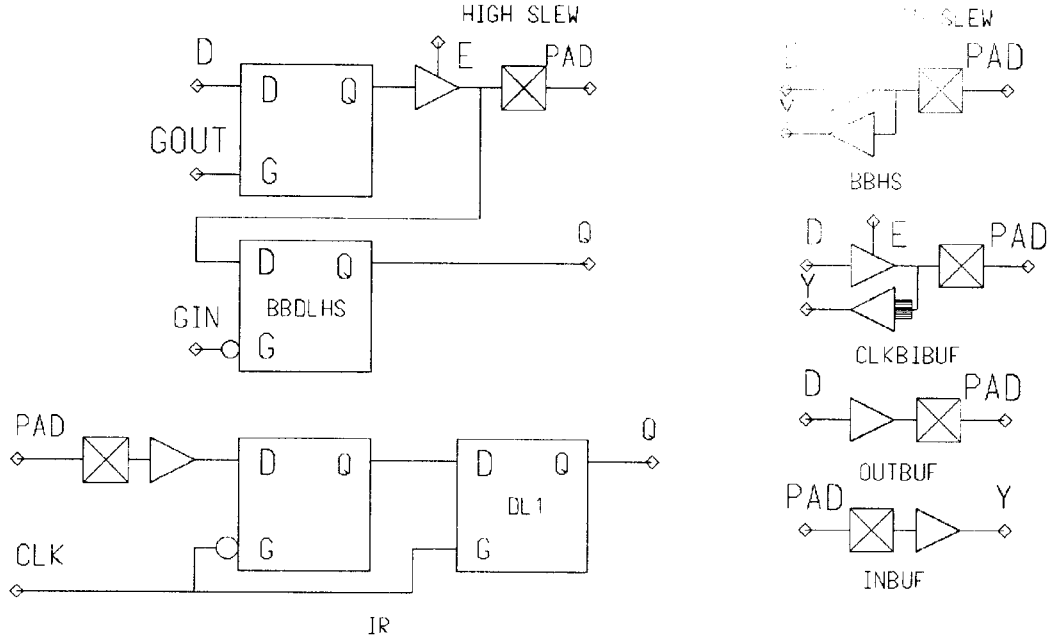


ABBILDUNG 7. Auswahl verschiedener Padtypen

3. Außerdem müssen die bidirektionalen Leitungen in eine Eingangs- und eine Ausgangsleitung zerlegt werden.

3.3.2 Simulation des VHDL-Codes

Nachdem der VHDL-Code compiliert wurde, kann er simuliert werden. Im **Design Manager** selektiert man hierzu die compilierte Datei und öffnet dann **Quicksim II**. (Abbildung 8)

Öffnen des VHDL-Codes:

Um den VHDL-Code zu öffnen, muß man in QuicksimII nur das **IKON Open Sheet** aktivieren. Danach erscheint der VHDL-Code in Quicksim II.

Auswahl der Signale:

Mit der linken Maustaste kann man nun die Signale selektieren, die man zur Untersuchung der Schaltung benötigt.

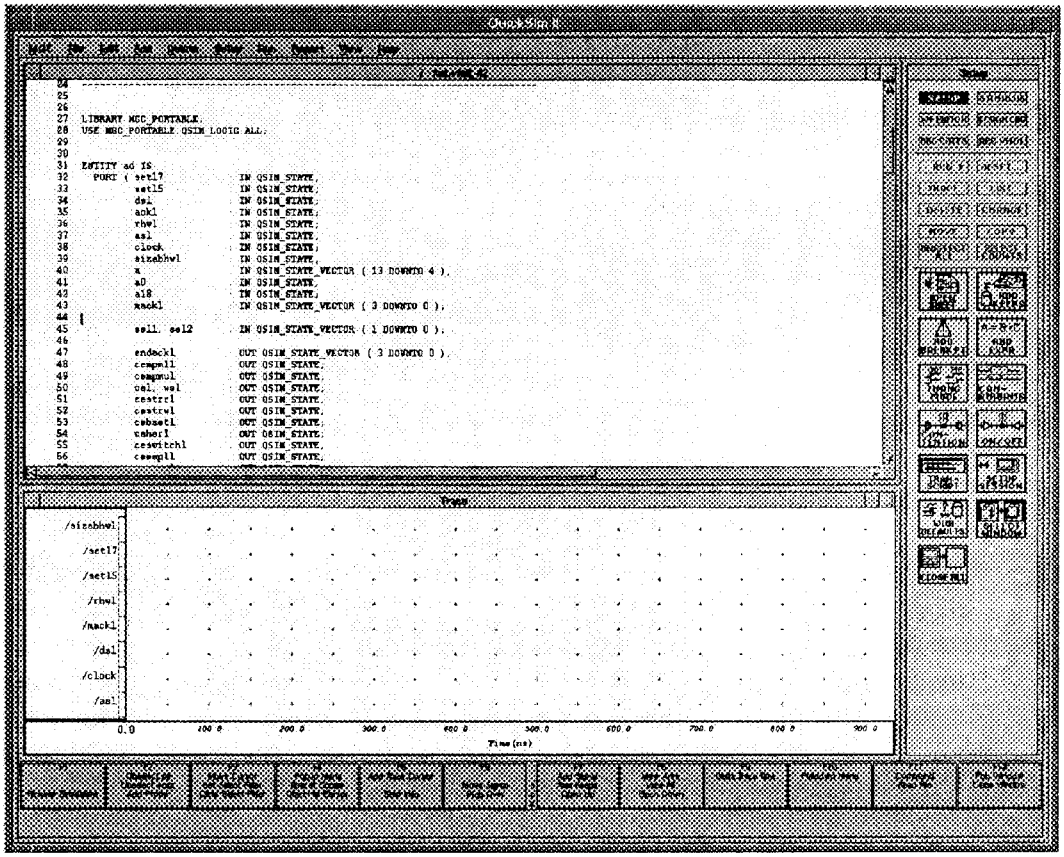


ABBILDUNG 8. Quicksim mit VHDL-Code und Tracefenster

Öffnen des Tracefensters:

Hat man alle Signale selektiert, die man zur Untersuchung braucht, so kann man durch aktivieren des **IKON Trace** das Tracefenster öffnen.

Zuweisen der Signalzustände:

Nun muß man den Eingangssignalen ihre Signalzustände zuweisen. Dazu drückt man das **IKON Stimulus** in der Setup Palette. Damit schaltet man die Palette für die Eingabe der Signalwerte um. Nun selektiert man das Signal, dem man einen Wert zuweisen will und öffnet dann durch Drücken des **IKON Add Forces** die Dialogbox (Abbildung 9). Hier kann man nun dem Signal die Signalwerte zuweisen.

Für getaktete Signale gibt es eine eigene Dialogbox. Sie kann durch Aktivieren des **IKON Add Clock** geöffnet werden. In ihr können dann die Takte definiert werden.

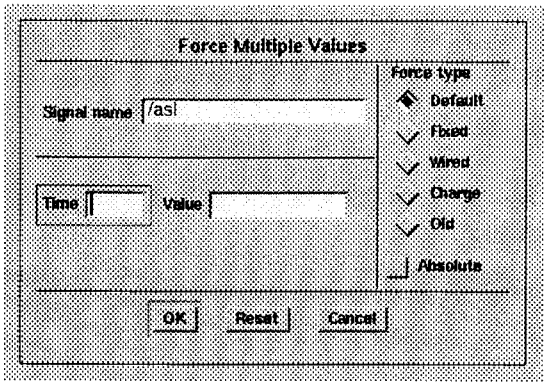


ABBILDUNG 9. Force Multiple Value Dialogbox

Simulation:

Um die Schaltung zu simulieren, muß man nur noch das **IKON Run** aktivieren und in der Dialogbox die Zeitdauer der Simulation eintragen.

Will man die Simulation mit anderen Signalwerten wiederholen, so muß man zuerst die momentane Simulation zurücksetzen, da der Simulator sonst am Ende der alten Simulation fortfährt.

Hierzu aktiviert man das **IKON Reset**. In der Dialogbox (Abbildung 10) aktiviert man nun den Schalter **State**, um den Zustand der alten Simulation zurückzusetzen. Falls man den alten Zustand der Simulation speichern will, kann man den Schalter **Save 'results'Waveform DB** aktiviert lassen. Im anderen Fall muß man ihn deaktivieren. Betätigt man nun noch den Schalter **OK**, wird die momentane Simulation zurückgesetzt.

Ändern der Forces:

Will man die Simulation mit neuen Werten wiederholen, so muß man zuerst die alten Forces löschen. Dazu selektiert man die Signale, welchen man Werte zuweisen will und aktiviert dann das **IKON Delete Forces**.

In der Dialogbox (Abbildung 11) kann man die Forces für die gesamte Zeitdauer der Simulation oder nur für einen bestimmten Zeitbereich zurücksetzen.

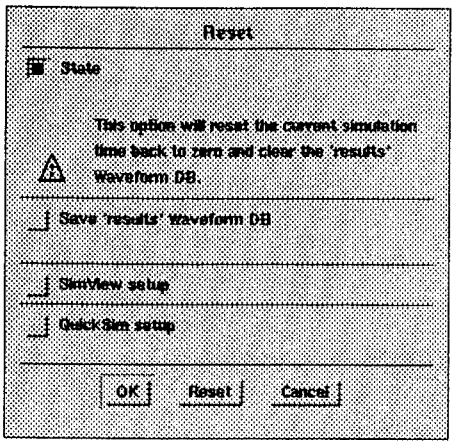


ABBILDUNG 10. Quicksim Reset

Um die Forces nur ab einem bestimmten Zeitpunkt zu löschen, trägt man diesen Zeitpunkt bei Time ein. Dabei ist zu beachten, daß die Forces nur bis zu dem Zeitpunkt geändert werden, bis zu dem man dem Signal neue Werte zuweist. Für die Zeit danach bleiben die alten Werte erhalten. Damit besteht die Möglichkeit, die Forces nur in einem kleinen Zeitfenster zu ändern.

Trägt man bei Time keinen Wert ein, so wird die gesamte Simulation zurückgesetzt.- Hat man alle Änderungen vorgenommen, so kann man die Simulation wiederholen.

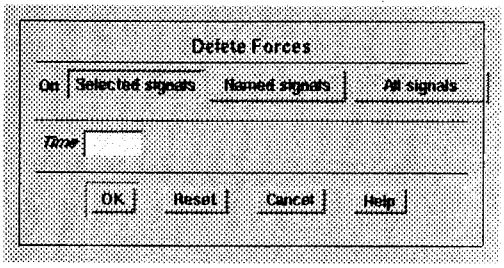


ABBILDUNG 11. Dialogbox Delete Forces

3.3.3 Synthese des VHDL-Codes

Nachdem der VHDL-Code komplett erstellt und getestet worden ist, kann er synthetisiert werden. Dazu ruft man das Synthesetool **Autologic** wie in Kapitel 6.2 beschrieben auf.

Öffnen des VHDL-Codes:

Um in Autologic den VHDL-Code zu öffnen, wählt man **File > Open > Sheet**. Daraufhin wird der VHDL-Code in **Autologic** angezeigt.

Festlegen der Zieltechnologie:

Da der VHDL-Code in einem FPGA realisiert werden soll, muß man ihn auf die Technologie des FPGA's synthetisieren. Hierzu wählt man **Setup > Set Destination Tech.** In der Dialogbox (Abbildung 12) wählt man dann die entsprechende Technologie aus.

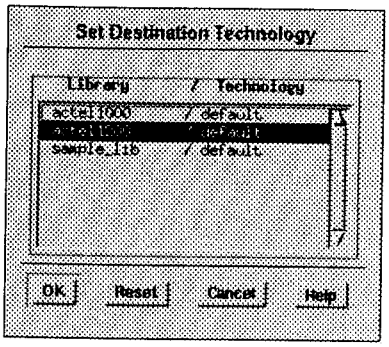


ABBILDUNG 12. Dialogbox Set Destination Technologie

Festlegen der Optimation Recipes:

Wenn man einen VHDL-Code nicht auf die Generic-Technologie, sondern auf eine andere Technologie umsetzt, so muß man die Optimierungsart festlegen. Dazu wählt man **Optimize > Set Opt Recipes**. In der Dialogbox (Abbildung 13) drückt man das **IKON Add**. Nun öffnet sich eine weitere Dialogbox (Abbildung 14), in der man die einzelnen Optimierungsarten auswählen kann. Mit dem **IKON Add** fügt man eine Optimierungsart dazu. Ist man fertig, kann man die Dialogbox mit dem **IKON Done** wieder schließen.

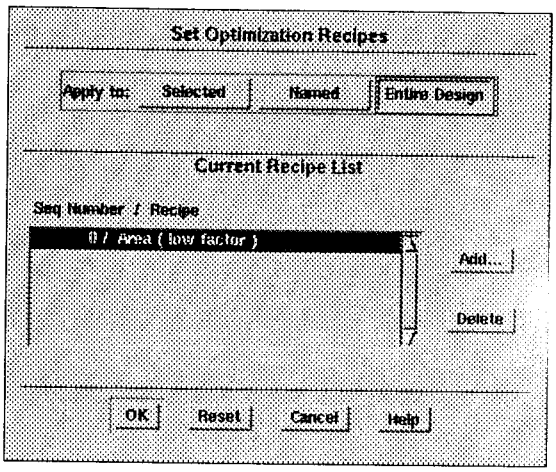


ABBILDUNG 13. Dialogbox Set Optimization Recipes

Synthetisieren:

Sind alle Einstellungen vorgenommen, kann der VHDL-Code synthetisiert werden. Dazu wählt man **Synthesize > Synthesize Design**.

In der Dialogbox (Abbildung 15) muß man nun noch den Schalter **Schematic** drücken, damit ein Schaltplan erstellt wird. Wenn man die Schaltung in einem anderen Verzeichnis ablegen will, muß man bei **Specify Destination** den Schalter **Yes** drücken. Anschließend kann man bei **Destination Component Path** den Pfad des neuen Verzeichnisses angeben. Betätigt man nun noch den Schalter **OK**, so wird der VHDL-Code synthetisiert.

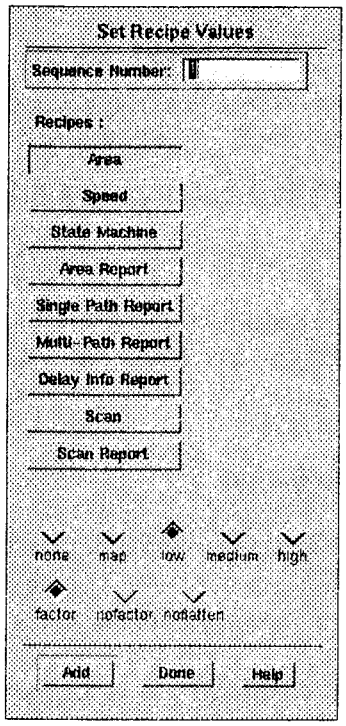


ABBILDUNG 14. Dialogbox Set Recipes Value

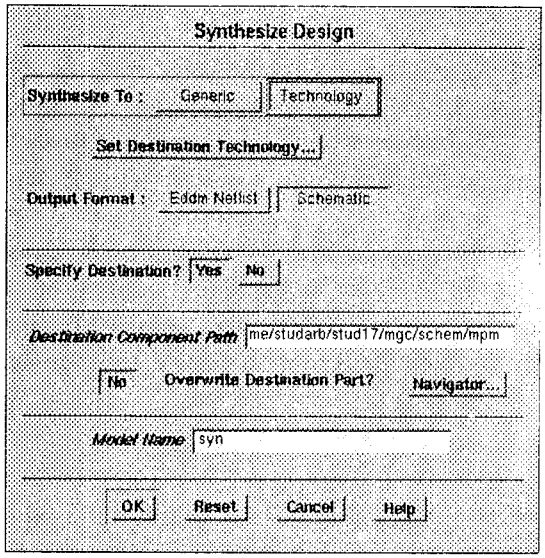


ABBILDUNG 15. Dialogbox Synthesize Design

3.3.4 Erstellen des Designviewpoints

Bevor die synthetisierte Schaltung mit Quicksim II simuliert werden kann, muß zuerst ein Designviewpoint erzeugt werden. In diesem Designviewpoint wird dem Parameter **als_technology** ein Wert zugewiesen, der die Technologie bezeichnet, die simuliert werden soll. Ist es z.B. ein Design mit ACT2-Bausteinen, so wird dem Parameter der Wert ACT2 zugewiesen. Der Designviewpoint wird wie folgt erstellt:

- Öffnen eines hptermfensters
- Wechseln in das Arbeitsverzeichnis
- Im Arbeitsverzeichnis setzen der Umgebungsvariable MGC_WD auf das Arbeitsverzeichnis mit *setenv MGC_WD 'pwd'*
- Eingeben des Befehls zum Erstellen des Designviewpoints:

  presimvpt fam:<actn> <design_name>

Den Platzhalter *<actn>* kann man durch ACT1, ACT2 oder ACT3 ersetzen, anstelle von *<design_name>* kommt der Name des Designs, im allgemeinen der Name des Arbeitsverzeichnisses.

3.3.5 Funktionale Simulation

Bei der funktionalen Simulation wird die synthetisierte Schaltung mit UNIT-Delay simuliert, d.h jedem Baustein wird eine Verzögerungszeit von 0.1ns zugewiesen. Eine andere Simulation ist zu diesem Zeitpunkt noch nicht möglich.

Der Ablauf der Simulation erfolgt wie in Kapitel 3.3.2 schon beschrieben.

3.3.6 Einbau der Pads

Bevor die Schaltung optimiert werden kann, müssen die I/O-Pads eingefügt werden. Hat man ein hierarchisch aufgebautes Design, so müssen die Pads auf der obersten Hierarchieebene plaziert werden. In den unteren Ebenen dürfen keine Pads vorhanden sein, da dies sonst zu Fehlern beim Erstellen der ACTEL-Netzliste führt.

Die ACTEL-Bibliothek weist eine ganze Reihe verschiedener Pads auf. Besonders zu beachten sind dabei die Pads für getaktete Leitungen. Jede getaktete Leitung sollte über einen Clockpad angesteuert werden. Damit wird gewährleistet, daß alle von diesem Signal angesteuerten Baugruppen gleichzeitig angesteuert werden.

3.3.7 Simulation mit bidirektionalen Pads

Will man seine Schaltung mit den Pads simulieren und enthält die Schaltung bidirektionale Pads, so muß man in Quicksim II beim Zuweisen der Signalzustände für die bidirektionalen Pads den **Force Typ *wired*** wählen, da die bidirektionalen Pads ansonsten nicht korrekt arbeiten.

Dies kann man in der Force Multiple Value-Dialogbox machen, indem man bei den bidirektionalen Pads den Schalter *Wired* setzt. (Abbildung 16)

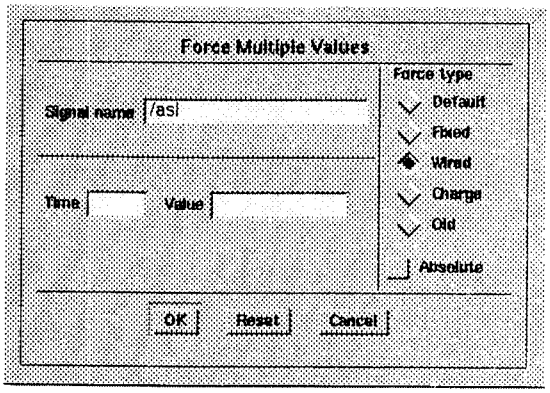


ABBILDUNG 16. Einstellung für bidirektionale Pads

3.3.8 Optimieren

Als letzes wird die Schaltung in Autologic auf Fläche optimiert. Eine Optimierung auf Geschwindigkeit war bis zum Ende der Diplomarbeit nicht möglich, daher wird nur eine Optimierung auf Fläche durchgeführt.

Unterschiede zwischen der Libraryversion 2.2 und 2.3

Bei einem ACTEL-Design ist es wichtig, die Belastung der Netze, das Fanout, möglichst gering zu halten, damit beim Plazieren und Verdrahten der Schaltung keine unnötig großes Laufzeiten entstehen. Eine Leitung darf maximal ein Fanout von 10 besitzen, das heißt, ein Ausgang darf höchstens 10 Eingänge treiben. Will man ein schnelles Design entwerfen, so ist das Fanout auf 5, bei sehr schnellen Designs sogar nur auf 3 zu begrenzen.

Mit der Libraryversion 2.2 von Mentor war es nicht möglich,das Fanout der Netze einzustellen. Dies führte in der Diplomarbeit dazu, daß bis zu 30 Netze ein zu großes Fanout aufwiesen. Eine Korrektur von Hand war aufgrund der Größe des Designs sehr aufwendig.

Mit der Libraryversion 2.3 ist es nun möglich, in der Dialogbox Set Technologie Enviroment das Fanout sowohl für alle Netze sowie nur für ein paar spezielle einzustellen. Die default-Einstellung für das globale Fanout ist 6. Bei der Diplomarbeit wurde ein Fanout von 5 gewählt. Die Ergebnisse, die damit erzielt wurden, sind sehr zufriedenstellend.

Bei der Optimierung mit der Version 2.3 wird sehr viel Logik durch Flipflops und Latches ersetzt. Dadurch entstehen sehr viele Taktsignale, die nicht von Takttreibern getrieben werden. Werden mehr als 25 normale Netze als Taktleitungen verwendet, so führt dies im Designer zu einer Warnung, beeinflusst im Allgemeinen die Verdrahtbarkeit des FPGAs nicht.

Festlegen der Optimation Constraints:

Ein ACTEL-Design darf nur aus einer Ebene bestehen, da sonst im Designer gravierende Fehler auftreten. Nun müssen die Bedingungen festgelegt werden, mit denen die

Schaltung optimiert werden soll. Um die Bedingungen einzustellen, ruft man **Optimize > Set Opt Constaints > Set Hierarchy Control** auf.

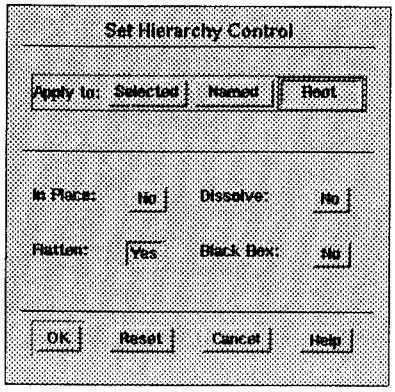


ABBILDUNG 17. Dialogbox Set Hierarchy Control

Um die Schaltung auf einer Ebene zu optimieren, selektiert man die Schalter **Root** und **Flatten:yes**. Dann wird die gesamte Schaltung auf einer Ebene optimiert.

Festlegen der Optimation Recipes:

Die Optimation Recipes werden wie in Kapitel 3.3.3 schon beschrieben, gesetzt.

Optimieren der Schaltung:

Um die Schaltung zu Optimieren, ruft man nun **Optimize > Optimize Design** auf. In der Dialogbox (Abbildung 18) kann man nun die gewünschten Formate auswählen und dann die Schaltung optimieren lassen.

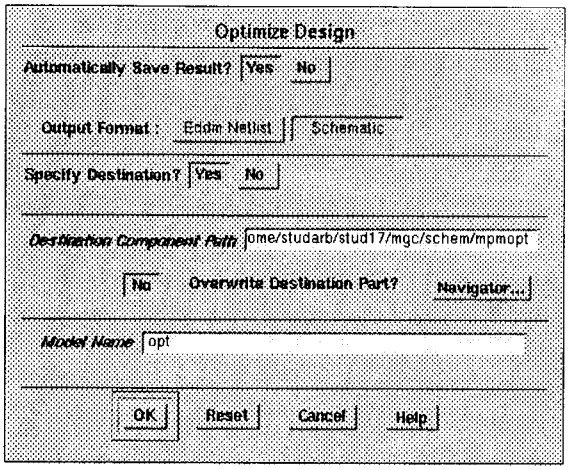


ABBILDUNG 18. Dialogbox Optimize Design

3.4 Erstellen der ACTEL-Netzliste

Nachdem die Schaltung erstellt und funktional getestet ist, kann nun über die definierte Schnittstelle die ACTEL-Netzliste erzeugt werden. Der Ablauf ist wie folgt:

- Öffnen eines Hptermfensters
- Wechseln in das Designverzeichnis
- Eingabe des Befehls zum Erstellen der Netzliste:

  mgc2adl fam:<actn> <design_name>

Der Platzhalter *<actn>* wird durch ACT1, ACT2 oder ACT3 ersetzt, der Platzhalter *<design_name>* steht für den Namen des Designverzeichnisses.

3.5 Die ACTION LOGIC SYSTEM-Umgebung

Die weitere Entwicklung der Schaltung erfolgt nun in der ALS-Umgebung. Alle Tätigkeiten , die zur Fertigstellung des FPGA's noch nötig sind, erfolgen in einem Tool, dem **Designer**.

Für die Realisierung des FPGA's stellt der Designer folgende Funktionen zur Verfügung:

- Validate: Überprüfen des Designs auf elektrische Fehler und auf Fanout-Belastung
- Place: Plazieren der einzelnen Module
- Route: Verdrahten der Module
- Extract: Erstellen der Daten für eine Echtzeitsimulation
- Fuser: Erstellen der Daten für das Programmiergerät

Außerdem bietet der Designer die Möglichkeit, über einen Graphic Editor das physikalische Layout des FPGA's zu betrachten und hier interaktiv z.B. die Anordnung der Pin's festzulegen.

3.6 Der Designablauf

Abbildung19 zeigt den vollständigen Ablauf zur Fertigstellung eines FPGA's in der ALS-Umgebung. Vorausetzung für den Ablauf ist eine aus Fläche und Geschwindigkeit optimierte Netzliste.

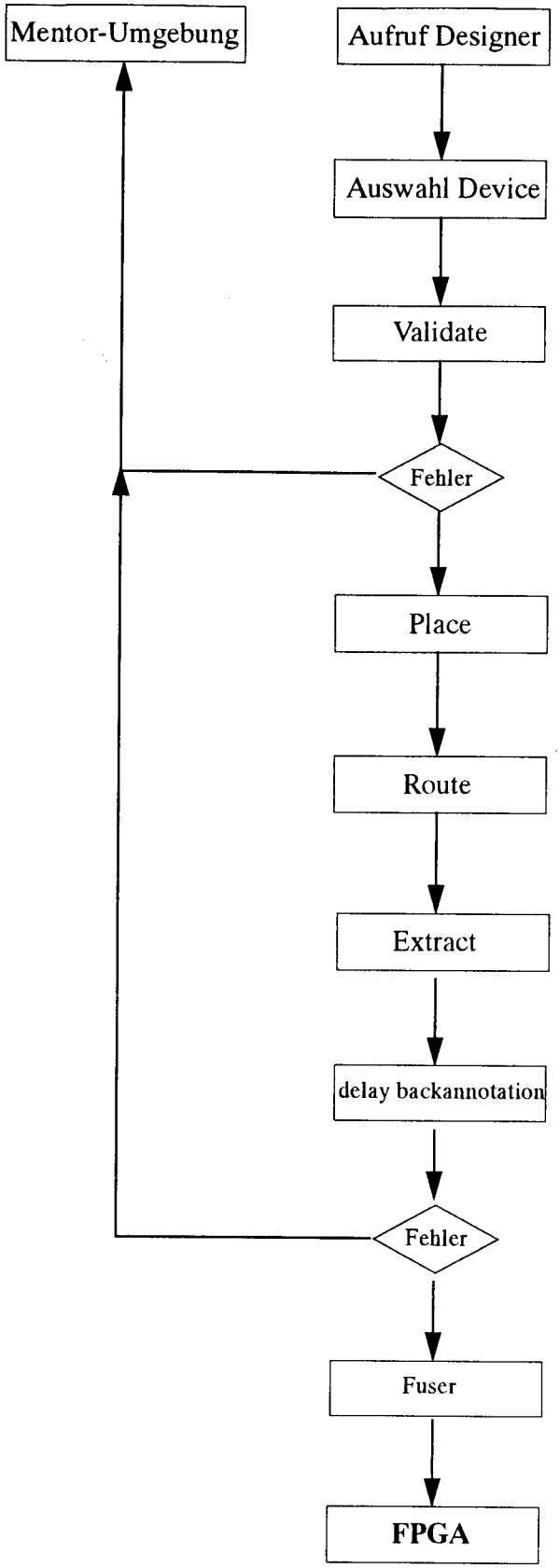


ABBILDUNG 19. Designablauf im Designer

3.6.1 Aufruf des Designers

Arbeitet man an der Maschine, auf der die ACTEL-Software installiert ist, im CAE-Labor des Fachbereichs Elektronik ist das die Maschine **cae2**, kann man den Designer wie folgt aufrufen:

- Öffnen eines hptermfensters
- Wechseln in das Designverzeichnis
- Eingabe des Befehls *designer*

Anschließend öffnet sich der Designer.(Abbildung 20)

Arbeitet man an einer anderen Maschine, ist der Aufruf folgendermaßen:

Öffnen eines hptermfensters

Einloggen auf der Maschine cae2:

xhost cae2
telnet cae2
login:
password:
setenv DIPLAY caeX:0
csh
designer

Nun kann man auch auf dieser Maschine mit dem Designer arbeiten.

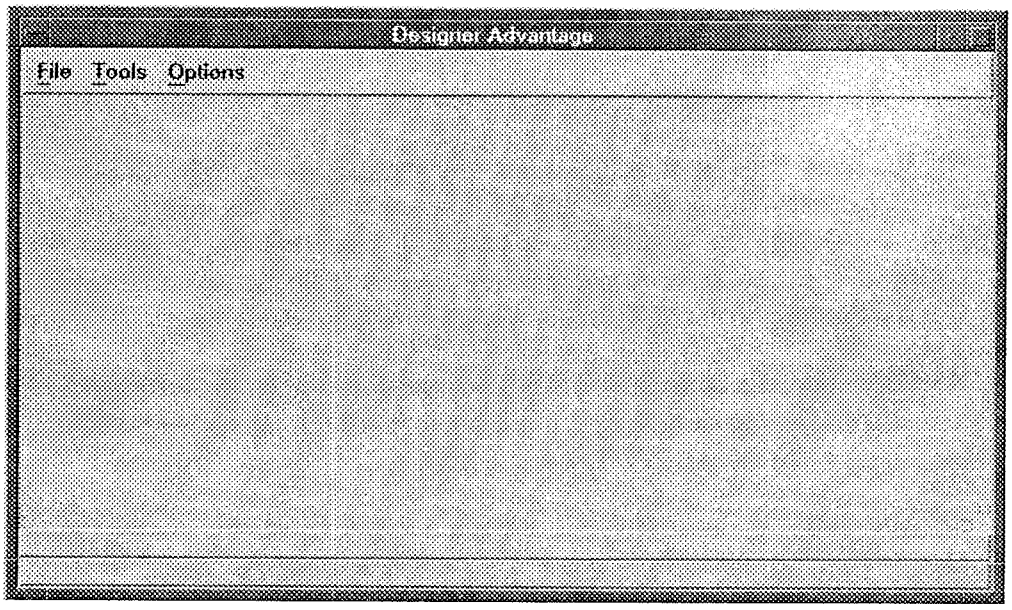


ABBILDUNG 20. Der Designer

3.6.2 Öffnen des Designs

Um das Design zu öffnen, wählt man in der Menueleiste des Designers **File > Open**. In der Dialogbox (Abbildung 21) kann man nun das Design auswählen. Falls man mehrere Versionen des Design gespeichert hat, so kann man diese über den Schalter Version auswählen.

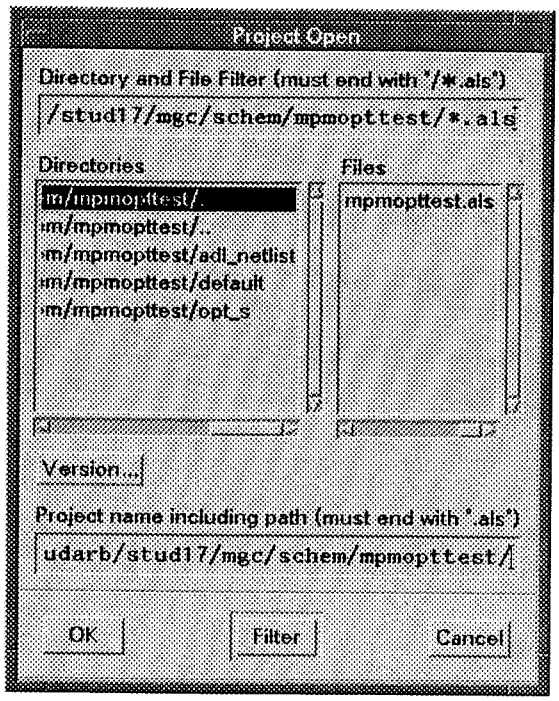


ABBILDUNG 21. Dialogbox Project Open

3.6.3 Auswahl des FPGA's

Als nächstes wird man aufgefordert, die Familie, die Größe und die Bauform des FPGA's auszuwählen. (Abbildung 22) Diese Aufforderung erfolgt automatisch, wenn man das Design das erste Mal aufruft. Will man später eine andere Bauform oder Größe des FPGA's, so kann man die Device Selection-Dialogbox mit **Option > Device/Package** erneut öffnen und die gewünschten Änderungen vornehmen.

Nachdem man die Auswahl bestätigt hat, wird man noch darauf hingewiesen, daß die aktuelle Plazierung des Designs mit den neuen Änderungen veraltet ist und der Validator erneut aufgerufen wird. (Abbildung 23)

Ebenso erfolgt noch der Hinweis, daß der Validator gestartet werden muß, bevor das Design geöffnet wird. Hier besteht dann nochmal die Möglichkeit, Änderungen an der FPGA-Auswahl zu treffen.(Abbildung 24)

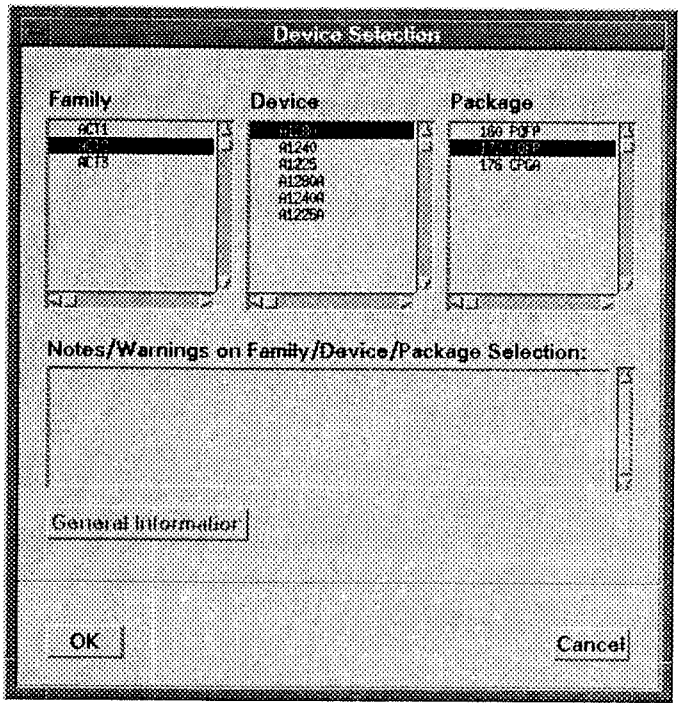


ABBILDUNG 22. Dialogbox Device Selection

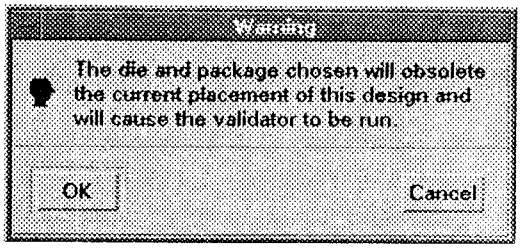


ABBILDUNG 23. Warnung bei der Auswahl des FPGA's

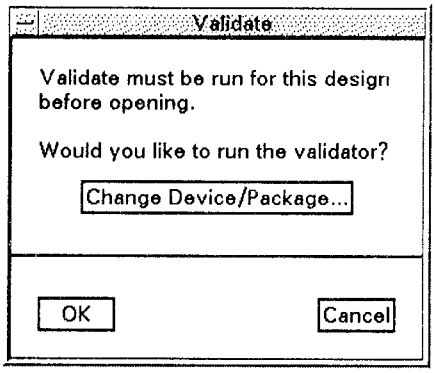


ABBILDUNG 24. Dialogbox Validate

3.6.4 Der Validator

Der Validator überprüft das Design auf Übertretungen elektrischer- oder Design-Regeln und Kapazitätsüberschreitungen (Fanout) der Netze anhand des ausgewählten FPGA's.

Zu den elektrischen, bzw. Designregeln zählen:

- kurzgeschlossene Ausgänge
- nicht angeschlossene Eingänge
- offene Pins
- selektierte ACTEL-Familie
- Größe des FPGA's

Bei diesen Fehlern bricht der Validator mit einer Fehlermeldung ab. Die Fehler müssen dann in der Mentor Graphics-Umgebung korriegiert werden, bevor das Design erneut geladen werden kann.

Bei dem Fanout erfolgt nur dann eine Fehlermeldung, wenn ein Ausgang ein Fanout von 24 hat, d.h wenn er maximal 24 Eingänge treibt.

Eine Warnung erfolgt, wenn das Fanout eines Ausgangs den Wert 10 überschreitet. Sind es nur wenige Netze mit einem Fanout größer als 10, so kann dies toleriert werden. Man muß dabei aber bedenken, daß dadurch die Schaltung langsamer wird. Will man eine schnelle Schaltung, so sollte jedes Netz kein größeres Fanout als 5 haben.

Neben den Fehlermeldungen und den Warnungen stellt der Validator noch statistische Angaben über die Verdrahtbarkeit, die Anzahl der verwendeten Module, den Flächenverbrauch und die durchschnittliche Belastung der Netze zur Verfügung. (siehe Abbildung 20)

3.6.5 Place

Hat das Design den Validator fehlerfrei durchlaufen, so kann das FPGA plaziert und verdrahtet werden.

Für das Plazieren des FPGA's stehen mehrere Möglichkeiten zur Verfügung. Man kann die Plazierung vollautomatisch oder interaktiv durchführen. Ebenso hat man die Möglichkeit, zuerst interaktiv, z.B. die Pads zu plazieren und danach die Module automatisch plazieren zu lassen.

Automatische Plazierung:

Um das FPGA automatisch zu plazieren, wählt man in der Menueleiste **Tools > Automatic Layout Tool**. In der Dialogbox **Automatic Layout Tool** (Abbildung 25) wählt man nur den Schalter **Place**. Drückt man nun noch den Schalter **OK**, so wird das FGPA automatisch plaziert.

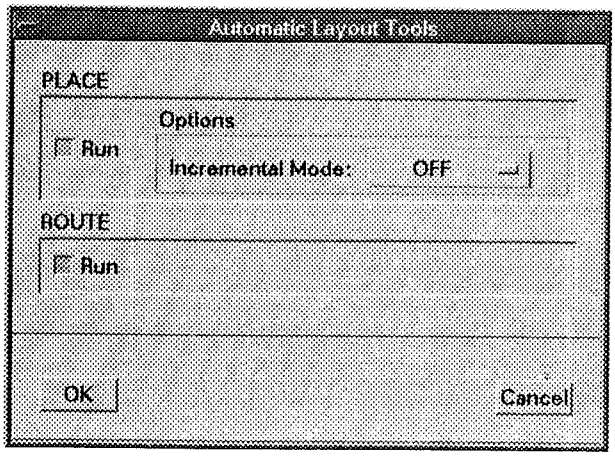


ABBILDUNG 25. Automatic Layout Tool

Das automatische Place-Tool plaziert die Pad's und die Module zu 100% auf dem FPGA. Die automatische Plazierung gewährleistet die Einhaltung der maximalen Geschwindigkeit. Durch eine interaktive Plazierung der Module kann diese aber noch erhöht werden.

Die Plazierung der Pads und der Module, ebenso wie die interaktive Plazierung erfolgt im Graphic Editor.

Der Graphic Editor

Der Graphic Editor wird mit **Tools > Graphic Editor** aufgerufen. Nach kurzer Zeit öffnet sich dann der Graphic Editor. (Abbildung 26)

Eine detaillierte Beschreibung des Graphic Editors kann der entsprechenden Literatur entnommen werden [5], da dies den Rahmen dieser Arbeit sprengen und nicht zum Inhalt dazu gehört. Hier wird nur auf die Funktionen eingegangen, die für den Ablauf wichtig sind.

Interaktives Plazieren:

Beim interaktiven Plazieren von Elementen hat man zwei Möglichkeiten. Zum einen kann man einzelne Elemente zuerst plazieren, wenn sie an einen festen Ort im FPGA müssen. Dies bietet sich zum Beispiel bei den Pads an, da die Busse zumeist im FPGA besser verdrahtet werden können als auf einer Platine.

Die andere Möglichkeit ist, auf einem plazierten FPGA die einzlnen Module zu verschieben, um damit z.B. ein besseres Verhalten des FPGA's zu erhalten.

- **Plazieren von Elementen aus der Object List**

Im Graphic Editor hat man die Möglichkeit, noch nicht plazierte Elemente aus der Object List zu plazieren.

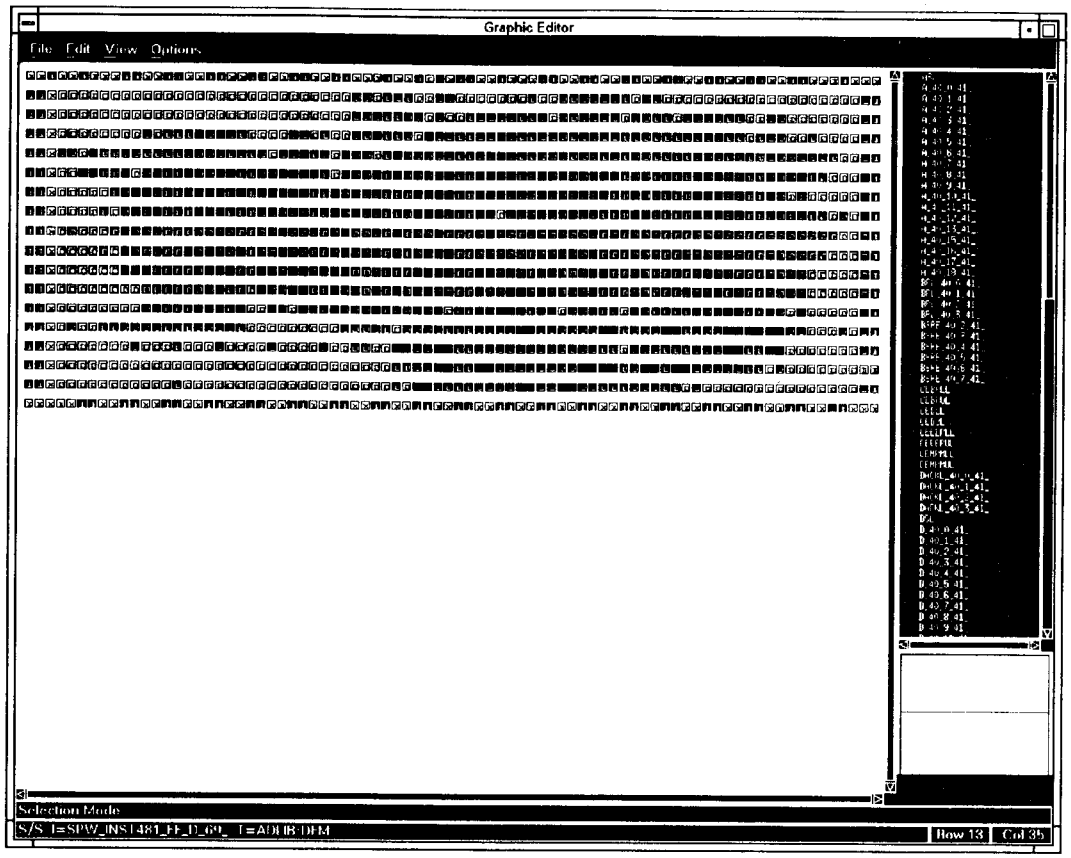


ABBILDUNG 26. Graphic Editor

Um die zu plazierenden Elemente zu selektieren, kann man mit **Options > Configure Object List** in der Dialogbox (Abbildung 27) die Modultypen selektieren, die man plazieren möchte.

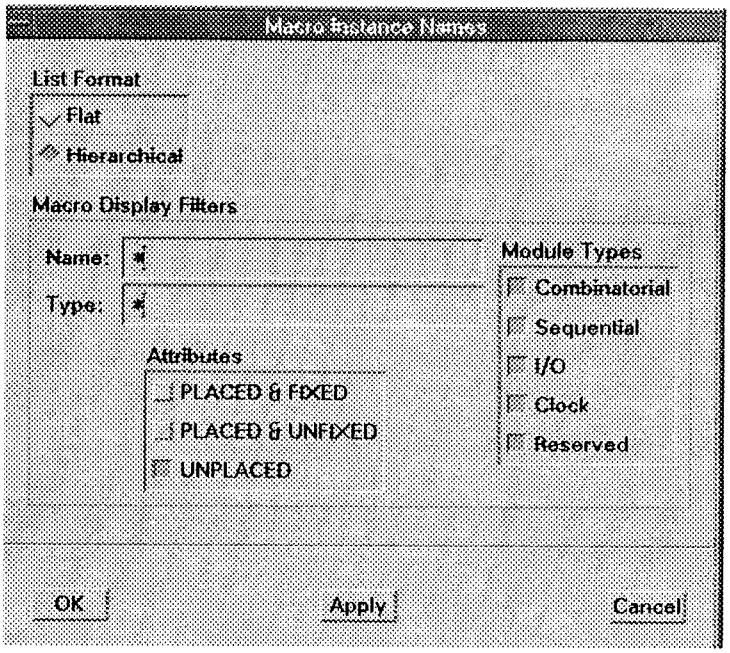


ABBILDUNG 27. Dialogbox Configure Object List

Ausserdem hat man die Möglichkeit, sich z.B. die Pins statt mit der Instancebezeichnung mit dem Netznamen anzeigen zu lassen. Dazu wählt man **Options > Chance Object List > I/O Net Names**. In der Dialogbox hat man nun die Möglichkeit, sich die plazierten oder die nichtplazierten Pins anzeigen zu lassen. Ebenso die schon plazierten und fixierten Pins.

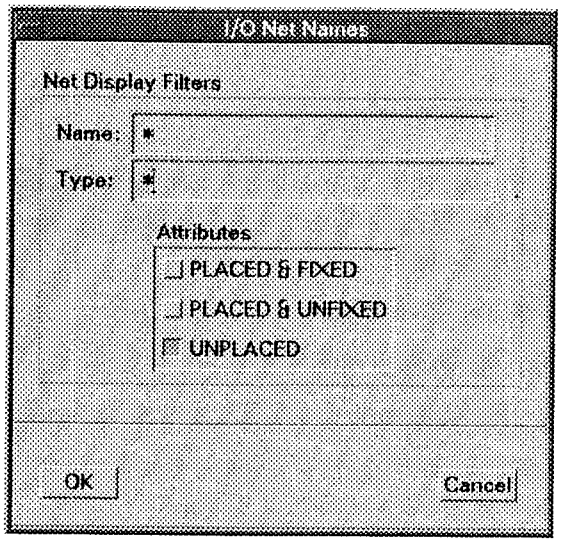


ABBILDUNG 28. Dialogbox I/O Net Name

Um nun ein Modul zu plazieren, geht man wie folgt vor:

- selektieren des Moduls in der Object List
- In der Menueleiste **Edit > Place** wählen, in der Message Bar erscheint die Meldung: *Click on destination module for macro <macro_name>*
- Plazieren des Moduls, indam man mit der linken Maustaste auf das Zielmacro drückt
- **Plazieren und Bewegen von Elementen im Graphic Display Area**

Neben dem Plazieren von Elementen aus der Object List hat man noch die Möglichkeit, schon plazierte Macros an andere Stellen zu bewegen. Dazu geht man folgendermassen vor:

- Mit der linken Maustaste das Macro selektieren
- Mit gedrückter Maustaste das Macro an seine neue Position plazieren

3.6.6 Route

Nachdem alle Macros plaziert sind, kann das FPGA verdrahtet werden. Dies erfolgt automatisch mit dem Layouttool.

Dazu wählt man in der Menueleiste **Tools > Automatic Layout Tool**. Nun aktiviert man Schalter **Run** bei **Route** und drückt dann den Schalter **OK**. Nun wird das FPGA automatisch verdrahtet.

3.6.7 Extract

Nun können die Daten für die Echtzeitsimulation berechnet werden. dazu wählt man **Tools > Export Timing Data**. Danach werden die Daten für die Echtzeitsimulation des plazierten und verdrahteten FPGA's berechnet. Dabei werden die Verzögerungszeiten, die durch die Plazierung und die Verdrahtung des FPGA's entstehen, mitberücksichtigt.

3.6.8 Delay backannotation

Bevor die Schaltung mit Echtzeitdaten in **Quicksim II** simuliert werden kann, muß das backannotation-File für **Quicksim II** erstellt werden. Dies erfolgt wieder über eine vordefinierte Schnittstelle:

- Öffnen eines hpterm-Fensters
- Wechseln in das Designverzeichnis
- Eingabe des Befehls zum Erstellen eines backannotation-Files

del2mgc fam:<actn> <temperature range> <voltage range> <speed grade> <design_name>

Für die ACT2-Familie kann man die Platzhalter durch folgende Werte ersetzten:

<temperature range> : ind, mil, com

<voltage range> : ind mil, com

<speed grade> : std, -1

Die Bedeutung dieser Werte, sowie die Werte für die anderen ACTEL-Familien kann der entsprechenden Literatur entnommen werden.

3.6.9 Echtzeitsimulation

Für die Echtzeitsimulation für den Wert *com* (commerziell) stellt ACTEL die Werte aus Tabelle 1 zur Verfügung.

min (commercial)	**typ (commercial)**	**max (commercial)**
0 Grad Celsius	25 Grad Celsius	70 Grad Celsius
5.25 Volt	5.00 Volt	4.75 Volt
best-case process	typical-case process	worst-case process

TABELLE 1. Bedingungen für die Simulation mit kommerziellen Werten

Um diese Werte in Quicksim II einzustellen, wählt man in Quicksim II **Setup > Kernel > Analysis**. In der Dialogbox (Abbildung 29) kann man nun die verschiedenen Einstellungen vornehmen.

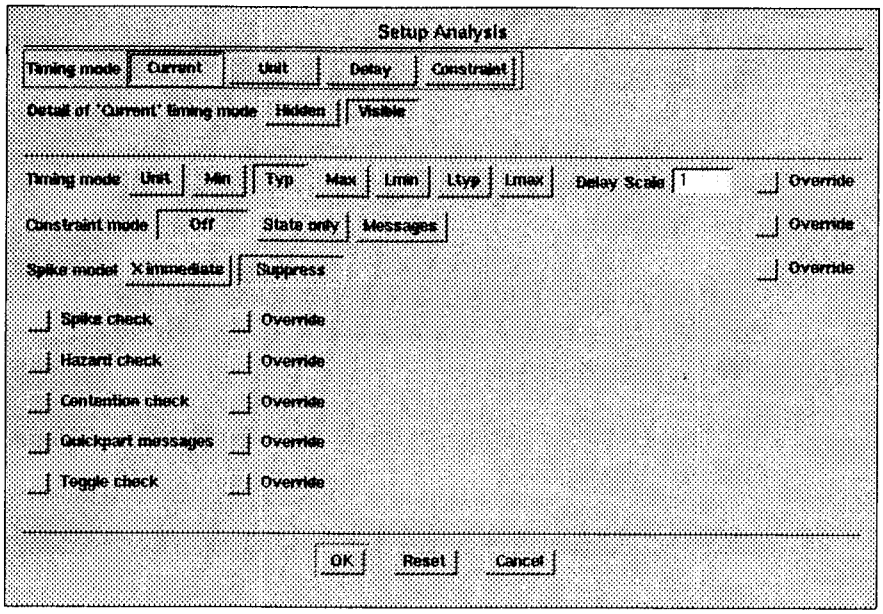


ABBILDUNG 29. Dialogbox Setup Analysis

3.6.10 Fuser

Entspricht das FPGA auch in der Echtzeitsimulation den Anforderungen, so können die Daten für das Programmieren des FPGA's erstellt werden. Dazu wählt man im Designer **Tools > Fuser**. In der Dialogbox (Abbildung 30) muß man nun noch die Signatur des FPGA's eintragen. Damit kann man die einzelnen FPGA's später auseinanderhalten, wenn man sie mit dem ACTION PROBE DIAGNOSTIC TOOL der Firma ACTEL untersucht.

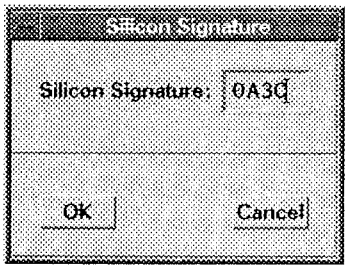


ABBILDUNG 30. Dialogbox Silicone Signature

Die Signatur muß in hexadezimaler Weise eingegeben werden.

3.6.11 Programmierung

Eine Programmierung des FPGA's konnte aus zeitlichen Gründen in dieser Diplomarbeit nicht mehr erfolgen. Dies ist Gegenstand einer Nachfolgearbeit.

Eine Beschreibung über die weitere Vorgehensweise zum Programmieren eines FPGA's erfolgt daher hier nicht.

4.0 Literaturverzeichnis

[1] Phoenix Contact GmbH & Co Entwicklungsdokumentation des MPM .

[2] Uwe Feist : *''Entwurf und Realisierung eines Multi-Ported-Memory Ansteuerchips mit Hilfe der Hardwarebeschreibungssprache VHDL''*, Diplomarbeit

[3] Andreas Hölz : *''Entwurf und Realisierung eines Multi-Port-Memory Ansteuerchips auf HP-Workstations mit dem Mentor-Graphics-Softwarepaket''*, Diplomarbeit

[4] ACTEL Designer Series for the X Window System HP 700 Mentor Graphics Enviroment

[5] ACTEL Designer Advantage for X-Windows User's Guide

[6] ACTEL FPGA Data Book and Design Guide

[7] ACTEL ACT Family Field Programmable Gate Array DATABOOK

[8] ACTEL Macro Library Guide

[9] Mentor Graphics VHDL Refercence Manual

[10] Mentor Graphics AutoLogic VHDL Synthese Guide

[11] Mentor Graphics AutoLogic Optimization Guide

MPC Workshop SS 1994

Digitaler Ton- und Rauschgenerator

B. Vettermann, G. U. Paul

Hintergrund

* Zur Maskierung von Phantomgeräuschen (Tinnitus) wurde eine analoge Schaltung in Dickschicht-Hybrid-Technik entwickelt: "Hinter dem Ohr Gerät".

* Als Alternative wird ein fernbedienbares "Im Ohr Gerät" gewünscht (Schlafkomfort).

Aufgabenstellung

* Möglichst Ein-Chip-Lösung (Platz)

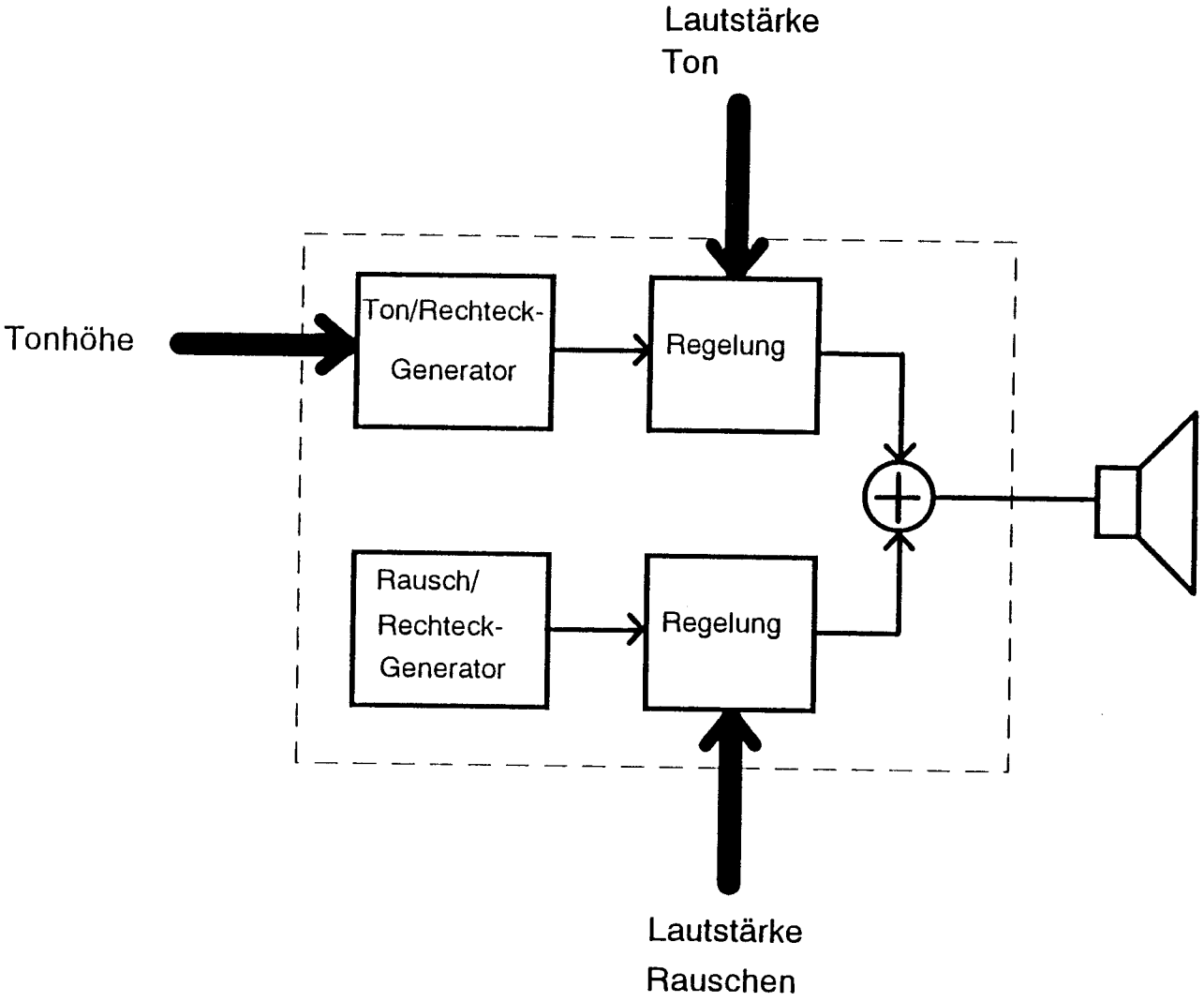
* Minimale Stromaufnahme (Batterie)

* Tonfrequenzen von 200Hz bis 8000Hz

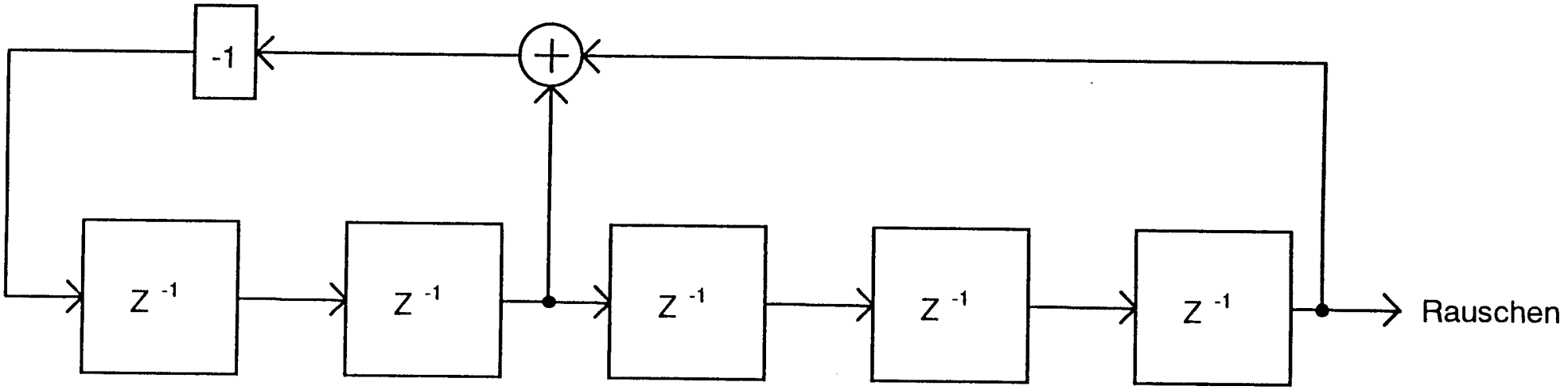
* Einstellbare Lautstärke

* Zusätzliches Hintergrundrauschen

Tinnitus- Ton/Rausch-Generator



Rausch/Rechteck-Generator



Spektrum des Rauschgenerators

Bei entsprechender Dimensionierung ist nahezu **Weißes Rauschen** möglich !

Höchste Frequenz:

Taktfrequenz des Schieberegisters
+ Oberwellen der Rechtecksignale

Niedrigste Frequenz:

Taktfrequenz/2^n-1
n: Anzahl der Register

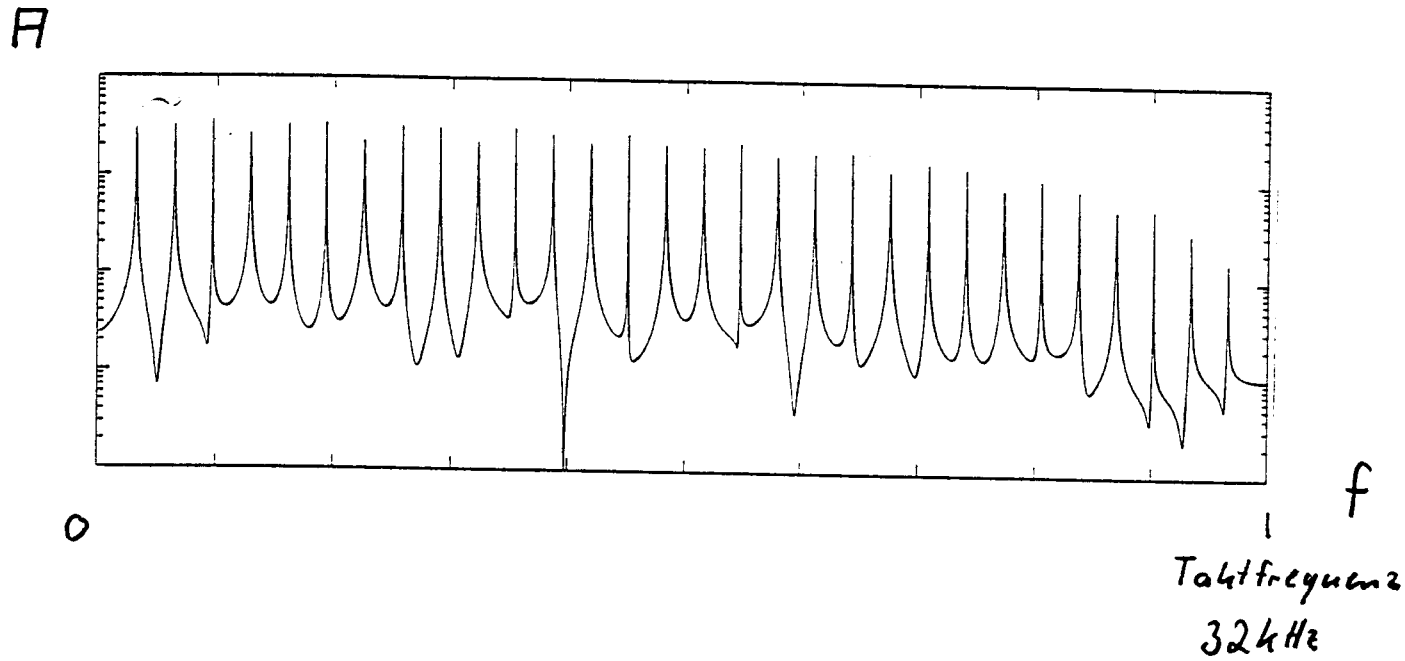
entspricht der Wiederholfrequenz
des Generators

Es sind 2^n-1 Frequenzen in gleichmäßigen Abständen enthalten.

Beispiel: 32kHz Takt, 5 Register
Rückkopplung bei : 2, 5

Höchste Frequenz: 32kHz
Niedrigste Frequenz: 32kHz/31 etwa 1kHz

Hier ist kein Rauschen zu hören, sondern die 1kHz Wiederholfrequenz.

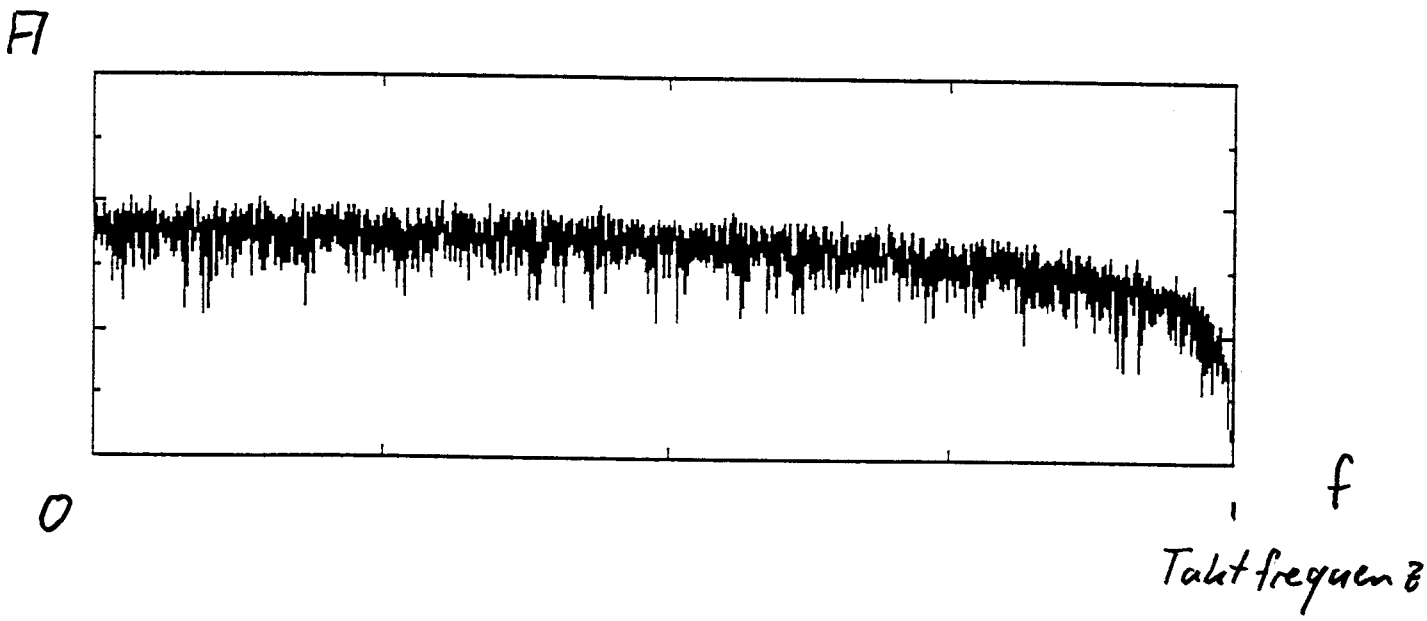


Besser:

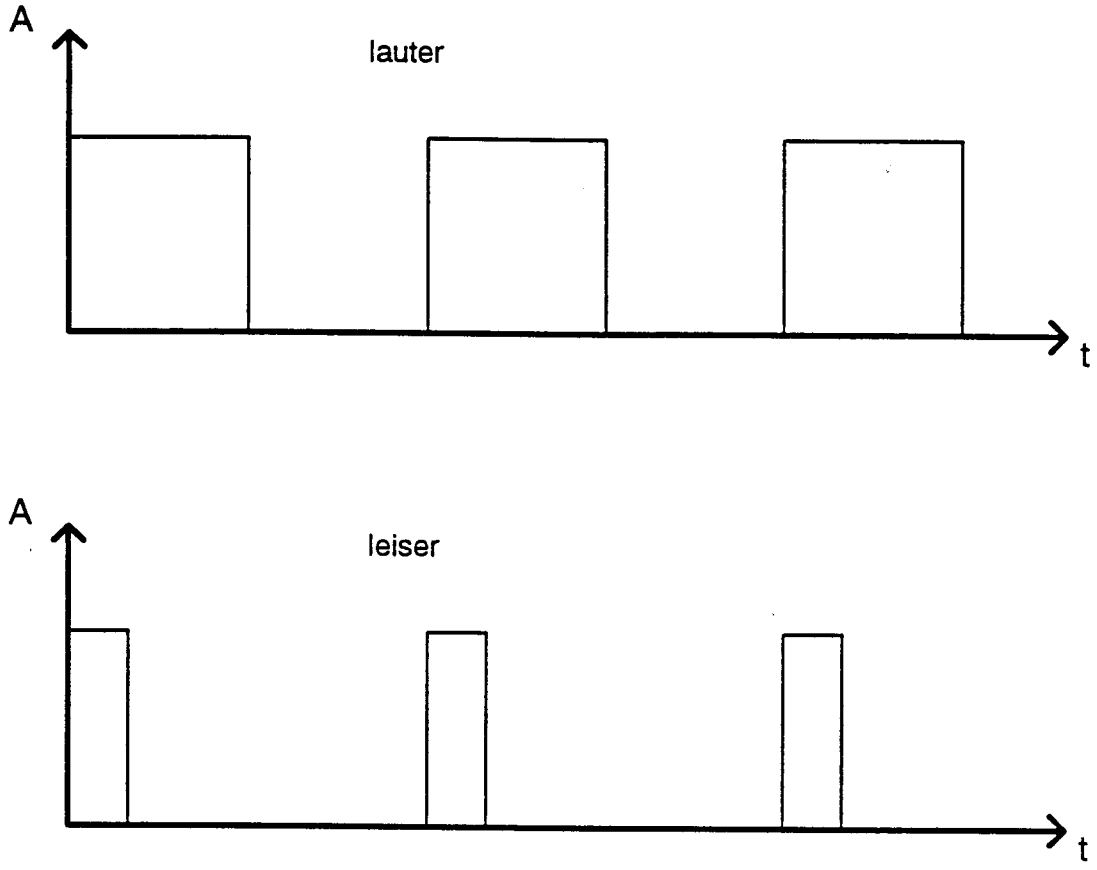
32kHz Takt, 16 Register
Rückkopplung bei : 1, 3, 12, 16

Höchste Frequenz: 32kHz
Niedrigste Frequenz: 32kHz/2^16-1 etwa 0.5Hz

Die Wiederholfrequenz liegt bei 0.5 Hz , somit unterhalb der Hörgrenze.
Im Hörbereich wird fast Weißes Rauschen erzeugt.

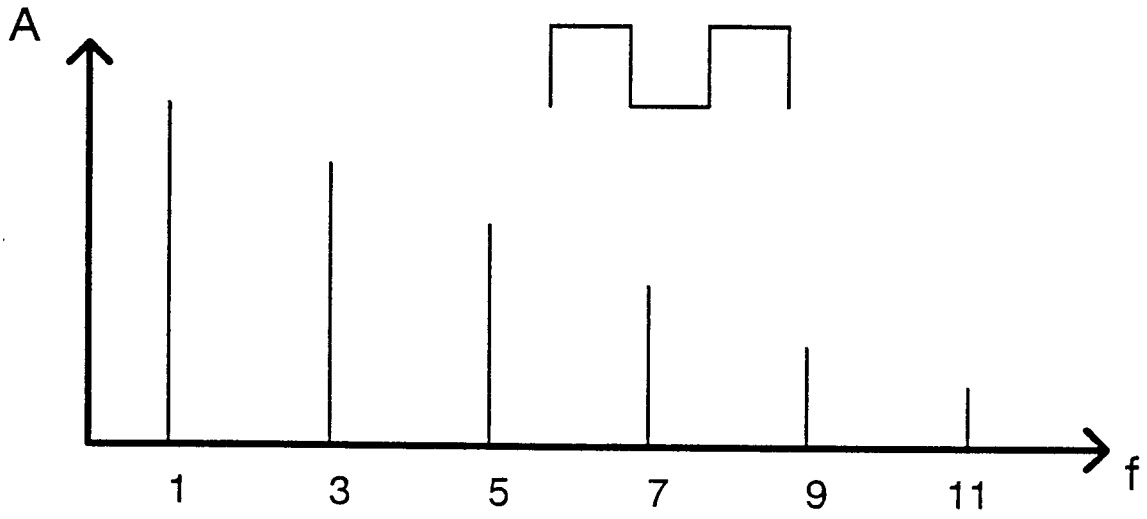


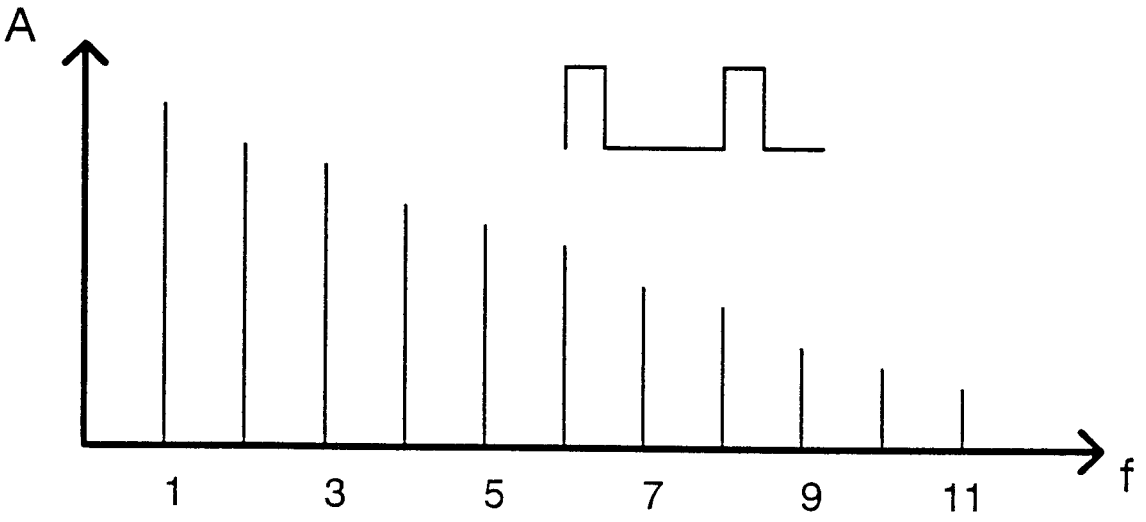
Lautstärkeregelung durch Pulsweitenmodulation



Problem:

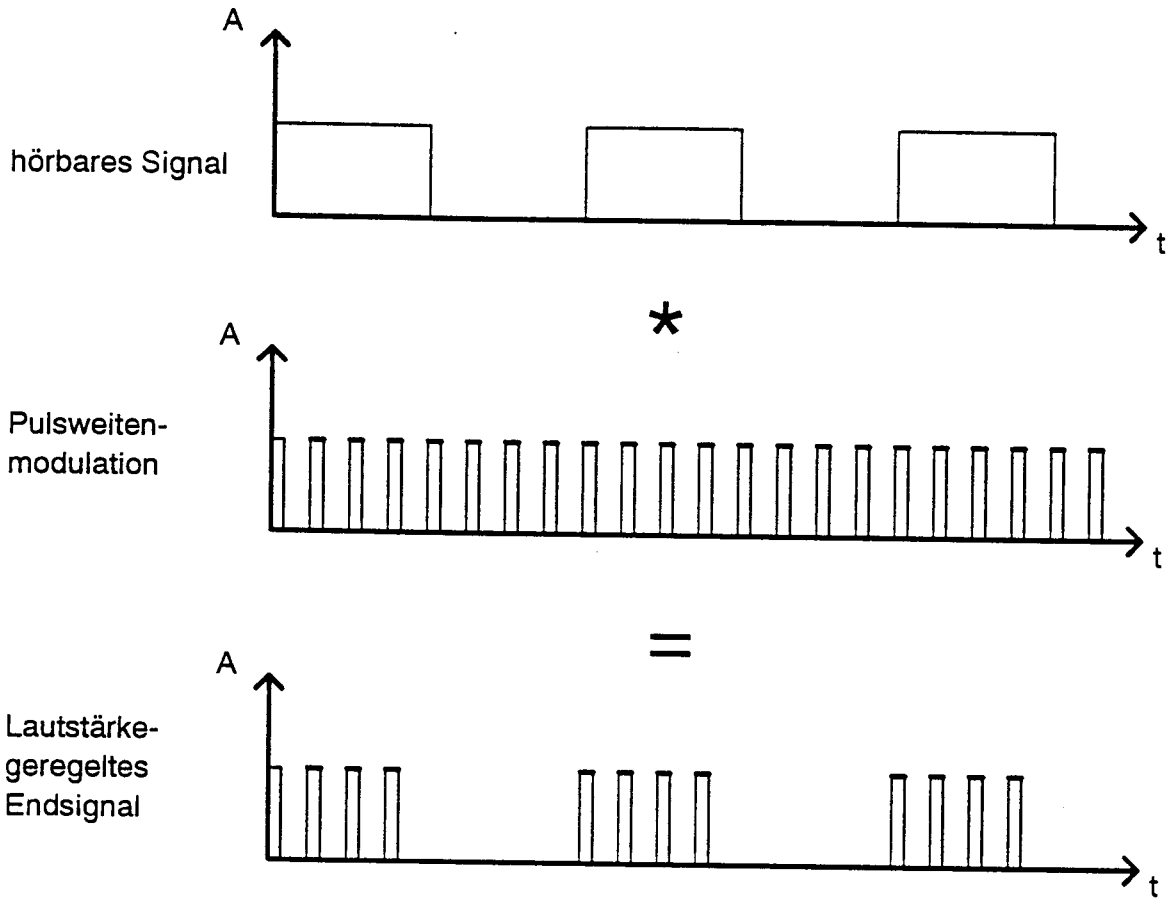
Wird die Pulsweite eines symmetrischen Rechtecksignals verändert, kommen auch geradzahlige Oberwellen im Hörbereich hinzu.
Obwohl die Grundschwingung in ihrer Amplitude abnimmt, wird keine Lautstärkendifferenz wargenommen.





Lösung:

Ein pulsweitenmoduliertes Rechtecksignal muß in einen Bereich oberhalb der Hörgrenze verlagert und mit dem eigentlichen Nutzsignal multipliziert werden.



Das Rauschsignal wird auf die gleiche Weise geregelt.