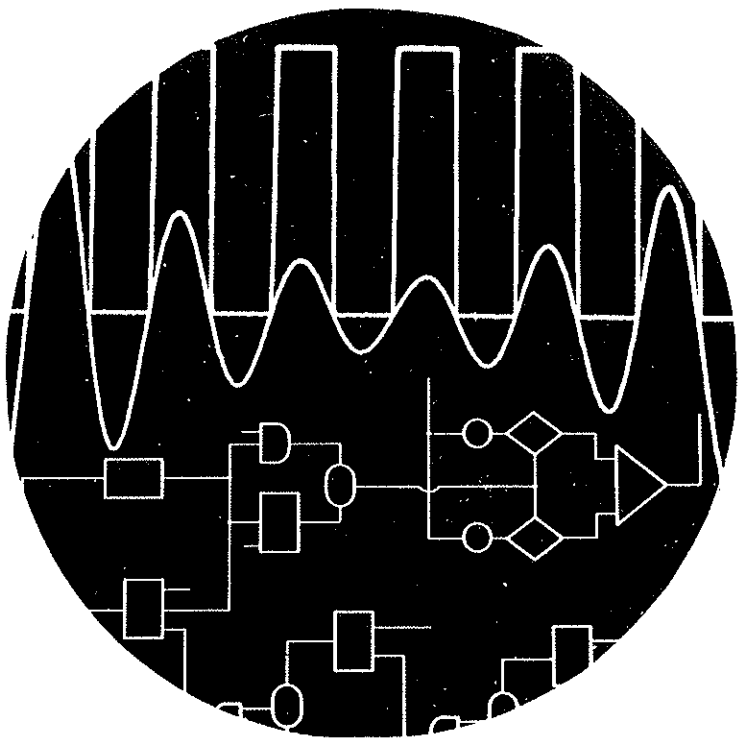
Interface

herausgegeben von der
Abt. Hybridrechenanlage des
EDV-Zentrums der
Technischen Universität Wien

Nummer 18
Jänner 1982



Der Pipelined Boolean Processor ersetzt
die parallele Logik im AutoPATCH-System

INHALTSVERZEICHNIS

Redaktion: Irmgard Husinsky
Eigentümer, Herausgeber, Verleger: EDV-Zentrum der Technischen Universität Wien, Abteilung Hybridrechenanlage
Vervielfältigung: Österreichische Hochschülerschaft Technik
Alle: Gußhausstraße 27-29, A-1040 Wien
Telex: 76875 rzthw a

Ein Vergleich des Rechenbetriebes unter den Betriebssystemen JCS/TS 7 und JCS/VS 8

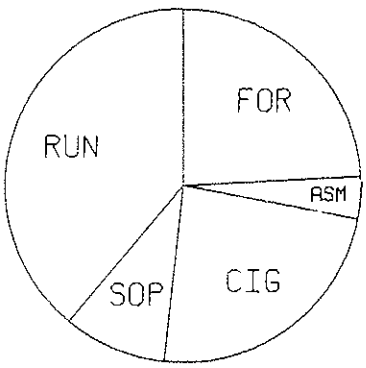
E. Wittek

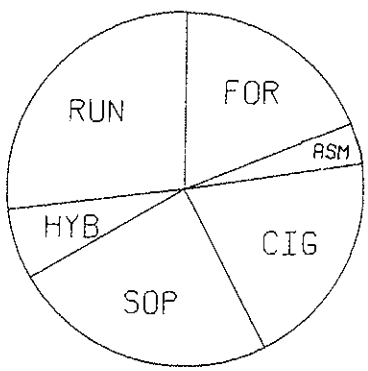
Anfang März 1981 erfolgte die Betriebssystemumstellung von JCS/TS 7 auf JCS/VS 8 (siehe INTERFACE 17). Nach neunmonatigem Einsatz des neuen Systems ist nun, Ende 1981, ein erster Vergleich zwischen den beiden Systemen möglich (der für den Vergleich herangezogene Zeitraum: 1.3.80 - 30.11.80 JCS/TS 7 und 1.3.81 - 30.11.81 JCS/VS 8). Für den Vergleich wurde die Jobstatistik herangezogen, die die Anzahl der Tasks und die verbrauchte Rechenzeit bezüglich der einzelnen Prozessoren (FORTRAN-Compiler (FOR), Assembler (ASM), Core Image Generator (CIG), Source Prozessor (SOP), hybrider Prozessor (HYB)), die Anzahl der Runs mit der verbrauchten Rechenzeit, sowie die Ein/Ausgabe am Card Reader (CR), Line Printer (LP), Data Plotter (DT) und auf den Terminals (TE) enthält.

Die Anzahl der Benutzer, die in diesem Zeitraum an der Hybridrechenanlage gerechnet haben bzw. rechnen ist ungefähr gleich (116 im Vorjahr und 121 heuer), die Anzahl der Tasks hat sich aber gegenüber 1980 um 30% erhöht. Diese steigende Anzahl von Tasks ist hauptsächlich auf die größere Anzahl von Terminal-Sessions zurückzuführen, da immer mehr Benutzer anstelle von Lochkarten mit Source Files arbeiten.

TASKS

	1980	1981	ANSTIEG IN %
FOR	6687	6750	1
ASM	1087	1313	21
CIG	6570	7138	9
SOP	2590	8700	236
HYB*	} 10788	2300	
RUN		9719	11



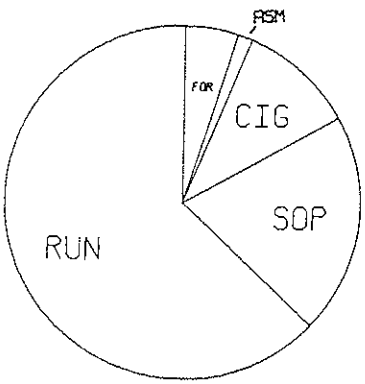


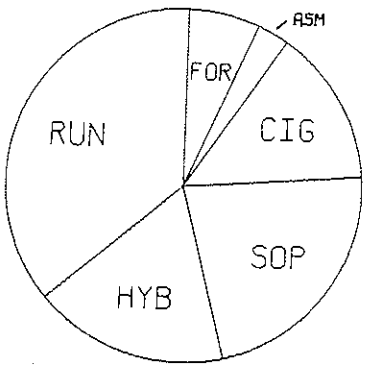
*) Im Betriebssystem JCS/TS 7 wurden hybride Tasks (Verwendung des hybriden Prozessors HYBSYS) nicht separat erfaßt.

Obwohl die Anzahl der Tasks von insgesamt 27722 auf 35920 gestiegen ist, ist die benötigte Rechenzeit (CPU-Sec) von 2.4 Mill. auf 1.2 Mill., das heißt um die Hälfte, gesunken. Da gegenüber dem Vorjahr keine Hardwareänderungen am Digitalrechner durchgeführt wurden, ist diese enorme Leistungssteigerung allein auf das neue Betriebssystem zurückzuführen. Mehr als 30% der Tasks sind interaktive Tasks, bei denen ein Großteil der Zeit auf Input gewartet wird. Daher ist die Maschinenauslastung umso besser, je mehr Jobs vom System parallel bedient werden. (In der derzeitigen Konfiguration können bis zu vier Batch-User und bis zu sieben Terminal-User parallel arbeiten im Gegensatz zu maximal zwei Usern in JCS/TS 7.)

CPU-STD

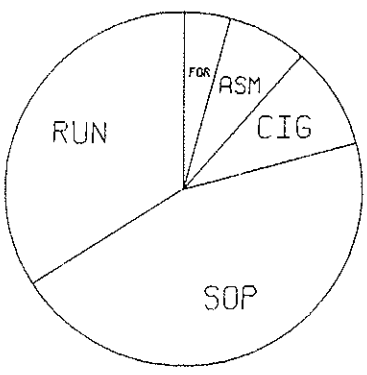
	1980	1981
FOR	37.67	22.77
ASM	9.21	10.14
CIG	72.00	50.27
SOP	137.29	78.70
HYB	} 427.76	62.85
RUN	}	127.92

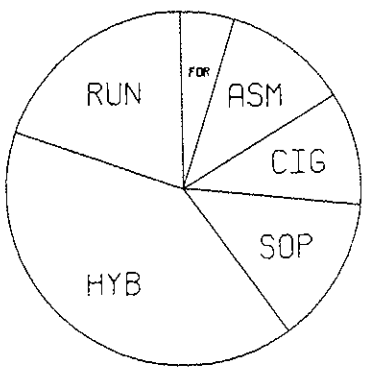




CPU-SEC/TASK

	1980	1981		RÜCKGANG IN %
FOR	17.59	12.15		31
ASM	30.51	27.81		9
CIG	39.45	25.35		36
SOP	190.83	32.56		83
HYB	} 142.75	98.37	} 57.14	
RUN	}	47.38	}	60

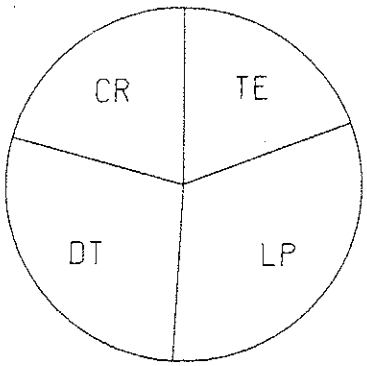


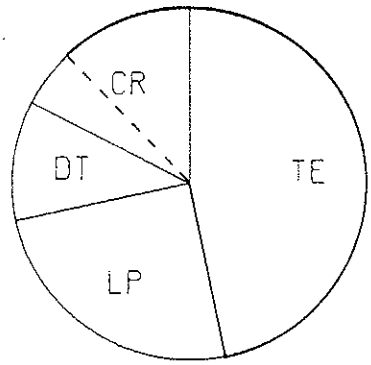


Eine starke Veränderung ist auch in der Input/Output-Struktur zu bemerken.

I/O KILOWORTE

	1980	1981
CR	85428	89708
DT	117422	56779
LP	130942	128348
TE	79772	241215





Der Output am Data Plotter ist auf die Hälfte gesunken, was aber großteils auf vermehrte Zeichenarbeit im Zusammenhang mit der Entwicklung einer neuen graphischen Software im Vorjahr zurückzuführen ist. Der Output am Line Printer ist ungefähr gleich geblieben. Zum Card Reader Input ist zu vermerken, daß auch der Source-Input der Batch-Jobs als Card Input verrechnet wird, sodaß zum Terminal I/O noch ungefähr 3/4 des Card Reader Inputs dazugezählt werden müssen und die eigentliche Eingabe durch Karten nur ca. 1/4 des angegebenen CR-Inputs beträgt. Nimmt man an, daß durchschnittlich 40 Zeichen auf eine Karte gelocht werden, so wurden im Vorjahr pro Minute ungefähr 24 Karten eingelesen, im Gegensatz zu 5 Karten heuer. An den Terminals werden im Durchschnitt pro CPU-Sekunde 15 Zeichen eingegeben und 1080 Zeichen ausgegeben.

Zusammenfassend kann gesagt werden, daß durch das neue Betriebssystem die Maschine viel besser ausgelastet ist und die Ein-/Ausgabe am Terminal durch den virtuellen Source Prozessor von 20% auf 60% der gesamten Ein-/Ausgabe angestiegen und daher die Ein-/Ausgabe auf den papierverbrauchenden Devices um 40% gesunken ist.

Am Rande sei noch bemerkt, daß höchstwahrscheinlich durch ein komfortableres Betriebssystem die Benutzer mehr nach der Trial-and-Error-Methode arbeiten. Denn waren es im Vorjahr nur 5.3% der gesamten Tasks, die durch das Auftreten eines Fehlers abgebrochen wurden, so sind es heuer bereits 13.83%.

aktuelle mitteilungen

ÖFFNUNGSZEITEN

Die Hybridrechenanlage ist von

Montag bis Freitag
von
8 Uhr bis 18 Uhr geöffnet.

ACHTUNG: NEUE TELEFONNUMMERN

Ab 5.1.1982 gelten neue Telefonnummern an der Technischen Universität. Die Kurzrufnummer der Technischen Universität Wien lautet 56 01. Die wichtigsten Klappen an der Hybridrechenanlage sind:

Leiter: Dipl.Ing.Dr. Wolfgang Kleinert	Klappe 3702
Operatoren	Klappe 3706

NEU: PARALLELE LOGIK

Im Zuge der AutoPATCH-Erweiterung steht nun der an der Hybridrechenanlage entwickelte logische Parallelprozessor (PBP= Pipelined Boolean Processor) mit entsprechender Software-Unterstützung durch das Hybrid Simulation System HYBSYS zur Verfügung. Somit können nun auch Aufgaben der parallelen Logik mit HYBSYS gelöst werden. Siehe auch "Der Pipelined Boolean Processor als Ersatz für die parallele Logik des EAI 680 Analogrechners" auf Seite 19.

DOKUMENTATION

Ab sofort ist ein neues, überarbeitetes HYBSYS User Manual erhältlich (S 50,-), das auch die neue parallele Logik beinhaltet.

Betriebssystem JCS/VS 8

Das im März 1981 installierte neue Betriebssystem JCS/VS 8 konnte nunmehr in fast allen Punkten mit den geplanten Eigenschaften bereitgestellt werden. Eine Übergangsphase war notwendig, um eine Zeitlang die Kompatibilität zum alten System JCS/TS 7 zu gewährleisten.

In dem variabel konfigurierbaren virtuellen System zeigte sich, daß eine Konfiguration mit 4 parallelen Batch Usern und 7 parallelen Terminal Usern den derzeitigen Anforderungen genügt.

Verbessert wurde die Garbage-Collection, falls Files gelöscht werden, sodaß auf den Districts möglichst viel Platz zur Verfügung gestellt werden kann. Für jeden File wird nun ein File-Attach-Count mitgeführt, der jeden Zugriff zu diesem File registriert.

Zur Unterstützung des Datenschutzes ist für Terminal-Benützer ein Log-In nur über ihren spezifischen Pass-Code möglich. Dieser ist nur dem Benützer bekannt und garantiert relative Datensicherheit im Zusammenhang mit der Möglichkeit, Files vor fremdem Zugriff zu schützen. Solche geschützte Files können von fremden Benützern gar nicht gelistet werden.

Kleinbuchstaben und Sonderzeichen können für File-Namen, aber auch in FORTRAN- oder Assembler-Formaten kodiert werden. Alle Run-Time-Fehlermeldungen werden am Terminal ausgeschrieben.

Abspeichern von Programmen, neue Districteinteilung für Benützer

Die Districts 13, 14 und 19 - 24 stehen allen Benützern zur Verfügung. Auf diesen Districts können Programme zur späteren Verwendung abgespeichert werden. Die Districts 13, 19, 20 und 21 sind für Benützer des gesamten Hybridsystems reserviert. Auf District 14 sollen digitale Source Files erstellt werden. Auf den Districts 22 und 23 können Closed-Shop-Programme abgespeichert werden. District 24 kann im Time-Sharing-Betrieb für digitale Programme verwendet werden. Im Interesse aller Benützer sollte diese Einteilung eingehalten werden.

			DISTRICT	
			SOURCE FILES	OBJECT CORE IMAGE DATENFILES
USER	HYBRID		13	19,20,21
	DIGITAL	Terminal	14	24
		Closed-Shop	14	22,23

District 16 steht für Datenfiles auf Magnetband zur Verfügung.
Eine aktuelle Liste der Benützerdistricts ist am Anschlagbrett bei der Hybridrechenanlage ausgehängt.
Nach dem Ablaufen einer Jobnummer werden alle zugehörigen Files gelöscht. Ebenso werden Files, auf die mehr als zwei Monate nicht zugegriffen wurde, gelöscht.

KURSE des Arbeitsbereiches "Regelungstheorie und Hybridrechentechnik" des Instituts für Technische Mathematik:

Kurs AH 1+2 15.2. - 19.2.1982
Simulationen an den neuen Kleinrechnern EAI 1000 - Einführung in das AutoPATCH-System - Prozessor HYBSYS

Kurs AH 3+4 22.2. - 26.2.1982
AutoPATCH-System mit Prozessor HYBSYS und Plotterunterstützung - Hybride Verfahren und Anwendungen: gewöhnliche Differentialgleichungen, partielle Differentialgleichungen, Integral- und Funktionalgleichungen, Optimierung (Overlayunterstützung in HYBSYS)

Auskünfte: Dr.F. Breitenecker, Dr.F. Rattay (Klappen 3746, 3747, 3754)

HYBSYS - Aktuelle Informationen und Mitteilungsfile

Seitdem das Rechnen mit HYBSYS nicht mehr an den Ort der Hybridrechenanlage gebunden ist, da man HYBSYS von jedem angeschlossenen graphischen Terminal aus starten kann, ist es notwendig, die Benützer über das Terminal über aktuelle Ereignisse zu informieren. Zu diesem Zweck gibt es nun einen Informationsfile, der sowohl den Benützer informiert als auch Bemerkungen der Benützer aufnimmt. Dieser Informationsfile enthält Mitteilungen über:

- Zeitpunkt der letzten Release
- bekannte Software-Fehler der momentanen Release
- Hardware-Fehler (gesperrte Makros)
- Neuerungen (Hardware und Software)
- Ankündigungen von HYBSYS-Kursen, Rechner-Betriebszeiten
- andere HYBSYS-Informationen

Der entsprechende Befehl heißt INFO. Mit INFO; erhält man den Inhalt des Informationsfiles am Terminal.

INFO=text; merkt eine Meldung zur Eintragung in den Informationsfile vor, wobei die Jobnummer des Benützers auch vermerkt wird. Diese Benützermeldungen werden jedoch erst nach Bestätigung durch das HYBSYS-Personal in den Informationsfile aufgenommen. Es sollen nur Probleme eingetragen werden, die offensichtlich fundamentale Hard- oder Softwareprobleme mit HYBSYS betreffen. Für Run-Time-Meldungen an den Operator steht der Befehl NOTE zur Verfügung (NOTE=text;).

kurse

RH1 GERÄTETECHNIK EAI PACER 600A AUTOPATCH-SYSTEM
Termin: 1982/03/17
Vortragender: Dr. W. Kleinert

RH3 HINWEISE FÜR FORTRAN-PROGRAMMIERER AN DER HYBRIDRECHENANLAGE
Termin: 1982/03/09
Vortragender: Dipl. Ing. F. Blöser

RH7 SOFTWAREUNTERSTÜTZUNG FÜR DIE BENÜTZUNG DES PACER 600 ALS PLOTTER SYSTEM
Termin: 1982/03/09 und 1982/03/10
Vortragender: Dipl. Ing. F. Blöser

RH9 EINFÜHRUNG IN DAS HYBRIDE AUTOPATCH-SYSTEM
Dieser Kurs wird nach Bedarf (mindestens zwei Teilnehmer) für wissenschaftliche Benützer, die keinerlei Vorkenntnisse auf dem Gebiet der hybriden Programmierung besitzen und an einer möglichst raschen Problemlösung interessiert sind, abgehalten.

Die folgende Kurse werden nach Vereinbarung abgehalten:

RH2 BENÜTZUNG DES BETRIEBSSYSTEMS JCS/VS 8

RH6 EAI ASSEMBLER

RH11 ASSEMBLER-PROGRAMMIERUNG FÜR FORTGESCHRITTENE MIT ÜBUNGEN

RH13 BEDIENUNG DES HYBRIDEN PROZESSORS HYBSYS
RH14 ZUR SIMULATION DYNAMISCHER SYSTEME AM AUTOPATCH-SYSTEM, TEIL 1 BZW. TEIL 2

RH16 PROGRAMMENTWICKLUNG AM TERMINAL

Nähere Auskünfte und Anmeldungen zu den Kursen telephonisch oder persönlich bei Herrn M. Schandl (1040 Wien, Gußhausstraße 27-29. 4. Stock, Zimmer 1404/05, Tel: 56 01 / 3706DW).

PROGRAMMGESTEUERTES EDITIEREN VON SOURCE FILES

A. Blauensteiner, E. Wittek

Im letzten INTERFACE wurde der Source Processor JCSSOP, der das Generieren und Manipulieren von Source Files ermöglicht, beschrieben. Source Files haben ein Datenformat, das spezifisch auf das Editieren ausgerichtet ist, wobei es unwesentlich ist, ob die Informationen Daten für einen Programmlauf oder ein Programm selbst sind.

Seit kurzem können mit Hilfe von FORTRAN-aufrufbaren Unterprogrammen JCSSOP-Commands programmgesteuert exekutiert werden, d.h. durch FORTRAN- bzw. Assembler-Programme können auf Source Files abgespeicherte Programme geändert, verbessert oder erweitert werden.

Zeitraubende Textersetzungen, die über den gesamten Source File erfolgen sollen, können programmgesteuert im Closed-Shop-Betrieb durchgeführt werden. Programmänderungen können in Abhängigkeit von Resultaten erfolgen, wobei der User die Resultate nicht selbst auswerten muß. Z.B.: die Anfangswerte für eine Iteration stehen in einem DATA-Statement oder werden als Input von einem Source File genommen. In Abhängigkeit vom Resultat können die betreffenden Werte verändert und das Programm wieder exekutiert werden. Programmoptimierungen wie z.B. Ersetzungen von immer wieder auftretenden Befehlssequenzen durch Subroutine-Aufrufe können automatisiert werden.

Durch das Unterprogramm ATTACH wird der angegebene Source File positioniert, durch NEW ein neuer Source File generiert, wobei die abzuspeichernde Information mit Hilfe eines Feldes an das Unterprogramm übergeben wird. Alle nachfolgenden JCSSOP-Commands beziehen sich immer auf den zuletzt angesprochenen Source File. Ein Source File unterteilt sich in eine Anzahl von fortlaufend numerierten Lines, beginnend mit 1. Viele Befehle von JCSSOP beziehen sich nur auf einen eingeschränkten Bereich des Files, der durch eine untere und obere Bereichsgrenze definiert ist. Dieser Bereich kann durch die Subroutinen RANGE, GO, JUMP, SKIP und MOVE definiert bzw. verändert werden, wobei RANGE die untere und obere Bereichsgrenze festlegt, GO den Bereich auf eine einzige, durch ihre Nummer angegebene Line beschränkt (der Inhalt der Line steht zugleich als Output zur Verfügung), durch JUMP und SKIP der Source File von der oberen bzw. unteren Bereichsgrenze an auf das Vorkommen eines angegebenen Strings durchsucht wird und der Bereich auf die erste diesen String enthaltende Line eingeschränkt wird und MOVE den momentan gesetzten Bereich um eine angegebene Anzahl von Lines nach oben bzw. unten verschiebt.

Das Einfügen von Lines erfolgt durch das Unterprogramm ADD, das Löschen durch die Subroutine PURGE, wobei der gesamte momentan angewählte Bereich gelöscht wird.

Der Inhalt des Source Files kann durch CHANGE, TEXT und SHIFT verändert werden. CHANGE ersetzt den Inhalt der angegebenen Line durch die an das Unterprogramm übergebene Information. Die Line muß dabei im Bereich liegen. Durch TEXT wird ein String durch einen anderen ersetzt, wobei der gesamte gesetzte Bereich auf das Vorkommen des Strings überprüft wird. Beide Strings, die unterschiedliche Länge aufweisen können, sind Eingangsparameter von TEXT. Mit Hilfe von SHIFT wird die Reihenfolge der Source Lines verändert. Der angewählte Bereich wird hinter eine mit ihrer Nummer anzugebende Line versetzt und von der alten Stelle gelöscht.

Ein neuer Source File kann nicht nur durch NEW, sondern auch durch MERGE generiert werden. Der Unterschied liegt darin, daß bei MERGE Teile von bereits vorhandenen Source Files zu einem neuen File zusammengeschlossen werden. Die Teilstücke (maximal 1Ø) werden durch SAVE in eine MERGE-Table eingetragen, wobei der Name des Files, der District und die Bereichsgrenzen vorgemerkt werden. Der neue Source File enthält die Lines in der Reihenfolge der Eintragungen. Der Inhalt dieser Tabelle kann durch CLEART wieder gelöscht werden.

Zum "Listen" von Source Lines stehen zwei Unterprogramme zur Verfügung. LIST listet den angewählten Bereich. MASK listet den angewählten Bereich in Abhängigkeit von einem angegeben Text. Listen bedeutet dabei, daß jede Line, die diesen String enthält, mit ihrer Nummer an das rufende Programm übergeben wird. Genaue Beschreibungen der Subroutinen sind in der Programmberatung erhältlich.

BEISPIEL

Von Zeit zu Zeit sollen Meßdaten durch ein Programm ausgewertet werden. Die Resultate sollen auf einem File abgespeichert werden, wobei bei jeder Auswertung ein neuer File errichtet werden soll. Der Name des neuen Files kann dabei auf folgende Weise an das Programm übergeben werden, ohne daß dabei das Programm selbst geändert werden muß:
Mit Hilfe der Option NAME=namexy auf der /RUN-Steuerkarte und der Routine SYSØ74 kann der Name namexy an das Programm übergeben werden. Die Steuerkarten für das Programm seien auf dem Source File STATSO (District 2Ø) abgespeichert:

```
/JOB,USER=nr
/RUN STATC,LOAD=2Ø,NAME=STAT/Ø
/END
```

Wird dieser Source File nun exekutiert (z.B. vom Terminal aus mit Hilfe von BATCH), so wird das Programm STATC vom District 2Ø exekutiert, wobei der Name STAT/Ø an das Programm übergeben wird. Unmittelbar am Beginn des Programmes STATC muß die Routine SYSØ74 aufgerufen werden, um den in der NAME-Option angegebenen Namen auf ein lokales Feld abzuspeichern. Am Ende des Programmes STATC kann durch CHANGE die NAME-Option im File STATSO geändert werden, sodaß beim nächsten Lauf von STATC ein neuer Filename zur Verfügung steht.

```
INTEGER NAME(3), LINE(2Ø)
CALL SYSØ74 (NAME)              STAT/* wird auf NAME abgespeichert.
   .
   .
   .
CALL FILE (NAME, 2Ø, 5Ø, 88)
   .
   .
   .
CALL ATTACH (6HSTATSO, 2Ø)      Der Source File STATSO wird positioniert.

CALL GO (2, 2Ø, LINE, NR)       Der Bereich wird auf die Line 2 einge-
                                schränkt, gleichzeitig steht der In-
                                halt auf LINE zur Verfügung.

LINE(15) = LINE(15)+1           Der zu ändernde Textteil steht auf
                                Position 3Ø (15. Wort des Feldes LINE).

CALL CHANGE (NR, 2Ø, LINE)      Die Line mit der Nummer NR wird durch
                                LINE ersetzt.

CALL EXIT
END
```

Anstelle des Befehles CALL GO (...) gibt es noch andere Möglichkeiten, die Line mit der NAME-Option im Source File zu finden:

CALL SKIP (2, 4H/RUN, 2Ø, LINE, NR)
Jede Zeile des Source Files wird überprüft, ob sie mit /RUN beginnt, und der Bereich wird auf die erste gefundene Line gesetzt. Der Inhalt der Line und ihre Nummer stehen als Output zur Verfügung.

CALL SKIP (3, 6H>NAME=, 2Ø, LINE, NR)
bzw. CALL SKIP (3, 6H>STAT/, 2Ø, LINE, NR)
Jede Zeile des Source Files wird überprüft, ob sie den String NAME= bzw. STAT/ enthält und der Bereich auf die erste diesen String enthaltende Line gesetzt. Der Inhalt der Line und die Nummer stehen wieder als Output zur Verfügung.

EAI Computer Users' Group Meeting in Glasgow

E. Wittek

An der Universität von Glasgow trafen am 15. September 1981 Teilnehmer aus elf verschiedenen Ländern zum EAI Computer Users' Group Meeting 1981 ein. Am Vormittag wurde die Möglichkeit geboten, das Rechenzentrum der Universität zu besichtigen. Nach dem Mittagessen wurde das Meeting durch Prof. Richards eröffnet. Anschließend daran referierten vier Universitätsangehörige über an der Universität durchgeführte Simulationen. Am Abend fuhren alle Teilnehmer zum eigentlichen Veranstaltungsort, dem Hotel Loch Lomond in der Nähe von Glasgow.

Während der nächsten zwei Tage wurden 19 technische Papers vorgetragen. Im Unterschied zum letzten Meeting wurde an beiden Tagen die Möglichkeit geboten, an Working Groups teilzunehmen. Es wurden Themen wie ECSSL, EAI-PDP-Connections, Hard- und Software behandelt, wobei auch die Antworten, die von EAI auf die im letzten Jahr ausgearbeiteten Fragen gegeben wurden, diskutiert wurden. In zwei EAI Sessions berichteten Mitarbeiter von EAI über neue Produkte und es gab Diskussionen zwischen EAI und den Mitgliedern der Users' Group. Aus Zeitmangel mußten einige sehr interessante Diskussionen abgebrochen werden und daher wurde das Meeting auf allgemeinen Wunsch um eine zusätzliche Session am Freitag Vormittag verlängert.

EAI Computer Users' Group Meeting 1982

Prof. Bremšak von der Universität Ljubljana, Jugoslawien, hat sich bereit erklärt, das nächste Users' Group Meeting Ende September 1982 zu organisieren. Wie schon bei den letzten zwei Meetings wird auch diesmal nur der erste Tag an der Universität verbracht. Der eigentliche Veranstaltungsort ist der Seebadeort Portorož.

METAASSEMBLER AN DER HYBRIDRECHENANLAGE

F. Berger, A. Blauensteiner

EINLEITUNG

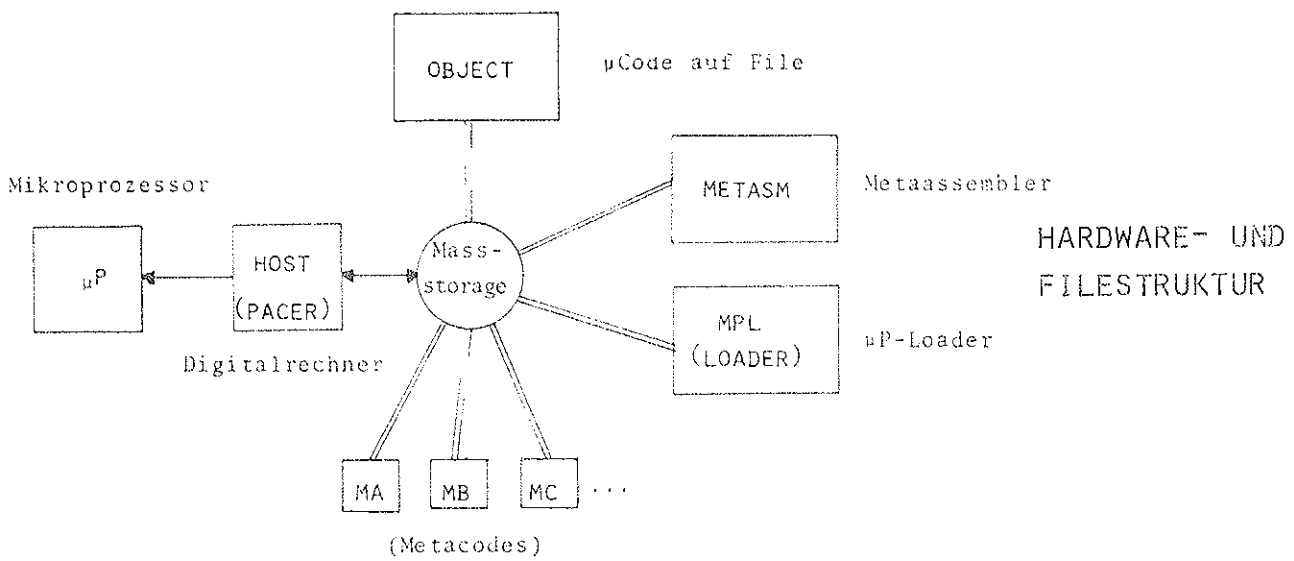
Seit Mai 1981 gibt es an der Hybridrechenanlage die erste Version des Metaassemblers METASM.

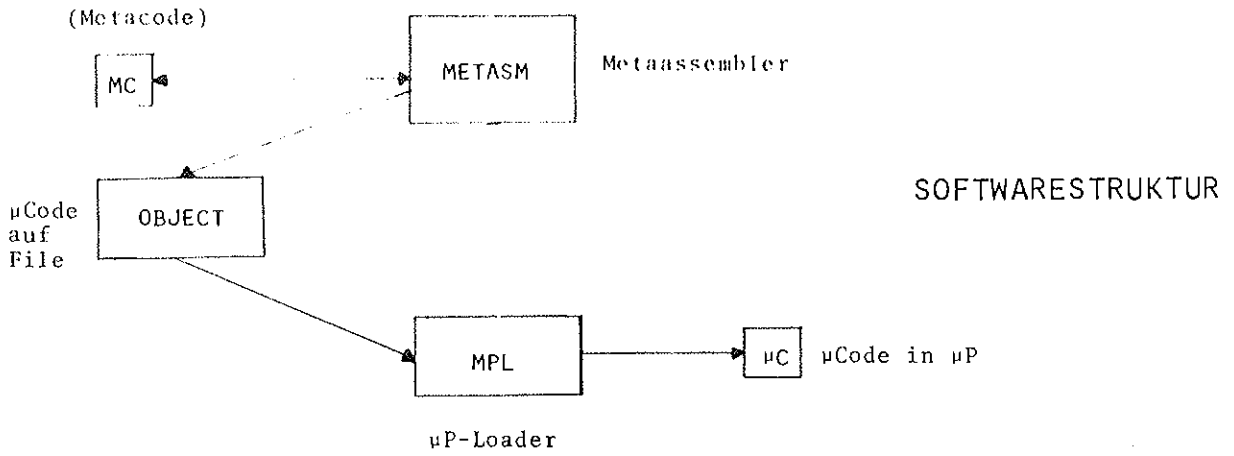
Ein Assembler ist ein Programm, das ein anderes, in einer bestimmten symbolischen Syntax geschriebenes Programm interpretiert und übersetzt und daraus einen binären Output erzeugt. Ein Metaassembler ist eine allgemeinere Form eines Assemblers, der besonders für die Erstellung von Mikroprogrammen geeignet ist.

Ein Metaassembler unterscheidet sich von einem gewöhnlichen Assembler vor allem dadurch, daß die Syntax vom Benutzer vor dem eigentlichen Assemblieren selbst definiert werden muß. Beim traditionellen Assembler kann der Benutzer selbst Marken für Instruktionen und Namen für spezielle Datenworte definieren, die Instruktionen selbst, d.h. die Syntax, die Formate,die Wortlänge etc. sind vom Assembler fest vorgegeben.

Ein Metaassembler muß aber flexibler sein, da er für verschiedene Hardwarekonfigurationen mit verschiedenen Formaten und Wortlängen (z.B. Mikroinstruktionen über 100 Bit) verwendet werden soll. Außerdem werden hier oft nur wenige Bits in einem ganzen Wort definiert, wobei der Rest des Wortes unverändert bleiben soll.

All dies erfordert, daß ein Metaassembler aus zwei unabhängig durchzuführenden Teilen besteht: dem Definitionsteil für die Festlegung der Syntax und dem eigentlichen Assemblierungsteil, der die im Definitionsteil definierte Syntax, die definierten Wortlängen, Konstanten und Formate berücksichtigt.





METASM DEFINITIONSTEIL

Einige Codes, sogenannte Pseudo-Codes, sind fest vorgegeben.
Der Pseudo-Code DEF kennzeichnet den Beginn eines Definitionsteils und er bewirkt das Errichten eines Metacode Files. Mit dem Pseudo-Code META kann ein bereits vorhandener File an den neu definierten angefügt werden. Die Anzahl der Halbbytes, die ein vom Benutzer definiertes Wort enthalten muß, kann mit dem Pseudo-Befehl WORD angegeben werden. Dieser Befehl kann mehrfach verwendet werden, sodaß mit verschieden langen Operationscodes gearbeitet werden kann. Der Befehl INVRT bewirkt die komplementierte Darstellung aller folgenden Befehle. Mit dem Pseudo-Code DUMMY kann der Wert der sogenannten Don't Care Bits festgelegt werden. Das Ende des Definitionsteils wird durch den Pseudo-Code END gekennzeichnet.

Der Benutzer kann beliebige Befehle definieren, wobei er den Operationscode, eine Kennzeichnung, ob der Befehl eine Konstante oder Variable (Adresse, Tag) referenziert, die Wortlänge des Befehls und die entsprechende Bitdefinition angeben muß.

METASM ASSEMBLIERUNGSTEIL

Das Metaassemblerprogramm entspricht einem herkömmlichen Assemblerprogramm.
Es beginnt mit dem fest definierten Befehl META, dann folgt das eigentliche Programm mit den Marken, den vom Benutzer definierten und auf einem File residenten Operationscodes, den Adressen oder Konstanten und den Kommentaren.

Jedes Metaassemblerprogramm kann auch beliebig viele EQU (Equate-Befehle) enthalten. Der Befehl END beschließt das Programm.

DOKUMENTATION

Die Dokumentation wird durch die Ausgabe folgender Listen unterstützt:

- Liste des Definitions-Passes (Ausgabe des Definitionsteils),
- Liste der zwei Assembler-Passes (Ausgabe des Assemblierungsteils),
- Tag Table (Liste der definierten Marken mit Adressen und Reference Count),
- mit dem Pseudo-Befehl LIST kann zusätzlich eine Liste aller gültigen definierten Operationscodes mit entsprechenden Bitmustern ausgegeben werden.

Eine Reihe von Fehlermeldungen unterstützen die Korrektur der Programme.
Ein Metaassembler User Manual ist an der Hybridrechenanlage erhältlich.

BEISPIEL

Beispiel eines Mikroprozessors, der die Funktion $f(t) = 2t^2 + 14t$ laufend auswerten soll, wobei t an einem ADC anliegt und f(t) auf einem DAC ausgegeben werden soll. Die Instruktionen seien PACER-ähnlich definiert.

```
METAASSEMBLER  METASM 8.02   JCS/VS 8 * PACER 100      PAGE     1

DEFINITION PASS   META FILE:  METEST    27

     2: ****************************************************
     3: *
     4: *      PACER-LIKE CODES       DEFINITION PASS
     5: *
     6: ****************************************************
     7:                          0004          WORD:       4   16 BIT WORDS
     8:             *META*                     ADC             1:5L1H10L       ANALOG DATA IN
     9:             *META*                     DAC             1:5L2H8L1H      DIGITAL DATA OUT
    10:             *META*                     DEC     V       1:16T           CONSTANT DEFINITION
    11: *Frame*                                MUL             :2L2H           MULTIPLY CODE
    12: *Frame*                                E               :2L1H           RELATIVE ADDRESSING
    13: *Frame*                                DIS     V       :9D             DISPLACEMENT FIELD
    14: *Frame*                                SHIFT           :2L1H1L2H3L1H1L ALD SHIFT CODE
    15:             *META*                     ALD     V       1:(SHIFT)5T     ARITHMETIC SHIFT
    16:             *META*                     J       V       1:LH2L(E)(DIS)  JUMP UNCONDITIONALLY
    17:             *META*                     LA      V       1:2H2L(E)(DIS)  LOAD ACCU
    18:             *META*                     STA     V       1:3H1L(E)(DIS)  STORE ACCU
    19:             *META*                     M       V       1:(MUL)(E)(DIS) MULTIPLY
    20:             *META*                     A       V       1:2HLH(E)(DIS)  ADD TO ACCU
    21:             *META*                     MALD15  V       2:(MUL)(E)(DIS)(SHIFT)1L4H      M and ALD 15
    22:                                        END
```

LIST OF DEFINITIONS STORED IN META FILE: METEST 27

```
  1:  ADC          L      1     4   LLLL LHLL LLLL LLLL
  2:  DAC          L      1     4   LLLL LHHL LLLL LLLH
  3:  DEC      V   L      1     4   TTTT TTTT TTTT TTTT
  4:  MUL          Frame         4   LLHH
  5:  E            Frame         3   LLH
  6:  DIS      V   Frame         9   DDDD DDDD D
  7:  SHIFT        Frame        11   LLHL HHLL LHL
  8:  ALD      V   L      1     4   LLHL HHLL LHLT TTTT
  9:  J        V   L      1     4   LHLL LLHD DDDD DDDD
 10:  LA       V   L      1     4   HHLL LLHD DDDD DDDD
 11:  STA      V   L      1     4   HHHL LLHD DDDD DDDD
 12:  M        V   L      1     4   LLHH LLHD DDDD DDDD
 13:  A        V   L      1     4   HHLH LLHD DDDD DDDD
 14:  MALD15   V   L      2     4   LLHH LLHD DDDD DDDD LLHL HHLL LHLL HHHH
```


ASSEMBLER PASS 1 CODE SET: METEST OBJECT FILE: RUN/MM 38


```
  2: ************************************************************
  3: *
  4: *       F(t)=2t*t+14t
  5: *
  6: ************************************************************
  7: 0000  0400                          LOOP   ADC              AD CONVERSION
  8: 0001  E20B                  000C           STA    T         SAVE
  9: 0002  320A                  000C           M      T         POWER 2
 10: 0003  2C50                  0010           ALD        16    2t*t TO ACCU
 11: 0004  E209                  000D           STA    ZTQ       SAVE
 12: 0005  C207                  000C           LA     T         FETCH t
 13: 0006  3205                  000B           M      K14       14 TIMES
 14: 0007  2C4F                  000F           ALD        15    14t TO ACCU
 15: 0008  D205                  0000           A      ZTQ       PLUS 2t*t
 16: 0009  0601                                 DAC              DA CONVERSION
 17: 000A  43F6                  0000           J      LOOP      LOOP ALL THE TIME
 18: 000B  000E                  000E   K14     DEC        14    CONSTANT
 19: 000C  0000                  0000   T       DEC         0    TEMPORARY
 20: 000D  0000                  0000   ZTQ     DEC         0    TEMPORARY
 21:                                            END
```


TAG TABLE:

TAGNAM	ADR	REF	TAGNAM	ADR	REF	TAGNAM	ADR	REF
LOOP	0000	1	K14	000B	1	T	000C	3
ZTQ	000D	2						

Beispiel der Verwendung einer Mehrwortinstruktion für das gleiche Programm.

```
MALD15  =  M       C
           ALD     15
```

```
METAASSEMBLER  METASM 8.02   JCS/VS 8 * PACER 100     PAGE     7

 2: ************************************************************
 3: *
 4: *      F(t)=2t*t+14t
 5: *
 6: ************************************************************
 7: 0000  0400                          LOOP   ADC              AD CONVERSION
 8: 0001  E20C                  000D           STA    T         SAVE
 9: 0002  320B 2C4F             000D           MALD15 T         t*t TO ACCU
10: 0004  2C41                  0001           ALD         1    2 TIMES IN ACCU
11: 0005  E209                  000E           STA    ZTQ       SAVE
12: 0006  C207                  000D           LA     T         FETCH t
13: 0007  3205 2C4F             000C           MALD15 K14       14t IN ACCU
14: 0009  D205                  000E           A      ZTQ       PLUS 2t*t
15: 000A  0601                                 DAC              DA CONVERSION
16: 000B  43F5                  0000           J      LOOP      LOOP ALL THE TIME
17: 000C  000E                  000E   K14     DEC        14    CONSTANT
18: 000D  0000                  0000   T       DEC         0    TEMPORARY
19: 000E  0000                  0000   ZTQ     DEC         0    TEMPORARY
20:                                            END
```

```
METAASSEMBLER  METASM 8.02   JCS/VS 8 * PACER 100     PAGE     8

TAG TABLE:

TAGNAM   ADR   REF      TAGNAM   ADR   REF      TAGNAM   ADR   REF

LOOP     0000    1      K14      000C    1      T        000D    3
ZTQ      000E    2
```

Der Pipelined Boolean Processor als Ersatz für die parallele Logik des EAI 68Ø Analogrechners

W. Kleinert, E. Wittek

Wie bereits in INTERFACE 15/16 beschrieben, bestehen die Kernstücke der Hardware unseres PACER 6ØØ AutoPATCH-Systems aus einer von EAI entwickelten analogen Schaltmatrix und einem an der Hybridrechenanlage von W. Kleinert entwickelten Pipelined Boolean Processor. Mit Hilfe der analogen Schaltmatrix können die analogen Komponenten des EAI 68Ø Parallel Processors programmgesteuert verbunden werden. Der Pipelined Boolean Processor (PBP) dient als Ersatz für die parallele Logik der 68Ø und ist nunmehr voll in das AutoPATCH-System integriert. Er wird ebenfalls mit Hilfe des hybriden Prozessors HYBSYS programmiert [1].

Die Hardware des Pipelined Boolean Processors (PBP)

Das Design des PBP basiert auf der von A. Asthana und J. Shotliff im Jahre 1977 vorgeschlagenen Idee eines Parallel Logic Processors [2], unterscheidet sich aber grundlegend von der dort vorgeschlagenen Realisierung. Abbildung 1 zeigt ein Blockdiagramm des PBP.

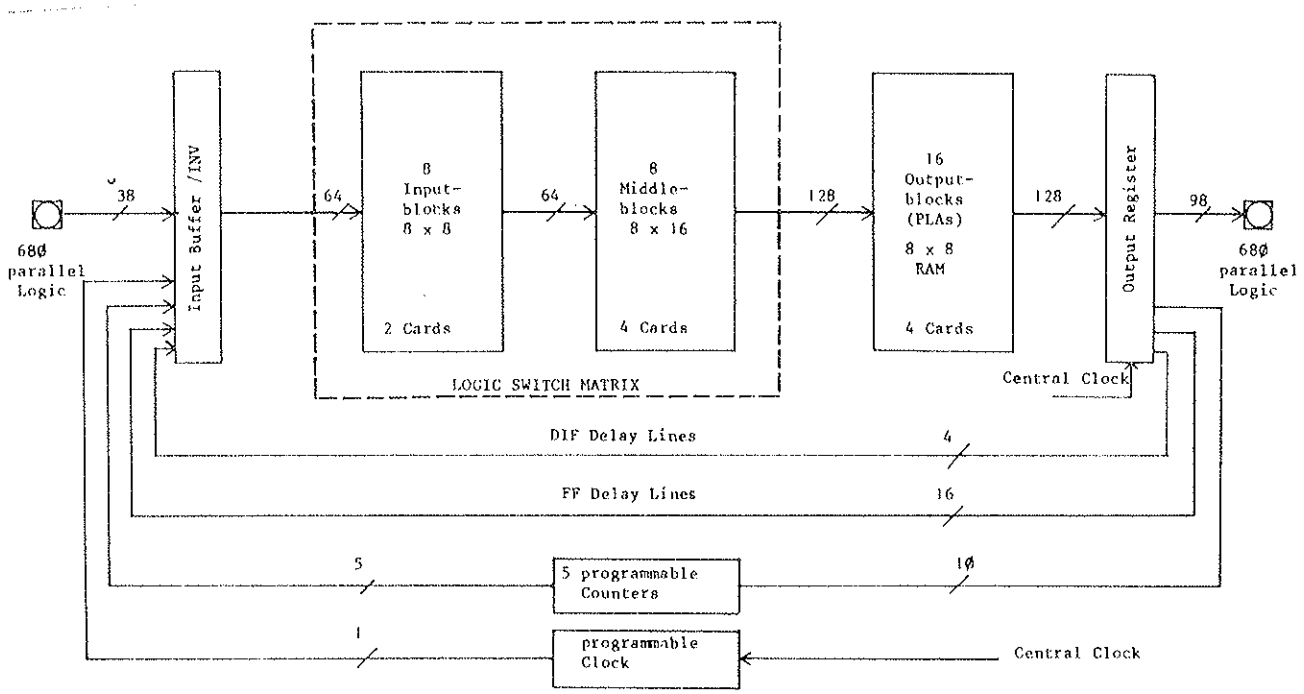


Abbildung 1

Blockschaltbild des PBP

Anmerkung: Dieser Artikel stellt die überarbeitete deutsche Fassung eines Vortrags dar, der auf dem EAI Users' Group Meeting 1981 in Glasgow gehalten wurde.

Die zwei Hauptteile sind eine logische Schaltmatrix und eine Anzahl von programmierbaren Logic Arrays (PLA) mit insgesamt 64 Eingängen und 128 Ausgängen. Die logische Schaltmatrix besteht aus acht Inputblöcken (8 x 8) und acht Mittelblöcken (8 x 16). Jeder dieser Blöcke besteht aus einer Anzahl von 8 x 1-Multiplexern mit zugeordneten Registern für die Schaltstellungen. Durch entsprechende Schalterpositionen kann zu jedem der 16 Outputblöcke eine beliebige Untermenge von maximal acht Inputsignalen geroutet werden. Jeder Outputblock kann als programmierbare Multiport-Device aufgefaßt werden, die acht logischen Outputvariablen maximal acht logische Inputvariable zuordnen kann. Realisiert wurden diese Outputblöcke mit 256 x 8 RAMs, die mit Wahrheitstabellen, die den benötigten Booleschen Operationen entsprechen, geladen werden.

Zwischen einem Input- und Mittelblock bzw. einem Mittel- und Outputblock besteht genau eine Verbindung. Daher ist, sobald ein entsprechender Mittelblock gewählt wurde, die Verbindung zwischen einem spezifischen Input und einem spezifischen Output eindeutig. Die Inputsignale erscheinen als Input an den PLAs in der durch einen Algorithmus ausgewählten Reihenfolge der Mittelblöcke.

Ein Register am Ausgang der Outputblöcke, das durch eine zentrale Clock getaktet wird, synchronisiert alle Outputsignale. Aus diesem Grund kann ein Flip-Flop durch eine einfache Rückführung, die als Delay-Line fungiert, und zusätzlichen Booleschen Verknüpfungen, die die Funktion eines S/R bzw. eines D-Flip-Flops beschreiben, realisiert werden. In die Rückführung wurden auch fünf Counter eingebaut, die durch einen LSI Chip (AM 9513, Abbildung 2) realisiert sind. Jeder dieser Counter hat zwei Eingangssignale, Source und Gate, und kann in 18 verschiedenen Modes programmiert werden.

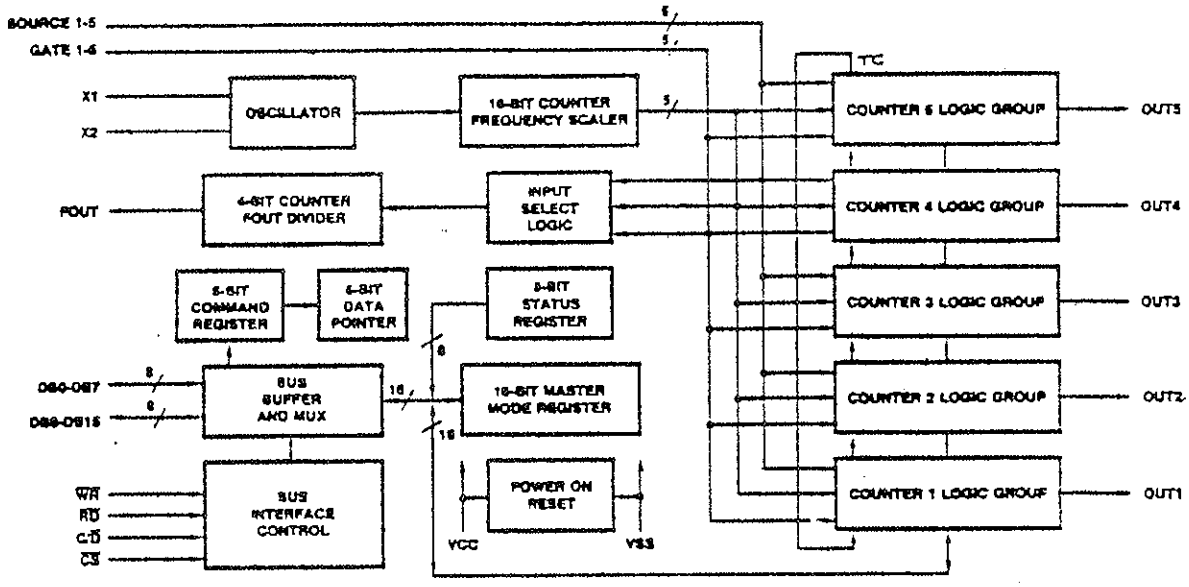


Abbildung 2

Blockschaltbild des Timer-Bausteins AM 9513

Wie schon erwähnt, ist der PBP eine Realisierung bereits bekannter Ideen, aber unterscheidet sich wesentlich in der Art der Implementierung. Um die physikalischen Dimensionen zu minimieren und um den Durchsatz zu erhöhen, wurden für jeden 8 x 8-Block des PBP Time-Sharing-Techniken verwendet. Die Time-Sharing-Register der Input-, Mittel- und Outputblöcke fungieren als Pipelineregister einer dreistufigen Pipeline. Das interne Timing des PBP ist mikroprogrammierbar und wird von einer 2ØMHz Clock gesteuert. Werden keine Flip-Flops und Counter verwendet, kann ein 2MHz-Takt verwendet werden. Bei Verwendung von Rückführungen muß die Zeit zwischen zwei zentralen Clock-Impulsen wegen der internen Gate Delays 1.6 µsec betragen.

PBP-SOFTWARE

Die Software unterteilt sich in zwei große Abschnitte:
- die Deklaration der logischen Variablen und Gleichungen,
- das Aufbringen des deklarierten Modells auf den PBP.

Von der Seite des HYBSYS-Benutzers sind es die beiden Befehle DECLAR und PREPAR, wobei durch die Exekution des Befehls PREPAR für die Logik intern folgende Funktionen durchgeführt werden:

PREPAR
- Zuordnung der Hardware-Komponenten
- Routen
- Berechnung der Wahrheitstabellen
- Laden und Initialisieren des PBP

DEKLARATION VON BOOLESCHEN FUNKTIONEN

Auf Grund der blockorientierten HYBSYS-Syntax muß jeder logische Ausdruck in einzelne logische Operatoren zerlegt werden. Im Gegensatz zu analogen Operatoren wird nicht jedem logischen Operator ein Hardware-Makro zugeordnet, da durch die Verwendung der RAMs in den Outputblöcken ein Teil der Booleschen Operationen wie AND, OR usw. allein durch die Berechnung ihres Wahrheitswertes mit Hilfe von Unterprogrammen realisiert werden kann. Dadurch können auch die logischen Outputvariablen Funktionen von mehr als einer Inputvariablen sein, was einen großen Unterschied zu den analogen Schaltproblemen darstellt.

Die Inputvariablen für den PBP sind Komparatoren, Controllines (=logische Konstante), zentrale Timer-Signale (z.B. OP) und externe Logiksignale.

Die logischen Funktionen AND, NAND, OR, NOR, EQU und XOR müssen als Operatoren deklariert werden und verknüpfen eine nichtlimitierte Anzahl von logischen Inputs. Jedem dieser Operatoren ist ein Unterprogramm zur Auswertung des Funktionswertes zugeordnet. Die Boole sche Funktion NOT muß nicht als eigener Operator deklariert werden, sondern wird mit Hilfe des Vorzeichens "-" beschrieben.

Flip-Flops, Differentiatoren und Counter sind logische Operatoren mit einer limitierten Anzahl von Eingangssignalen. Flip-Flops haben zwei Eingangssignale (Set und Reset). Wird nur das Set-Signal deklariert, so wird auf den Reset-Eingang automatisch das negierte Set-Signal gelegt, wodurch das Flip-Flop als einfaches Delay fungiert. Wie schon oben erwähnt, wird ein Flip-Flop durch eine einfache Rückführung und der entsprechenden Wahrheitstabelle realisiert.
Die Deklaration eines Differentiators resultiert in einer Zuordnung einer differenzierenden Delay-Line.

Mit Hilfe spezieller HYBSYS-Befehle werden die fünf Counter programmiert: z.B.: Auswahl der Modes, Laden der Register, Starten der Counter, usw.

Output des PBP können einzelne Inputvariable oder Boole sche Operatoren sein; diese sind fest verdrahtet, um Switches, Track/Stores, Senselines, externe logische Signale und Interrupts zu steuern.

ROUTEN

Zum Routen der logischen Schaltmatrix kann wegen des schon vorher erwähnten Unterschieds zwischen analogen und logischen Problemen nicht derselbe Algorithmus wie zum Routen der analogen Schaltmatrix verwendet werden. Darüber hinaus müssen von der Software auch folgende hardwarebedingte Einschränkungen überprüft werden:

- eine logische Outputvariable kann nur eine Funktion von maximal acht verschiedenen Inputvariablen sein,
- die Anzahl der verschiedenen Inputvariablen, die zu einem PLA geroutet werden, ist mit acht limitiert, d.h. die maximal acht logischen Funktionen, die einem PLA zugeordnet werden können, können insgesamt von maximal acht verschiedenen Inputvariablen abhängig sein.

Sind alle deklarierten Inputvariablen den entsprechenden Outputblöcken zugeordnet, so können die Verbindungen durch Auswahl der entsprechenden Mittelblöcke hergestellt werden. Der Algorithmus für die Auswahl der entsprechenden Mittelblöcke ist identisch mit dem für die analoge Schaltmatrix und wurde von EAI entwickelt. Wie schon vorher erwähnt, ist die Verbindung eines Inputblockes zu einem Mittelblock und eines Mittelblockes zu einem Outputblock fest verdrahtet und daher vorherbestimmt. Aus diesem Grund wird auch durch den Mittelblock das Inputpin des PLA , zu dem eine entsprechende Inputvariable geroutet wird, bestimmt.

BERECHNUNG DER WAHRHEITSTABELLEN

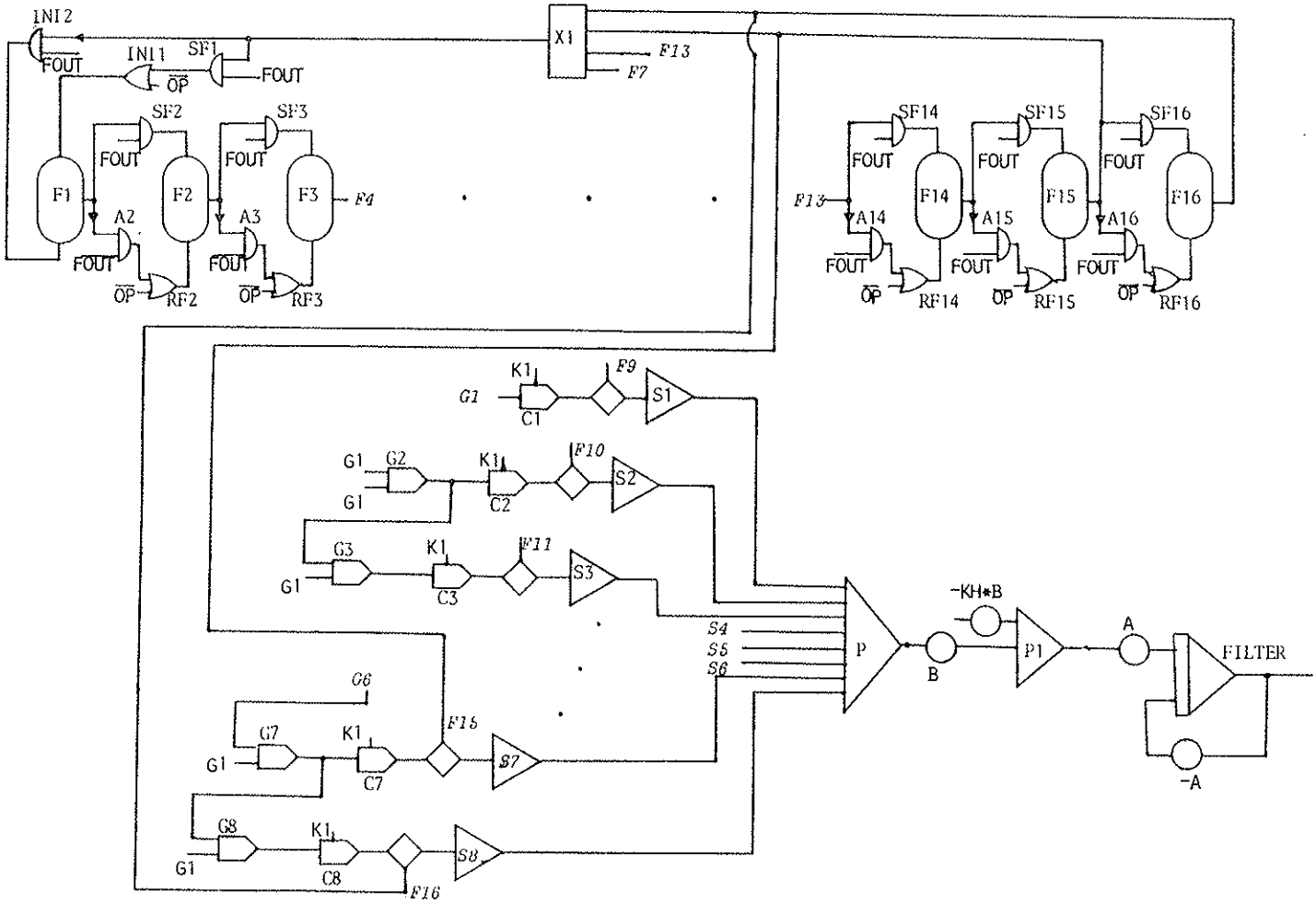
Nach dem Routen kann jeder PLA-Input mit einer Inputvariablen der logischen Schaltmatrix identifiziert werden und es müssen die Boole schen Funktionen, die den entsprechenden Outputpins eines PLAs zugeordnet sind, für alle möglichen binären Kombinationen der Eingangsvariablen berechnet werden. Da nicht vorausgesetzt werden darf, daß nicht verwendete Inputs der PLAs einen definierten Wert (z.B. .FALSE.) besitzen und der Wert der Inputvariablen zugleich die Adresse im RAM angibt, müssen die Funktionswerte für das gesamte RAM, d.h. 256, berechnet werden, um den Einfluß früherer Schaltungen zu eliminieren.
Da diese Berechnungen einen großen Teil der Gesamtzeit, die für PREPAR benötigt wird, ausmacht, werden sie nur für die verwendeten Inputs durchgeführt. Diese berechneten Werte werden dann in alle RAM-Plätze, die den nicht verwendeten Inputs entsprechen, abgespeichert. Werden zum Beispiel nur zwei Inputvariable zu einem PLA geroutet, so müssen alle diesem PLA zugeordneten Boole schen Operationen nur für vier verschiedene binäre Kombinationen ausgewertet werden, d.h. für jede Boolesche Funktion erspart man sich 252 Auswertungen.

BEISPIEL

Um die Verwendung der grundlegenden logischen Operatoren zu demonstrieren, wurde folgendes Beispiel gewählt:

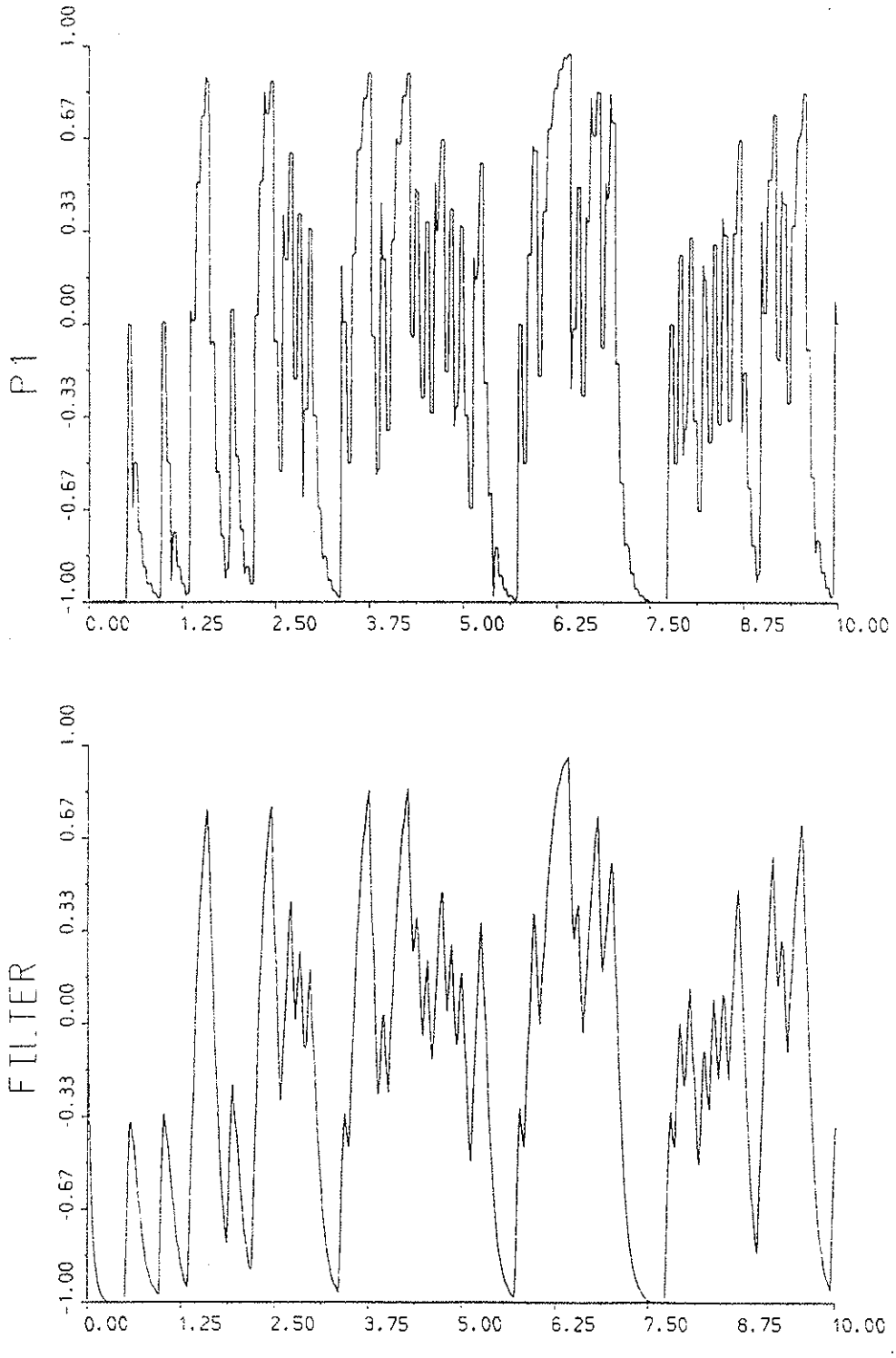
Ein Pseudozufallszahlengenerator soll mit einem Ringschieberegister und Antivalenz-Rückkopplung realisiert werden. Um die ganze zur Verfügung stehende Hardware auszunützen, wurden 16 Flip-Flops gewählt, obgleich sich damit keine Periode des Generators der Länge 2^{16} reproduzieren läßt (siehe [3]). Mit Hilfe von 8 Flip-Flops, ebensovielen Schaltern und einem Summierer wurde ein Digital-Analog-Wandler realisiert. Das so erhaltene binäre Rauschen (P1) wird durch einen Tiefpaßfilter (FILTER) zu einem gleichverteilten Pseudorauschen.

BLOCKDIAGRAMM



SIMULATION MIT HYBSYS

```
DECLAR FF:F[1:16]
DECLAR AND:SF[1:16],A[2:16],INI2
DECLAR OR:RF[2:16],INI1
DECLAR XOR:X1

DECLAR F[2:16]=SF[Ø],RF[Ø]
DECLAR SF[2:16]=F[-1],FOUT
DECLAR A[2:16]=-F[-1],FOUT
DECLAR RF[2:16]=A[Ø],-OP
DECLAR F1=INI1,INI2
DECLAR INI2=-X1,FOUT
DECLAR SF1=X1,FOUT
DECLAR INI1=SF1,-OP
DECLAR X1=F15,F16,F7,F13

DIGITL
DECLAR PAR:G1=2
DECLAR MULT:G[2:8]
DECLAR DIV:C[1:8]

DECLAR G[2:8]=G[-1],G1
DECLAR C[1:8]=K1,G[Ø]

ANALOG
DECLAR PAR:A=2Ø,B=2,KH=.5
DECLAR SWI:S[1:8]
DECLAR SUM:P,P1
DECLAR INT:FILTER

DECLAR S[1:8]=F[8]*C[Ø]
DECLAR P=S1,S2,S3,S4,S5,S6,S7,S8
DECLAR P1=P*B,-KH*B
DECLAR FILTER=-A*FILTER,P1*A

PREPAR
FOUT=75
CNT,VAL:
PLOT P1
PLOT FILTER
```



LITERATUR

[1] D. Solar: HYBSYS User Manual, Version 4M/A. Hybridrechenanlage, Technische Universität Wien, 1981

[2] A. Asthana,J. Shotliff: A Parallel Logic Processor for Automatically Patched Hybrid Computers. EAI, 1977

[3] Golomb, S. W.: Shift Register Sequencies. Holden-Day San Francisco, California, 1967

Ein schneller 32-Bit Arithmetik-Coprocessor für den Pacer 100

W. Kleinert

Die relativ langsame Floating Point-Arithmetik des PACER 100 stellt für einige unserer Benützer ein wirkliches Problem dar, sogar wenn der externe Hardware Floating Point Processor (FPP) verwendet wird. Der FPP von EAI ist eine I/O Device, die über den Standard I/O-Kanal mit einer Sequenz von DO, DF und DI Befehlen programmiert werden muß. Obwohl die Floating Point-Operationen innerhalb des FPPs in einer annehmbaren Zeit durchgeführt werden, verursacht der notwendige Datentransfer einen sehr großen Overhead. Verwendet man die Floating Point Inline Option des FORTRAN-Compilers, so bezahlt man etwas schnellere Rechenzeiten mit wesentlich mehr Speicherplatzbedarf.

In Tabelle 1 sind für drei verschiedene einfache FORTRAN-Zuweisungen die Zeiten, die zur Berechnung dieser Gleichungen mit Hilfe der verschiedenen Floating Point-Arithmetiken am PACER 100 notwendig sind, angegeben.

CPU-Zeiten in µsec
(ohne Verwendung der multilevel indirekt Adressierung)

	NONFPP	FPPRTL	INLINE	Neuer FP-COPROCESSOR
A=B	125	83	50	7.4
A=B*C	285	125	74	10.6
A=B*C+D	400	170	96	17.2

Tabelle 1

Obwohl die Verwendung von Inline-Coding eine Verbesserung um fast einen Faktor 4 bezüglich der NONFPP-Version bringt, sind die Rechenzeiten ungefähr 10mal so langsam wie die internen FPP-Zeiten. Ein weiteres Problem resultiert daraus, daß es extrem uneffektiv ist, diesen Typ von FPP in einem Multi-User-Betrieb zu verwenden, da es nur zwei 32-Bit-Register gibt, die bei jedem User-Wechsel über einen 16-Bit-Kanal abgespeichert und neu geladen werden müssen. Aus diesem Grund stand bisher der FPP neben HYBSYS nur bestimmten Sonderjobs zur Verfügung.

Um die Rechenzeiten von Programmen mit sehr viel Floating Point-Arithmetik zu beschleunigen, wurden zuerst die Möglichkeiten betrachtet, einen neuen schnelleren FPP oder einen Array Processor am Standard I/O oder DMA-Kanal zu verwenden. Diese Ideen wurden nicht verwirklicht, da der Engpaß des sukzessiven Transfers von 16-Bit-Worten über den I/O-Kanal erhalten bleibt und dadurch der oben erwähnte Overhead nicht eliminiert werden kann. Am Standard I/O-Kanal müssen zur Übertragung eigene Instruktionen verwendet werden, wobei für das Laden dieser Instruktionen zusätzlich Zeit verbraucht wird.

Mit dem DMA-Kanal kann zwar die volle Bandbreite des PACER 100 Memorys verwendet werden, doch liegt der Nachteil darin, daß es nur eine limitierte Anzahl von Kanälen gibt und außerdem zusätzlich Zeit für die Initialisierung eines jeden Transfers benötigt wird.

Abb.1 zeigt ein Blockdiagramm der PACER 100 CPU, aus dem ersichtlich ist, daß der PACER 100 keine busorientierte Maschine im üblichen Sinn ist. Daher ist es extrem schwierig, ein Shared Memory Interface zu einer sehr schnellen externen Device (z.B. einem Vektor Prozessor) zu entwerfen. Außerdem müßte zusätzlich der größte Teil des FORTRAN-Compilers neu geschrieben werden.

Anmerkung: Dieser Artikel stellt die überarbeitete deutsche Fassung eines Vortrags dar, der auf dem EAI Users' Group Meeting 1981 in Glasgow gehalten wurde.

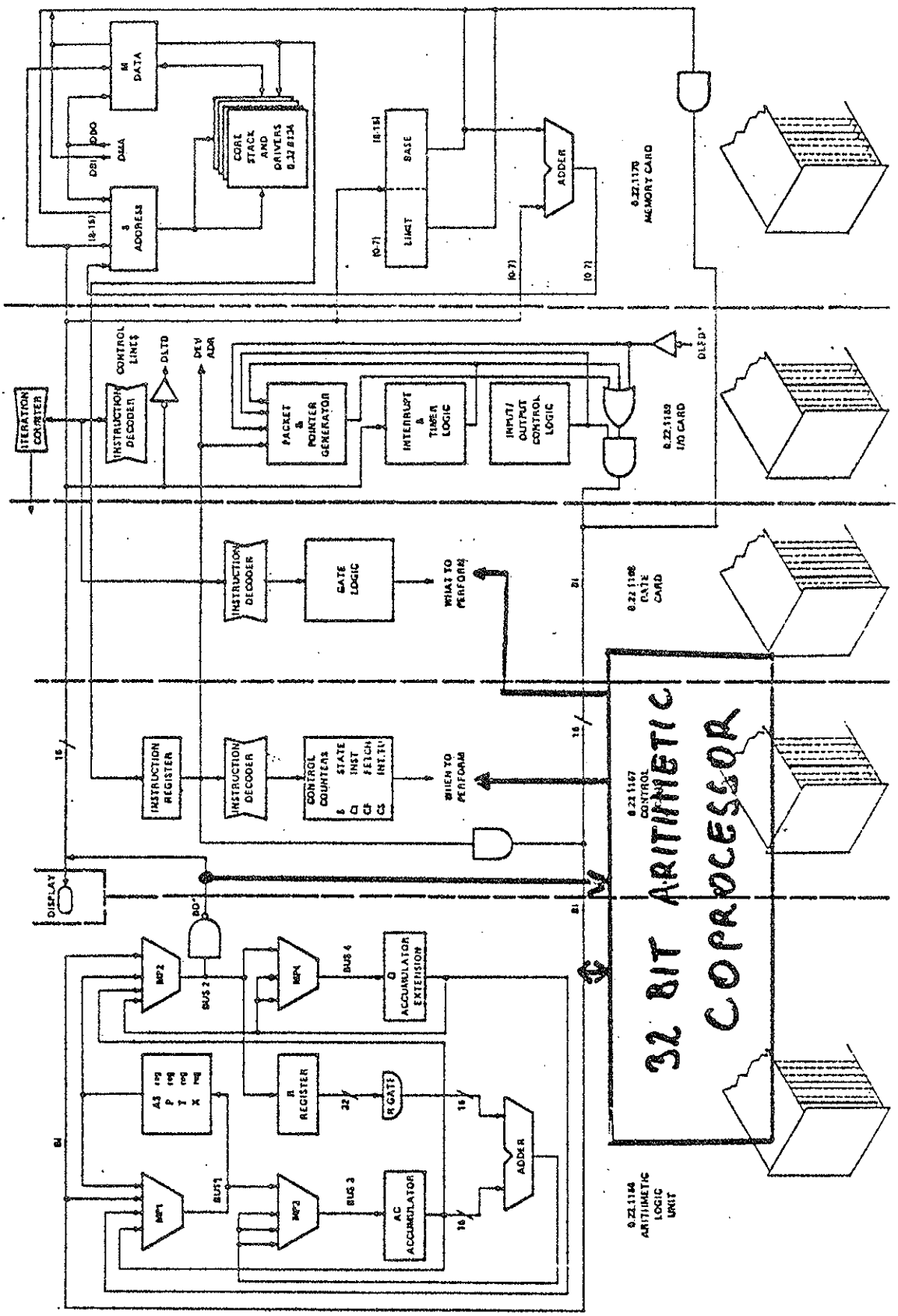


Abbildung 1
P-1ØØ Blockdiagramm

DER NEU ENTWICKELTE ARITHMETIK COPROCESSOR

Um hohe Geschwindigkeiten zu erzielen, sollte eine neue Floating Point Device die internen P-1ØØ BI- und BO*-Busse (siehe Abb. 1) verwenden und durch eine Erweiterung der P-1ØØ Instruktionen programmiert werden. Ein solches Gerät wird aber nirgends angeboten. Daher wurde beschlossen, einen neuen FPP selbst zu entwickeln, wobei folgende Zielvorstellungen dem Design zugrunde gelegt wurden:

- Es sollte eine Art 'Coprocessor' sein, der die internen Signale der CPU und bisher nicht benötigte oder illegale Instruktionen verwendet.
- Floating Point-Operationen sollten unter Verwendung von Standard LSI-Komponenten so schnell wie möglich durchgeführt werden.
- Die Anzahl der Speicherzugriffe sollte minimal sein.
- Die multi-indirect level Adressierung des P-1ØØ sollte verwendet werden können.
- Es sollten genügend Floating Point-Register vorhanden sein, um zwischen acht verschiedenen Usern ohne Zeitverlust wechseln zu können.
- Es sollten solche Instruktionen verwendet werden, die zu einem möglichst effektiven, FORTRAN-Compiler generierten Code führen, wobei die Änderungen am vorhandenen Compiler möglichst gering sein sollten.
- Um Änderungen und neue Entwicklungen leichter realisieren zu können, sollte eine mikroprogrammierbare Hardware verwendet werden.

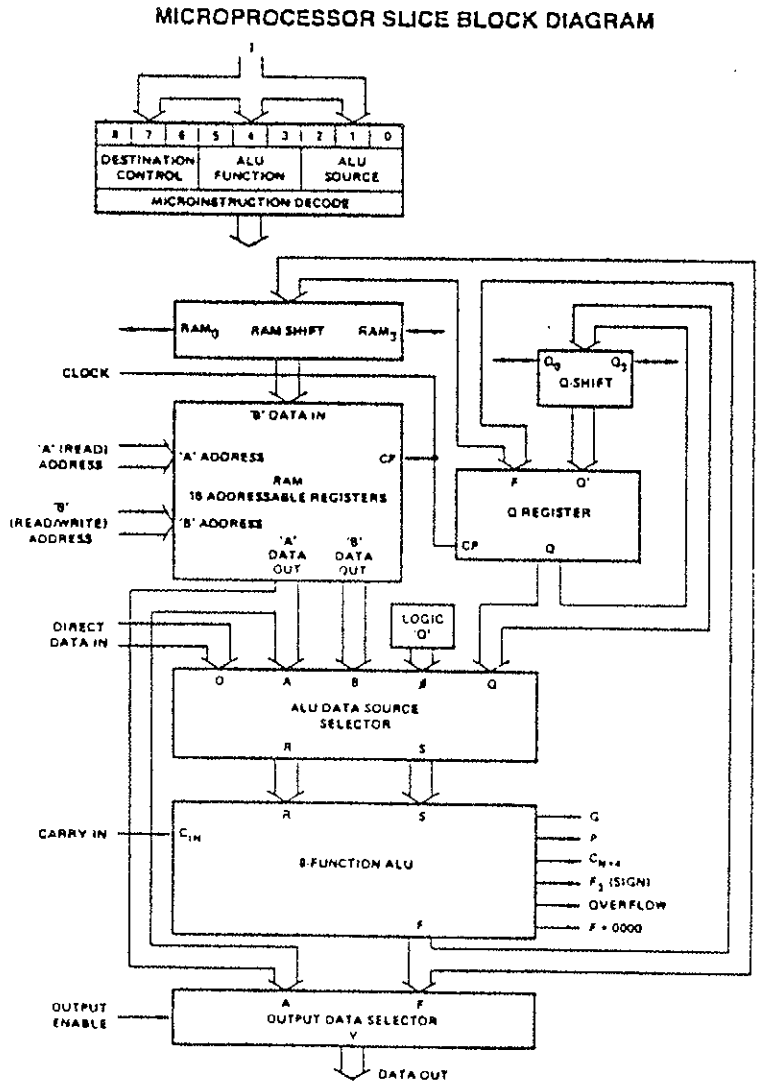


Abbildung 2
29ØIB von AMD

Nicht der einfachste, aber vielleicht fortschrittlichste Entwurf war ein mikroprogrammierbarer Processor mit einer 32-Bit CPU, die aus acht AM 29Ø1B 4-Bit Slice ALUs (siehe Abb. 2) und einem parallelen 24 x 24-Bit Multiplizierer, die durch einen AM 291Ø micro-program Sequencer gesteuert werden, besteht.

BLOCK DIAGRAM OF P-1ØØ ARITHMETIC 32-BIT COPROCESSOR

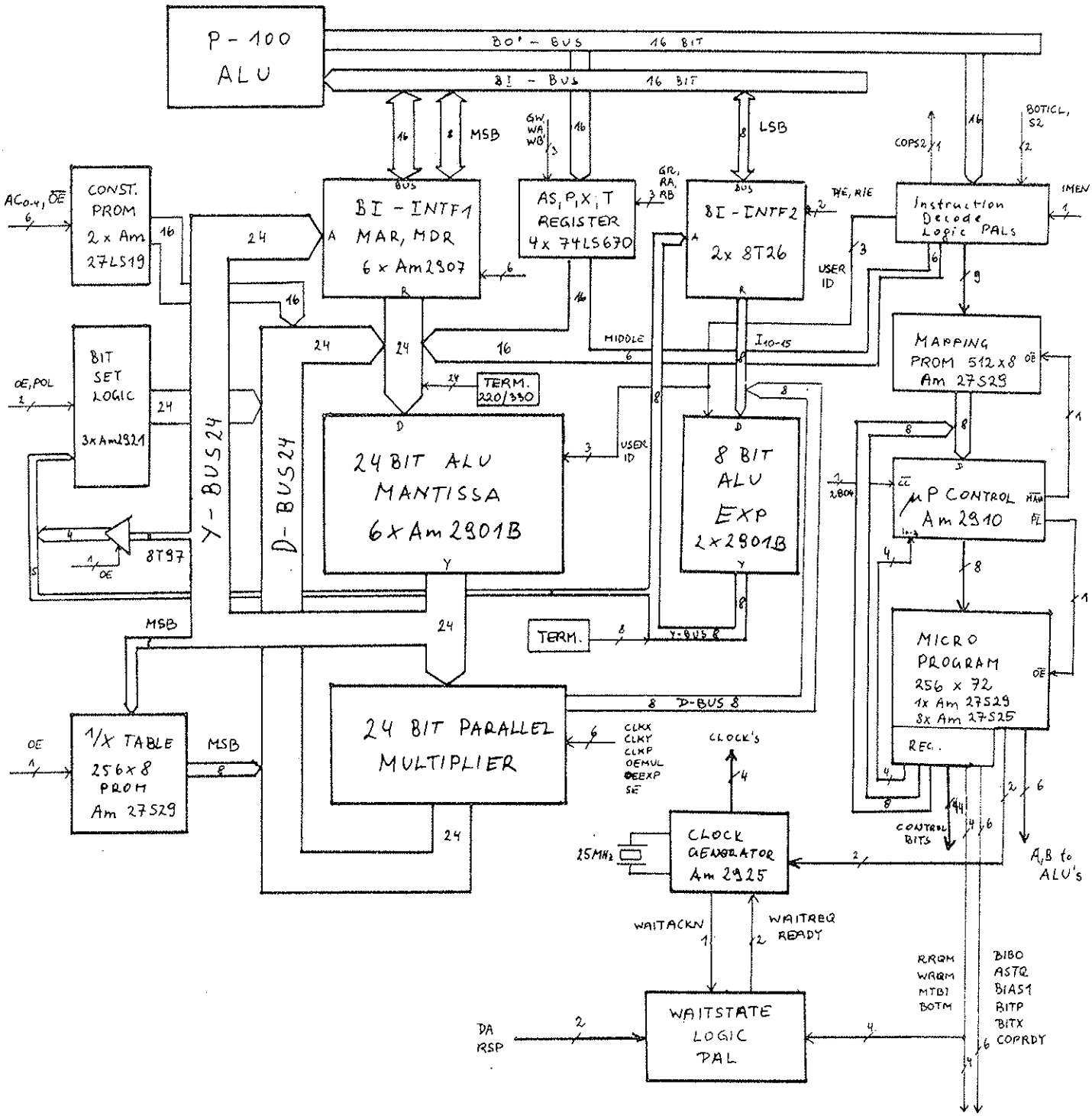


Abbildung 3

ALU SECTION OF P1ØØ ARITHMETIC COPROCESSOR (32 BIT)

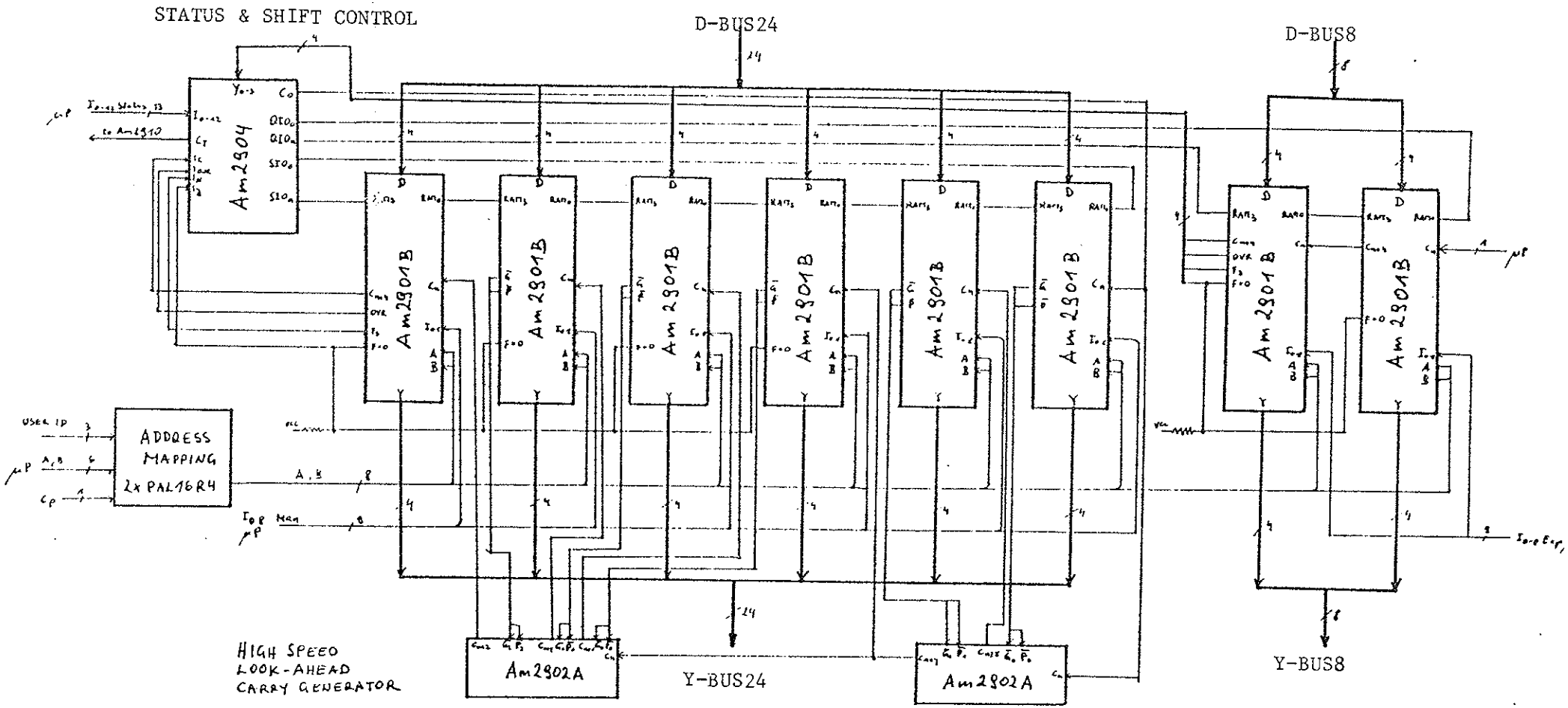


Abbildung 4

Abb. 3 zeigt ein Blockdiagramm des neu entwickelten Floating Point Coprocessors. Dieser Processor ist auf zwei Wirewrap Prototyp-Platinen im Minicontroller-Chassis des PACER untergebracht. Einige kleine Änderungen waren auch auf der PACER Control-Karte, der Gate-Karte, der ALU und im Verdrahtungsteil notwendig.

Wie an Hand des Blockdiagramms ersichtlich ist, besteht der Floating Point Coprocessor aus den folgenden Teilen:

- einer 32-Bit ALU mit 16 two-ported 32-Bit-Registern (8 AM 29Ø1B Slices),
- einem 24 x 24-Bit parallelen Multiplizierer,
- einem Drei-Weg Bus Interface zum BI- und BO*-Bus, das auch Register für Speicheradresse und Wert enthält,
- einem 8 x 8 PROM für die tabellierten Funktionswerte von 1/x,
- einer 24-Bit Set und Reset Unit,
- einem 16-Bit Konstantenspeicher,
- einer Computer Control Unit (CCU), die aus einem Mikroprogramm Sequencer und einem 1K Mikroprogramm-Speicher à 64-Bit mit Pipeline-Registern besteht,
- einem zweiten Instruktionsregister und einem Mapping PROM zum Decodieren,
- einem zweiten Satz der PACER Register AS, X, T, P
- einem mikroprogrammgesteuerten Clock Generator, der die Länge des Microcycles kontrolliert und optimiert.

Der zentrale Teil des 32-Bit-Processors besteht aus einer 24-Bit ALU für die Mantisse, die auch für 16-Bit-Integer-Manipulationen und Adress-Berechnungen verwendet wird, und einer parallel arbeitenden und einzeln mikrogesteuerten 8-Bit ALU für den Exponenten (siehe Abb. 4). Jede dieser ALUs hat 16 Register. Das logische Register Ø dient jedem von 8 Benützern als Floating Point-Akkumulator und wird durch eine dekodierte 3-Bit Time-Sharing-User-Identifikation jeweils einem von acht verschiedenen Registern zugeordnet. Sieben weitere Register werden als Arbeitsregister oder Datapointer während jeder Instruktion verwendet.

Um höchstmögliche Geschwindigkeit zu gewährleisten, hat jede ALU eine Anzahl von sehr schnellen 29Ø2A look-ahead Carry-Generatoren und eine separate Shift und Status Control Unit. Für optimales Timing wird ein AM 2925 Clock Generator zusammen mit einem 25 MHz Oszillator verwendet. Verschieden lange Microcycles (von 12Ø - 24Ø nsec) werden mikroprogramm-gesteuert ausgewählt. Dadurch kann jede Mikroinstruktion mit angemessener Geschwindigkeit exekutiert werden.

Der parallele 24 x 24-Bit Multiplizierer erwartet für die Floating Point Multiplikation vom y-Bus zwei normalisierte 24-Bit Mantissen im Zweier-Komplement. Diese Operanden werden in die Arbeitsregister von drei 12 x 12 parallelen Multiplizierern (TRW MPY 12 HJ) geladen, die das Resultat in 1Ø5 nsec berechnen. Zusammen mit einem 4 x 4-Bit parallel Multiplizierer (74284) für die Berechnung der zwei signifikanten Bits des Produkts der weniger signifikanten Teile der beiden Mantissen, werden in einem Netzwerk von 11 MSI 4-Bit Low Power Schottky ALUs drei Partial-Produkte berechnet und aufaddiert, um eine gerundete und normalisierte 24-Bit Mantisse des Produkts in weniger als 2ØØ nsec zu erhalten. Dieser Multiplizierer wird auch zur Skalierung der Mantisse bei der Floating Point Addition und Subtraktion während der Exponenten-Anpassung verwendet. Der relativ große Aufwand war nötig, da der schon lange angekündigte 24 x 24-Bit Multiplizierer-baustein von TRW (MPY 24 HJ) noch immer nicht erhältlich ist.

DER ERWEITERTE BEFEHLSSATZ

Der Coprocessor kann einen erweiterten Satz von P-1ØØ Instruktionen exekutieren, wobei hauptsächlich zwei der bisher illegalen Instruktionen (siehe Tabelle 2) dekodiert werden. Die Instruktionen '23ØØØ bis '23Ø43 werden für die verschiedenen Arten von Floating Point und Mixed-Mode-Instruktionen verwendet. Einem OP-Code können maximal 3 Zellen, die die Adressen der Operanden enthalten, folgen. Wird vom Coprocessor eine gültige Instruktion entdeckt, so wird zuerst eine Adress- und Operand-Fetch-Phase durchlaufen, wobei Bit Ø der Adresse des Operanden für die indirekte Adressierung verwendet wird. Mit Hilfe der normalen Read- und Write-Request-Signale des Memory-Interfaces und durch Setzen der notwendigen Gate-Signale werden Speicherplätze beschrieben bzw. ausgelesen. Der Coprocessor hat einen uneingeschränkten Zugriff zu den BI- und BO*-Bussen, da die PACER CPU solange im S2-State (Exekution einer Instruktion) wartet, bis der Coprocessor ein Ready-Signal überträgt. Im Falle einer Ein- oder Mehradress-Information oder einer Sprungbedingung muß der P-Counter für eine richtige Rücksprungadresse modifiziert werden.

Es wurden vier verschiedene Arten von Real-Arithmetik implementiert:

- Real-Operationen mit drei Adressen, wobei zwei Adressen für die Operanden und eine für das Resultat verwendet werden. Der Floating Point Akkumulator wird dabei nicht verwendet.
- Zwei Operationen arbeiten mit zwei Adressen. Der Unterschied liegt darin, daß bei der einen Operation der Floating Point Akkumulator als Resultat, bei der anderen als Operand verwendet wird, wobei das Resultat in zwei aufeinanderfolgenden Speicherplätzen abgespeichert wird.
- Als viertes gibt es die allgemeine Floating Point Register Operation, die den Akkumulator mit einem Operanden verknüpft und das Resultat wieder in den Akkumulator speichert. Diese Operation wird in vielen Minicomputern verwendet.

Diese Adressierungsarten wurden gewählt, da sie, wie man anhand der Beispiele im Anhang sehen kann, die besten Möglichkeiten zur Berechnung arithmetischer Ausdrücke mit einer Kette von Operationen angewendet auf verschiedene Variable bieten. Es wird ein Code generiert, der die geringst mögliche Anzahl von Speicherplätzen und Instruktionen benötigt.

Für die Identifikation eines der Multiprogramming-User wird vom Betriebssystem eine spezielle Instruktion verwendet.

Die Floating Point Arithmetik verwendet State-of-the-Art Computer-Arithmetik-Techniken. Multiplikation und Postnormalisation werden durch einen sehr schnellen parallelen 24 x 24-Bit Mantissen-Multiplizierer durchgeführt, wobei die Addition der Exponenten im gleichen Microcycle erfolgt. Eine Tabelle für 1/x (wobei die Funktionswerte mit 8-Bit Genauigkeit angegeben sind) liefert den Startwert für drei Newton-Raphson Schritte, um y/x zu berechnen. Der zeitaufwendigste Teil bei einer gewöhnlichen Addition oder Subtraktion ist normalerweise die Anpassung der Exponenten, die durch ein Shiften der Mantisse der kleineren Zahl erfolgt. Der Coprocessor verwendet die Bit-Set-Hardware und Multiplizierer, um die Skalierung der Mantisse während einer Mikroinstruktion durchzuführen.

Auf Grund des Designs des Floating Point Coprocessors war es sehr einfach, den Befehlssatz des PACER 1ØØ auch für Bitmanipulationen und Integer-Transfers mit Increment oder Decrement zu erweitern. Dafür werden die Befehle '21ØØØ bis '21377 verwendet (siehe Tabelle 2).

ERROR CONDITIONS

Während jeder Exekution der neuen Befehle wird eine Anzahl von Fehlerbedingungen überprüft. Bei Auftreten eines Fehlers wird an Stelle des normalen Returns der

Mnemonics	Addresses	FUNCTION	OP-Code	
FLOAT	-	|FA| = |A|	23000	Real Conversions
INT	-	|A| = |FA|	23001	
FLA	R	|FA| = R	23002	Real Load/Store
FSTA	-"-	R = |FA|	23003	
FMV	S,R	R = S	23007	Real Transfer
NEG	-	|FA| = -|FA|	23004	Real Inversion
SFP	-	skip if |FA| ≥ 0	23005	Real Test
SFN	-	skip if |FA| < 0	23006	
ADD	S,T,R	R = S + T	23010	Real Operations with three addresses
SUB	-"-	R = S - T	23011	
MUL	-"-	R = S * T	23012	
DIV	-"-	R = S / T	23013	
RADD	S,T	|FA| = S + T	23020	Real Operations with two addresses and result in FPP-accumulator
RSUB	-"-	|FA| = S - T	23021	
RMUL	-"-	|FA| = S * T	23022	
RDIV	-"-	|FA| = S / T	23023	
ADDR	T,R	R = |FA| + T	23030	Real Operations with two addresses and FPP-accumulator as one operand
SUBR	-"-	R = |FA| - T	23031	
MULR	-"-	R = |FA| * T	23032	
DIVR	-"-	R = |FA| / T	23033	
FADD	T	|FA| = |FA| + T	23040	Real Operations with one address and FPP-accumulator as operand and destination
FSUB	-"-	|FA| = |FA| - T	23041	
FMUL	-"-	|FA| = |FA| * T	23042	
FDIV	-"-	|FA| = |FA| / T	23043	
SB a	-	$\|A\|_a$ = 1 $\quad 0 \le a \le 15_{10}$	21000-21017	Single Bit Manipulation
RB a	-	$\|A\|_a$ = 0	21020-21037	
XB a	-	$\|A\|_a = 1 - \|A\|_a$	21040-21057	
SBS a	-	skip if $\|A\|_a$ = 1	21060-21077	
IMV a	I,J	J = I + a $\quad 0 \le a \le 63_{10}$	21200-21277	Integer Transfer
DMV a	-"-	J = I - a	21300-21377	

I,J Integer
R,S,T Real

|A| Accumulator (AS-Reg.)
|FA| 32 Bit FP-Accumulator

|FA| and |A| are only changed if affected by instruction

Tabelle 2

Erweiterung des P-100 Befehlssatzes

P-Counter mit '75Ø geladen, wodurch verschiedene Error oder Exit Routinen aufgerufen werden können.

Folgende Fehler können während der Exekution eines der neuen Befehle auftreten:

1. Divide by zero
2. Exponent overflow
3. Exponent underflow
4. Operand not normalized
5. Real to integer overflow
6. Multi-indirect level violated
7. No valid op-code

Im Falle eines Fehlers wird mit einem unbedingten Sprung zur Adresse '75Ø verzweigt, wobei der P-Counter in das X-Register und ein Code für den aufgetretenen Fehler in das Q-Register des P-1ØØ abgespeichert wird. An dieser Stelle kann je nach Run-Time-Library (RTL) direkt ein System Trap oder die Verzweigung zu einer RTL-Fehlerroutine programmiert werden.

SOFTWARE-UNTERSTÜTZUNG

Der erweiterte Befehlssatz und die Besonderheiten des schnellen Arithmetik Coprocessors werden vom selbstentwickelten virtuellen Time-Sharing Betriebssystem JCS/VS 8 (siehe auch INTERFACE 17) und vom Assembler und FORTRAN-Compiler, der nur in jenem Teil modifiziert werden mußte, in dem der Floating Point Inline Assembler Output generiert wird, unterstützt. Um die Mikroprogramme für den Coprocessor zu erstellen, wurde von Dipl.Ing. A. Blauensteiner ein eigener Metaassembler geschrieben (siehe Seite 14). Mit Hilfe des neuen 32-Bit Arithmetik Coprocessors wird die Real-Arithmetik in FORTRAN-Programmen 5-6mal so schnell durchgeführt wie mit dem FPP von EAI in Verbindung mit der Floating Point Inline Option, wobei zusätzlich weniger Speicherplatz benötigt wird. Die rechenintensiven HYBSYS-Programmteile werden ca. 1Ømal so schnell laufen wie bisher und normale FORTRAN-Programme ungefähr 2Ø-3Ømal so schnell.

LITERATUR

[1] Kai Hwang: Computer Arithmetic Principles, Architecture and Design. John Wiley and Sons, New York, 1979

[2] A. Blauensteiner: User Manual JCS/VS 8. Hybridrechenanlage, Technische Universität Wien, 1981

[3] A. Blauensteiner, F. Berger: METASM User Manual. Hybridrechenanlage, Technische Universität Wien, 1981

Anhang

Beispiele für FORTRAN-Compiler generierten Code

A = B

FPPRTL

```
0       L       '.L22
1       ADR     B
2       L       '.H22
3       ADR     A
```

FPP INLINE

```
0       52      756
1       LA      *+7,2
2       DO      22
3       LA      *+5,3
4       DO      23
5       LA      757
6       DF      23
7       J       *+2
10      ADR     B
11      DI      23
12      STA     *+4,3
13      DI      22
14      STA     *+2,2
15      J       *+2
16      ADR     A
```

NEW FP-COPROCESSOR

```
0       FMV
1       ADR     B
2       ADR     A
```

A = B * C

```
0       L       '.L22
1       ADR     C
2       L       '.M22
3       ADR     B
4       L       '.H22
5       ADR     A
```

```
0       52      756
1       LA      *+7,2
2       DO      22
3       LA      *+5,3
4       DO      23
5       LA      757
6       DF      23
7       J       *+2
10      ADR     C
11      LA      *+7,2
12      DO      22
13      LA      *+5,3
14      DO      23
15      LA      754
16      DF      23
17      J       *+2
20      ADR     B
21      DI      23
22      STA     *+4,3
23      DI      22
24      STA     *+2,2
25      J       *+2
26      ADR     A
```

```
0       MUL
1       ADR     B
2       ADR     C
3       ADR     A
```

A = B * C + D

FPPRTL

```
0       L       .L22
1       ADR     C
2       L       .H22
3       ADR     B
4       L       .A22
5       ADR     D
6       L       .H22
7       ADR     A
```

FPP INLINE

```
0       SP      756
1       LA      *+7,2
2       DO      22
3       LA      *+5,3
4       DO      23
5       LA      757
6       DF      23
7       J       *+2
10      ADR     C
11      LA      *+7,2
12      DO      22
13      LA      *+5,3
14      DO      23
15      LA      754
16      DF      23
17      J       *+2
20      ADR     B
21      LA      *+7,2
22      DO      22
23      LA      *+5,3
24      DO      23
25      LA      750
26      DF      23
27      J       *+2
30      ADR     D
31      DI      23
32      STA     *+4,3
33      DI      22
34      STA     *+2,2
35      J       *+2
36      ADR     A
```

NEW FP-COPROCESSOR

```
0       RMUL
1       ADR     B
2       ADR     C
3       ADDR
4       ADR     D
5       ADR     A
```

A = B * C + D - E

FPPRTL

```
0       L       .L22
1       ADR     C
2       L       .H22
3       ADR     B
4       L       .A22
5       ADR     D
6       L       .S22
7       ADR     E
10      L       .H22
11      ADR     A
```

FPP INLINE

```
0       SP      756
1       LA      *+7,2
2       DO      22
3       LA      *+5,3
4       DO      23
5       LA      757
6       DF      23
7       J       *+2
10      ADR     C
11      LA      *+7,2
12      DO      22
13      LA      *+5,3
14      DO      23
15      LA      754
16      DF      23
17      J       *+2
20      ADR     B
21      LA      *+7,2
22      DO      22
23      LA      *+5,3
24      DO      23
25      LA      750
26      DF      23
27      J       *+2
30      ADR     D
31      LA      *+7,2
32      DO      22
33      LA      *+5,3
34      DO      23
35      LA      752
36      DF      23
37      J       *+2
40      ADR     E
41      DI      23
42      STA     *+4,3
43      DI      22
44      STA     *+2,2
45      J       *+2
46      ADR     A
```

NEW FP-COPROCESSOR

```
0       RMUL
1       ADR     B
2       ADR     C
3       FADD
4       ADR     D
5       SUBR
6       ADR     E
7       ADR     A
```

Die historische Entwicklung von digitalen und analogen Rechenmaschinen

F. Rattay

Institut für Analysis, Technische Mathematik
und Versicherungsmathematik

Obwohl bereits in der antiken griechischen Hochkultur die Mathematik als Grundwissenschaft anerkannt wurde, war der Bedarf an Rechengeräten zu wissenschaftlichen Arbeiten lange Zeit sehr gering. Das von den Römern Abacus genannte Rechenbrett existierte bereits 1100 v. Chr. und war bis in die Neuzeit verbreitetes Rechenhilfsmittel und ist sogar in Teilen Rußlands und Südostasiens auch heute noch oft in Verwendung.

Die erste Rechenmaschine, 1623 von Schickard gebaut, konnte außer Addieren und Subtrahieren auch halbautomatisch Multiplizieren und Dividieren, wurde aber, bevor sie der Erfinder Kepler übergeben wollte, durch einen Brand zerstört. Aus dieser Zeit, nämlich um 1630, stammt bereits auch der erste Analogrechner, ein Rechenschieber mit verstellbarer Zunge.

1642 ersann der junge Pascal eine Maschine zur Addition und Subtraktion, die in etlichen Varianten nachgebaut wurde.

1673 stellte Leibniz die erste vollautomatische Maschine für die vier Grundrechenarten vor, mit den wesentlichen Neuerungen der Staffelwalze und des verschiebbaren Schlittens, der durch Verschieben des Stellenwertes das rasche Multiplizieren und Dividieren ermöglicht (Abb. 2). Infolge feinmechanischer Ungenauigkeiten war diese Maschine aber nie voll funktionstüchtig. Der Bedarf bzw. die zur Verfügung stehenden Gelder waren damals jedenfalls nicht so groß, daß mit Eifer an Verbesserungen oder gar an eine Fabrikation solcher Maschinen gedacht wurde. Vielmehr wurden sie eher als Kuriosa angesehen.

Auch die 1709 von Poleni in Padua erfundene Sprossenradmaschine mit Gewichtantrieb (Abb. 3), deren Prinzip der Zahlendarstellung bis in die neuesten Entwicklungen mechanischer Rechenmaschinen beibehalten wurde, versagte wegen feinmechanischer Fehler und wurde vom Erfinder wieder zerstört. Heute existieren von allen oben erwähnten Maschinen funktionstüchtige Nachbildungen. Der Österreicher Braun baute vermutlich nach Kontakt mit Poleni die erste wirklich voll funktionierende Maschine für die vier Grundrechenarten nach dem Sprossenradprinzip (Technisches Museum Wien).

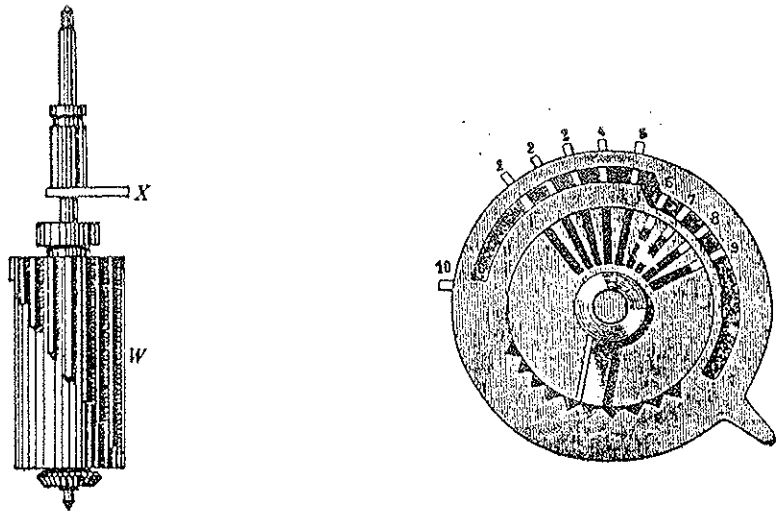


Abbildung 1

Staffelwalze (links) oder Sprossenrad (rechts) dienen als Basiselement bei praktisch allen mechanischen Rechenmaschinen. Sie erlauben bei einer vollen Umdrehung die Weiterdrehung des Zählrades um den gewünschten Wert.

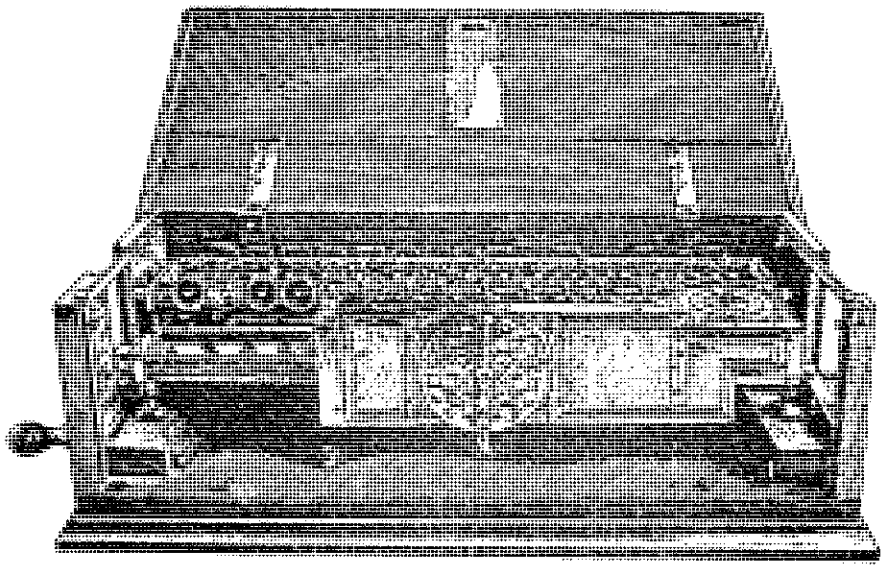


Abbildung 2

Rechenmaschine von Leibniz mit Staffelwalze und verstellbarem Schlitten.

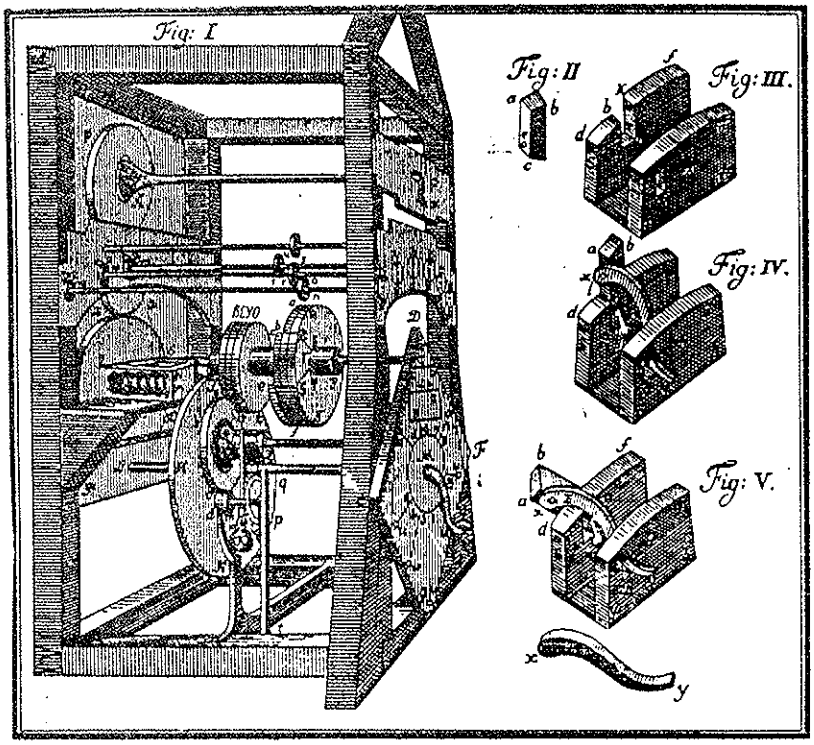


Abbildung 3

Poleni baute die erste Sprossenradmaschine.

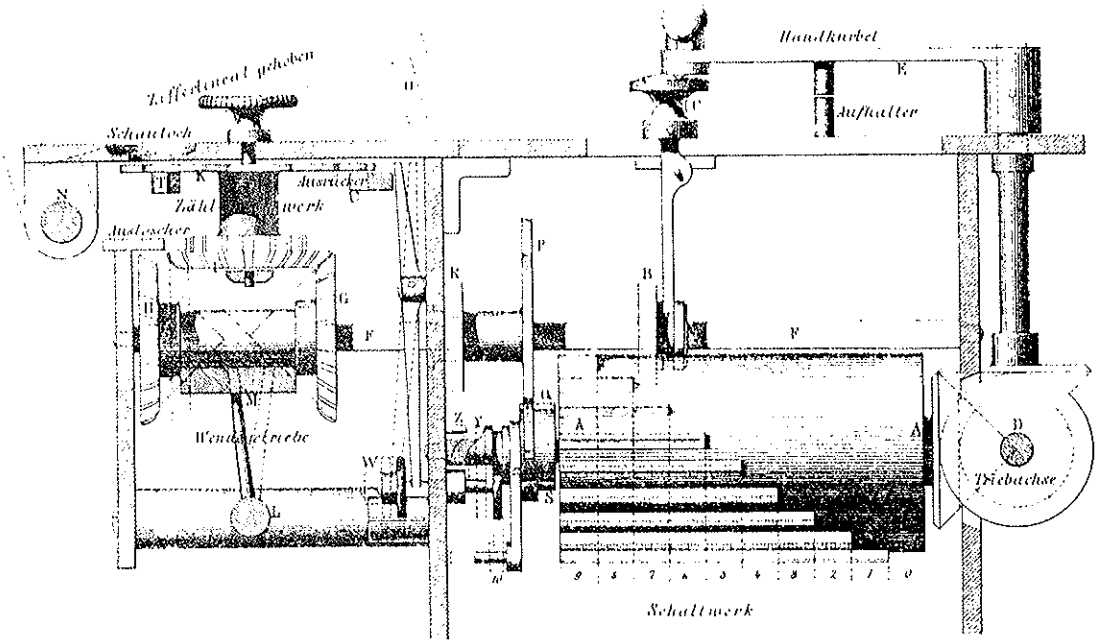


Abbildung 4

Prinzip der ersten fabriksmäßig hergestellten Rechenmaschine (Thomas).

Erst 1818-24 wurde in Frankreich die erste Fabrikation von Staffelwalzenmaschinen aufgenommen und bis 1878 wurden ca. 1500 Exemplare erzeugt (Abb. 4). 1862 wurde Babbage's Differenzenmaschine, die speziell zur Berechnung von Polynomen über höhere Differenzen zur Ermittlung von Funktionstafeln gebaut wurde, auf den internationalen Ausstellungen im Londoner Kristallpalast und in Paris gezeigt. Abgesehen von der Rechengeschwindigkeit hatten diese Rechner den Vorteil, daß in den Tafelwerken die Rechenfehler vermieden wurden. Während die Differenzenmaschine zur Auswertung von Polynomen bis zum 6. Grad herangezogen werden konnte, erlaubte Babbage's "Analytische Maschine" die flexible (programmierbare) Auswertung einfacher Algorithmen und gilt damit als Vorläufer des heutigen Digitalcomputers, der durch die 1890 bei der 11. Volkszählung in den USA verwendeten Hollerithlochkarten einen weiteren wesentlichen Fortschritt erfuhr.

Die Rechengeschwindigkeit der Computer bis zum Ende des zweiten Weltkrieges war allerdings noch recht gering und übertraf die kleiner mechanischer Tischrechner nur etwa um einen Faktor 4.

Alle bis zu diesem Zeitpunkt gebauten digitalen Rechner waren zur Lösung von Integrationsaufgaben praktisch ungeeignet, doch bestand durch den 1825 gefundenen Gonella-Integriermechanismus, der die Integration mittels eines in y-Richtung verschiebbaren Rades (Reibradgetriebe) bewerkstelligte, die Möglichkeit, graphisch vorgegebene Funktionen zu behandeln (Abb. 5). Zerlegt man die gleichförmige Drehung (unabhängige Variable) um die senkrechte Achse in gleiche Teile Δx, so entsteht an der waagrechten Achse jeweils eine Verdrehung um $y.\Delta x$ und insgesamt eine Verdrehung um $z(x) = \sum_1 y_i.\Delta x$. Für $\Delta x \to 0$ ergibt sich also bei kontinuierlicher Änderung von y: $z(x) = {}_0\int^x y\,dx$. Um die Drehbewegung ohne Schlupf zu übertragen, müssen relativ große Kräfte bzw. Reibungskoeffizienten wirken, die aber wieder ein Verändern, also eine Schiebebewegung in y-Richtung (Verschieben!) sehr erschweren. In weiteren Entwicklungen versuchte man diese Schwierigkeiten zu mindern, indem das Reibrad am Radumfang (wie etwa bei der 50 Groschen-Münze) in Richtung y gerippt wurde.

Eine wesentliche Bereicherung erhielten die mechanischen Integrierer 1876 durch die Brüder James und William Thomson, von denen letzterer unter dem Namen Lord Kelvin sehr bekannt wurde. James fand durch Zwischenschalten einer

Kugel die Möglichkeit, auch die Bewegung in y-Richtung durch Rollen zu übertragen, und verwendet anstelle der Achse mit dem Reibrad einen Zylinder mit gleicher Achsrichtung. Die Drehbewegung zwischen Scheibe und Zylinder wird nun durch eine Kugel übertragen, die durch eine Gabel in y-Richtung verschiebbar ist, wobei die Abmessungen so zu wählen sind, daß für y = 0 der Berührungspunkt der Kugel in den Scheibenmittelpunkt fällt. Die Übertragung der Kräfte erfolgt leichter, wenn die Apparatur geneigt wird, etwa so, daß die durch das Kugelgewicht verursachten Reibungskräfte zwischen Scheibe - Kugel und Kugel - Zylinder gleich sind.

Dieser Thomson'sche Integrierer wurde zunächst zur Bestimmung von $z = {}_0\int^x y\,dx$ und $z = {}_0\int^x y(x)w(x)\,dx$ benützt, wobei noch zwei zusätzliche, synchron mit x gedrehte Zylinder verwendet wurden, auf die die zu integrierende Funktion bzw. das Ergebnis aufgebracht wurden.

W. Thomson versuchte auch eine Differentialgleichung $y' = f(x,y)$ nach der Piccard'schen sukzessiven Approximation zu lösen, also mit $y_{n+1}(x) = {}_0\int^x f(x,y_n)\,dx$. Statt nun iterativ immer wieder die alten Funktionen aufzuintegrieren, erkannte er die Möglichkeit, die Steuerung y gleich vom Ergebnis abhängig zu machen und entdeckte damit das Rückkopplungsprinzip. Gegenüber dem Reibrad erbrachte der Kugelintegrierer ein wesentlich besseres mechanisches Verhalten, doch war es insbesondere bei Aufgaben mit Rückkopplungen nicht möglich, den gesamten mechanischen Antrieb über die Reibungskräfte der Kugel abzuwickeln. Es wurde vielmehr für den Integrierer ein Operator eingesetzt, der als "menschlicher Verstärker" die Gabel, die zur Kugelpositionierung diente, einem über einem Seiltrieb bewegten Zeiger folgend, nachführte. Interessanterweise haben die vier Grundprinzipien dieses ersten hundert Jahre alten Analogrechners zur Lösung von Differentialgleichungen auch heute für den elektronischen Analogrechner noch Gültigkeit:

- Für jede Rechenoperation (z.B. Integration, Multiplikation, Addition, Vorzeichenwechsel) wird ein eigenes Element verwendet, sodaß die Rechnung parallel abläuft.
- Der parallele Aufbau erlaubt die gleichzeitige Darstellung aller Variabler und deren Rückkopplung.

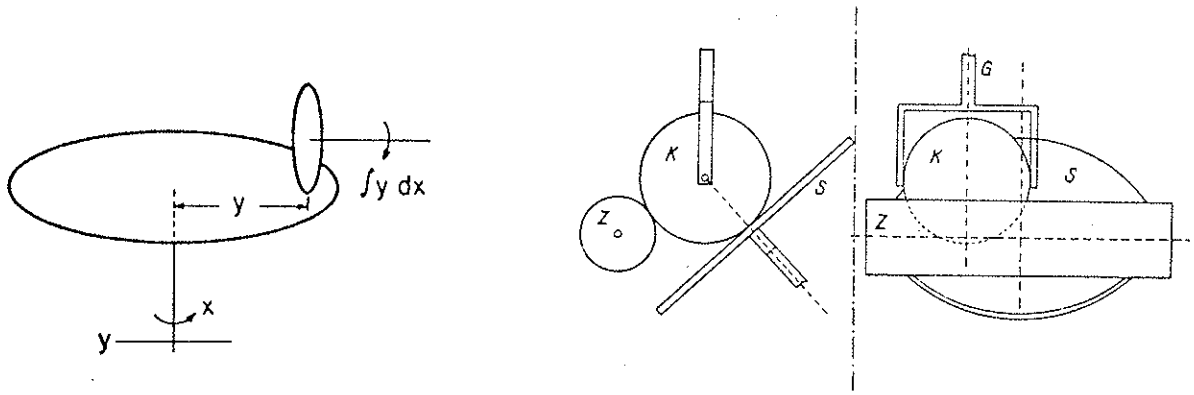


Abbildung 5

Wirkungsweise des Schneidrades (links) zur mechanischen Integration und des Thomson'schen Kugelintegrierers (rechts) zur Umgehung der Kraftreibung bei Bewegungen des Schneidrades in y-Richtung. Die Kugel K wird mittels der Gabel G im gewünschten Abstand y(x) geführt, sodaß von der mit x gedrehten Scheibe S durch K die Verdrehung des Zylinders Z um ∫ydx bewirkt wird. Bei gleichmäßiger Drehung werden somit Integrale nach der Unabhängigen x=t gebildet.

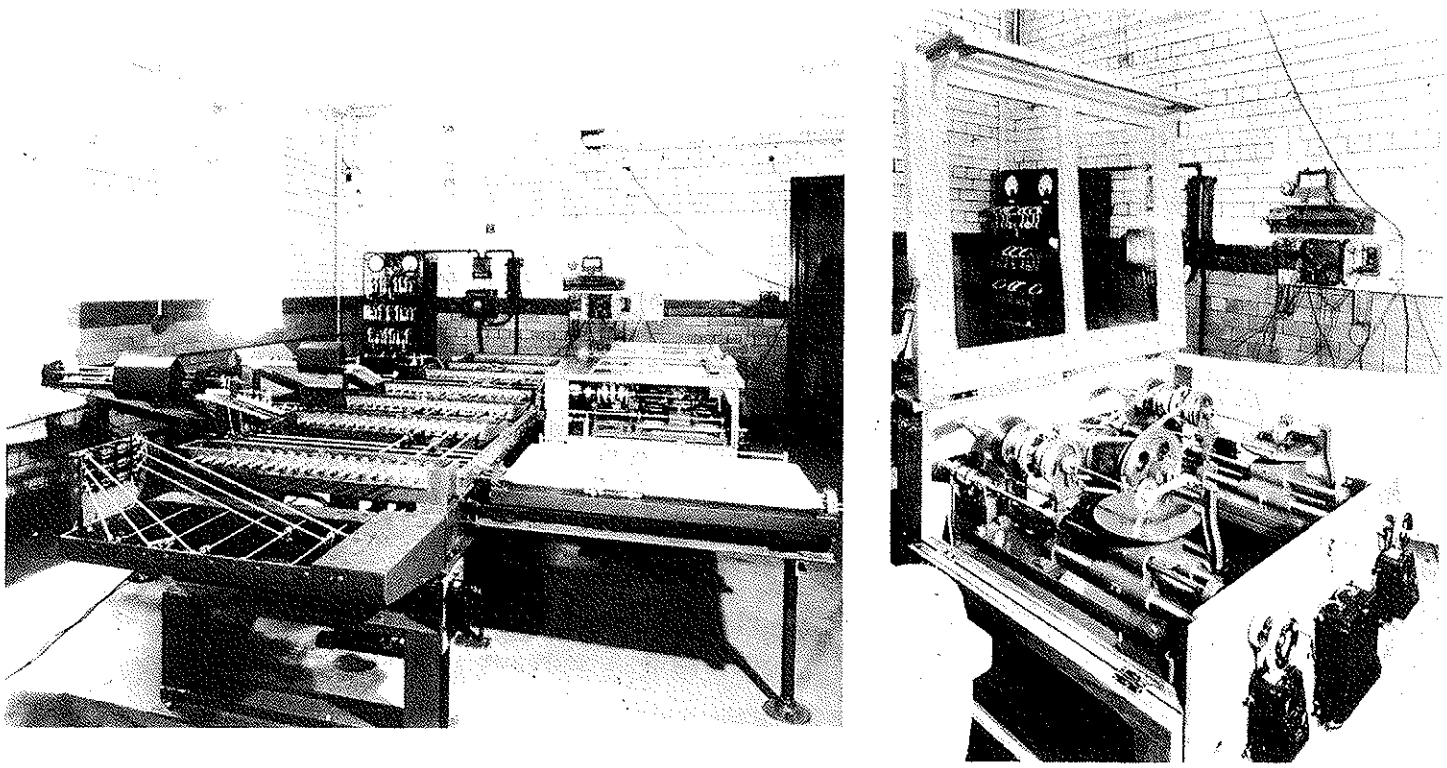


Abbildung 6

Mechanische Integrieranlage (Analogrechner) von Bush mit elektromechanischen Drehverstärkern. Rechts der Schneidradintegrierer im Detail. (Die Bilder stammen aus dem Science Museum in London.)

- Durch die geometrische Beschränkung der Abmessungen jedes einzelnen Rechenelements ist in direkter Weise eine Beschränkung der Variablen gegeben.
- Die Genauigkeit ist durch die physikalischen Gegebenheiten beschränkt und läßt sich auch mit großem Aufwand nicht beliebig steigern.

Unter Thomson wurde eine Anlage mit 9 Integrierern zur Gezeitenvoraussage gebaut und danach noch einige Analogrechner nach dem Reibrad- oder Kugelverfahren, wie etwa die Anlage von Kriloff für Differentialgleichungen 4. Ordnung, die im ersten Weltkrieg, noch unvollendet, verlorenging, sowie der Fahrdiagraph von Knorr (1914 für Differentialgleichungen 2. Ordnung).

Die erste Anlage von praktischer Bedeutung, die neben 6 Integrierern noch Funktions-, Multiplikations-, Additions-, Übersetzungs- und Ergebnisgetriebe aufwies und durch den Einsatz mechanischer Rotationsverstärker Genauigkeiten von 2-3% erbrachte, wurde 1920-30 von Bush gebaut.

Bis 1945 wurden nach diesem mechanischen Prinzip noch etliche Anlagen errichtet, wie etwa der mit 12 Integrierern bestückte Rechner in Oslo, dessen Achsen auf 0.006 mm genau gearbeitet waren, oder die große Anlage von Bush mit 18 Integrierern, 2000 Röhren und 150 Elektromotoren (Abb. 6).

Literatur:

Goldstine H.H.: The computer from Pascal to von Neumann, Princeton University Press, 1973

Willers F.: Mathematische Maschinen und Instrumente, Berlin, Akad. Verlag, 1951

SIMULATION DES SCHLEUSENFÜLLVORGANGES

Dipl.Ing.Dr. Gotthard Kayser
Inst. für Konstruktiven Wasserbau
Technische Universität Wien

1. EINLEITUNG

Zur Überwindung von Höhendifferenzen des Wasserspiegels an einer Wasserstraße werden bei Schiffahrtskanälen bzw. bei Staustufen eines ausgebauten Flusses Schiffsschleusen angeordnet.Beim Füllvorgang tritt das Wasser durch Umlaufkanäle vom Oberwasser in die Schleusenkammer ein, beim Entleerungsvorgang wird es in ähnlicher Weise in das Unterwasser abgelassen. Die Steuerung dieser Vorgänge erfolgt durch langsames Öffnen bzw. Schließen eines Regelungsverschlusses in den Umlaufkanälen.

2. HYDROMECHANISCHE BERECHNUNGSGRUNDLAGEN

Das hydraulische Grundproblem läßt sich vereinfacht auf die Füllung eines Behälters - über einen Verbindungskanal mit veränderlichem Querschnitt - aus einem zweiten Behälter zurückführen. Der instationäre Strömungsvorgang ist ein Ausfluß unter Druck, der bestimmt wird durch

- den Druckhöhenunterschied der verschieden hohen Wasserspiegellagen innerhalb und außerhalb der Schleusenkammer,
- den je nach Stellung unterschiedlichen Widerstand des Regelungsorganes im Verbindungskanal [1], [4].

3. MATHEMATISCHES MODELL

Mit den Bezeichnungen aus Abb. 1 läßt sich folgendes Gleichungssystem aufstellen:

$$\frac{dX}{dT} = -\frac{K}{D}.V$$

$$\frac{dZ}{dT} = +\frac{K}{F}.V$$

$$\frac{dV}{dT} = -\left[\frac{V|V|}{2M}(1+\varepsilon_i+\varepsilon_v)+\frac{g}{L_n.M}(Z-X)\right]$$

$$\frac{dV_r}{dT} = -\frac{1}{R}\left[\frac{V_r|V_r|}{2M}(R^2.(1+\varepsilon_i)+\varepsilon_r)+\frac{g}{L_n.M}(Z-X)+\frac{dR}{dT}.V_r\right]$$

$$V_r = \frac{V}{R}$$

$$\varepsilon_r = R^2.\varepsilon_v$$

$$\varepsilon_v = \left(\frac{1}{\psi.R}-1\right)^2$$

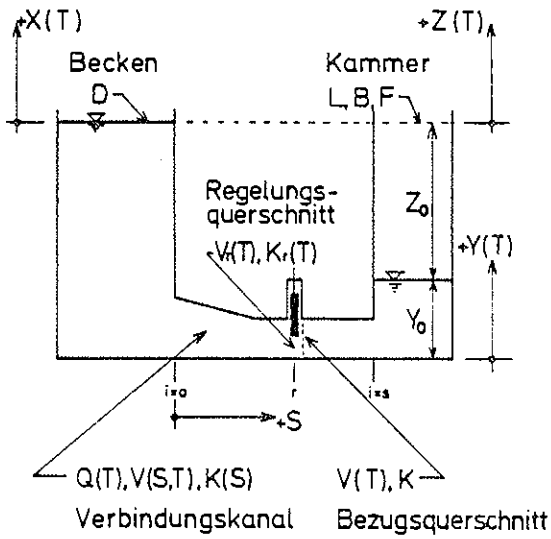


Abbildung 1

Das Gleichungssystem ist mit L_n als Bezugsgröße normiert

L . . . Länge der Schleusenkammer
B . . . Breite der Schleusenkammer
F . . . Grundrißfläche der Schleusenkammer
D . . . Grundrißfläche des Beckens
V . . . Fließgeschwindigkeit im Verbindungskanal
K . . . Querschnittsfläche des Verbindungskanals
Q . . . Durchfluß im Verbindungskanal
T . . . Zeit
S . . . Koordinate des Verbindungskanals
X . . . Koordinate des Beckenwasserspiegels
Y . . . Koordinate der Wassertiefe in der Kammer
Z . . . Koordinate des Kammerwasserspiegels
r . . . Index des Regelungsquerschnittes
D . . . Index der Zeit zu Schleusungsbeginn
R . . . Öffnungsgrad des Regelungsorganes
M . . . maßgebende Verbindungskanallänge
ε_i . . . konstanter Verlustbeiwert
ε_r . . . Verlustbeiwert des Regelungsorganes bezogen auf V_r
ε_v . . . Regelungsbeiwert
g . . . Erdbeschleunigung
ψ . . . Strahlkontraktion
C_o, C_1, C_2, C_3 Faktoren des Regelungsbeiwertes ε_r
ε_v . . . Verlustbeiwert des Regelungsorganes bezogen auf V

Das Öffnungsgesetz R(T) des Regelungsorganes wird durch Integration der Öffnungsgeschwindigkeit U(T) erhalten.

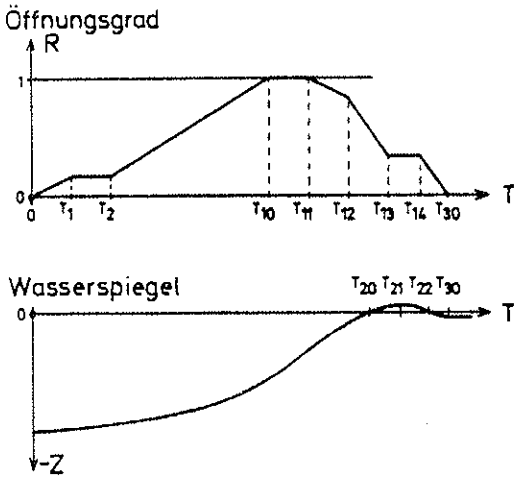


Abbildung 2

Das Gleichungssystem beschreibt einen Schwingungsvorgang des Wasserspiegels. Die quadratische Dämpfung weist neben dem konstanten Anteil ε_i noch einen zeitabhängigen Anteil ε_v bzw. ε_r durch die Stellung des Regelungsorganes auf.

Zu beachten ist besonders der Grenzfall des geschlossenen Regelungsorganes (R = Ø), bei dem der Verlustbeiwert $\varepsilon_v = \infty$ und die Strömungsgeschwindigkeit V = Ø wird.

4. MODELLEICHUNG

Um die Brauchbarkeit des entwickelten mathematischen Modells sicherzustellen, ist eine Eichung durch Vergleich mit einer ausgeführten Schleusenanlage notwendig.

Für einen Füllvorgang der Südschleuse Aschach ist das Öffnungsgesetz R(T) gegeben und die gemessenen Werte der Ganglinie des Schleusenwasserspiegels bekannt. Diese stellen die Sollwerte der Eichkurve $Z_s(T)$ dar. Durch Variation von Eichparametern soll die mit dem vorgegebenen Öffnungsgesetz berechnete Ganglinie Z(T) der Eichkurve möglichst genau angepaßt werden.

Für die Strahlkontraktion wird

$\psi = C_o + C_1 \cdot R + C_2 R^2 + C_3 R^3$ mit der Nebenbedingung

$C_o + C_1 + C_2 + C_3 = 1$ angesetzt.

Als Eichparameter treten folgende Größen auf:

- M . . . die maßgebende Verbindungskanallänge
- ε_i . . . der konstante Verlustbeiwert
- C_o, C_1, C_2 . . . die Faktoren des Regelungsbeiwertes ε_r

Als Ausgangswerte müssen für diese fünf Parameter möglichst gut geschätzte Näherungswerte eingegeben werden.

Das Problem der Eichung wird auf ein Optimierungsproblem zurückgeführt, bei dem die Abweichung der Ganglinie Z (T) von der Eichkurve $Z_s(T)$ minimiert werden soll.

Um eine möglichst gute Übereinstimmung der beiden Funktionen Z_s und Z zu erreichen, wird das Optimierungskriterium formuliert als

$$J = \int_{T_o}^{T_{33}} | Z_s - Z | dT$$

Als Nebenbedingungen soll zu ausgewählten Zeitpunkten $T_i = (T_1, T_{11}, T_{33})$ gelten: $N_i = Z_s - Z = Ø$.

Das vorliegende Optimierungsproblem ist ein Parameterproblem, da die Struktur des optimalen Steuerungsgesetzes, das nur durch unbekannte Parameter bestimmt wird, bekannt ist. Im Prinzip können die vorliegenden Differentialgleichungen unter Beachtung der Nebenbedingungen mit den gegebenen Anfangsbedingungen gelöst werden.

4.1 Die Lösung des Eichungsproblemes am Hybridrechner

Das Gleichungssystem wird am Hybridrechner mit Hilfe des Hybrid-Simulations-Systems (HYBSYS) simuliert. Die Differentialgleichungen können unter Beachtung der gegebenen Anfangsbedingungen am Analogteil des Hybridrechners gelöst werden. Die Nebenbedingungen der Gleichungen jedoch können nicht direkt erfüllt werden.

Eine Möglichkeit zur Lösung bietet aber der HYBSYS-Overlay ZERO, der an der Hybridrechenanlage zur Verfügung steht. Er berechnet die Lösung eines Sytems nichtlinearer Gleichungen. Dieses Verfahren - von M.J.D. Powell entwickelt - stellt eine Kombination der Newton-Raphson-Iteration mit der Methode des steilsten Anstiegs dar. Die Lösung erfolgt durch iterative Bestimmung einer Nullstelle ohne explizite Verwendung der Funktionalmatrix durch Minimierung der quadratischen Norm

$$\| \underline{e}(\underline{v}) \|_2 = \sqrt{e_1(\underline{v})^2 + \ldots + e_i(\underline{v})^2} = \min$$

Bei der hier vorliegenden Eichung sind die Komponenten des Vektors $\underline{e}$ die Nebenbedingungen N_i und das Optimierungskriterium J. Durch den HYBSYS-Overlay ZERO werden die Abweichungen der beiden Funktionen minimiert, wobei durch einen Faktor die Gewichtung der einzelnen Komponenten bestimmt wird. Die Nebenbedingungen N_i werden somit durch Minimierung der Abweichung erfüllt, das Optimierungskriterium dürch Minimierung des Funktionals J.

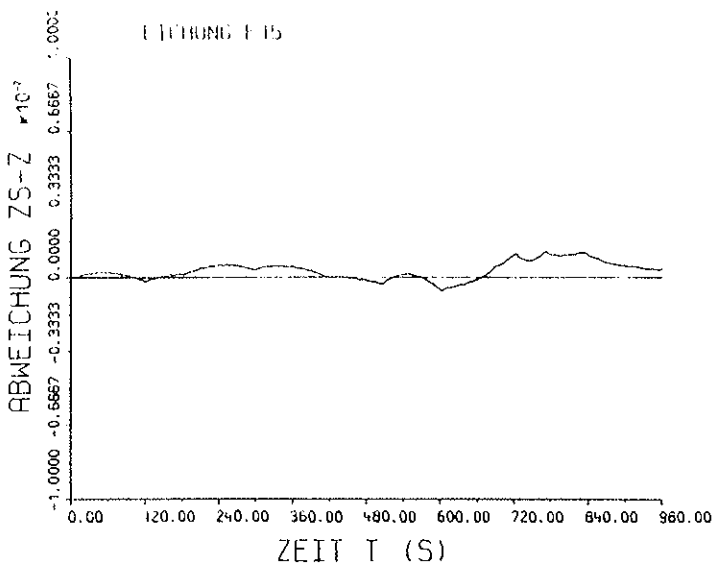


Abbildung 3

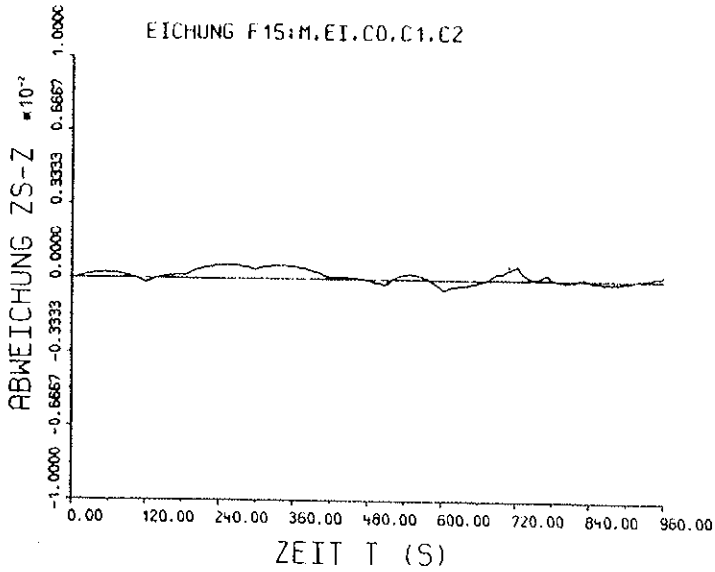


Abbildung 4

4.2 Beispiel einer Eichung

Als praktisches Beispiel wird die Eichung anhand eines Füllvorganges gezeigt, für den Messungen an der Südschleuse des Donaukraftwerkes Aschach vorliegen. Die Grundlagen für den Eichungsvorgang sind das Öffnungsgesetz R(T) und die Sollfüllkurve Z_s. Aus den Anlageverhältnissen ergeben sich folgende Festwerte:

L_n = 255.50 m
B = 24.00 m
K = 32.00 m^2

Die Ausgangswerte für folgende Parameter wurden näherungsweise angenommen:

M = 0.55
ε_i = 0.40
C_0 = 0.63
C_1 = 0.00
C_2 = 0.00

Der Zustand vor der Eichung ist charakterisiert durch die Abweichung der beiden Füllkurven (Z_s - Z), Abb.3.

Die Unstetigkeiten ergeben sich aus der punktweisen Eingabe der Sollfüllkurve Z_s über einen Funktionsgeber mit 16 Stützstellen.

Die Abweichung beträgt maximal 0.0016 L_n = 0.40m, das ist 3 % der Gesamthubhöhe von 13.00 m.

4.3 Ergebnisse der Eichung

Die Eichungsergebnisse zeigen eine gute Übereinstimmung der beiden Funktionen Z_s und Z. Die Verbesserung ist aus dem Vergleich der Abweichung vor der Eichung (Abb.3) und nach der Eichung (Abb.4) zu ersehen. Die Eichergebnisse der Parameter sind:

M = 0.94
ε_i = 0.31
C_0 = 0.63
C_1 = 0.01
C_2 = 0.01

Im besonders wichtigen Zeitraum der Ausspiegelung, wenn Z = Ø ist, ist eine sehr gute Übereinstimmung erreicht. Da die Ermittlung der Füllzeit und das Schwingen des Wasserspiegels um die Nullage besonders wichtig ist, kann mit dem aufgestellten Modell ein sehr gutes Ergebnis erreicht werden (Abweichung 0.00023 L_n = 0.06 m, das ist 0.5 % der Hubhöhe).

5. OPTIMIERUNG DER FÜLLVERSCHLUSZSTEUERUNG

Wartezeiten in den Vorhäfen von Schleusenanlagen und der Schleusungsvorgang selbst bedeuten immer einen Zeitverlust für den Schiffahrtsbetrieb. Daher wird angestrebt, möglichst wenig Staustufen mit großen Hubhöhen der einzelnen Schleuse zu bauen. Je größer die Hubhöhe bei einer Schleusenanlage ist, umso schwieriger wird die Einhaltung der Anforderungen, die an eine Schleuse gestellt werden, wie zum Beispiel

- kurze Füll- und Entleerungszeiten,
- ruhige Lage der Schiffe in der Schleusenkammer,
- Vermeidung des Überschwingens, bei dem die bewegten Wassermassen den Schleusenwasserspiegel noch über die Ausspiegelungshöhe schwingen lassen.

Außer durch die bauliche Ausführung des Füllsystems kann der Füllvorgang durch die Steuerung des Regelungsorganes beeinflußt werden.

5.1 Aufgabenstellung

Für eine gegebene Schleusenanlage soll die kürzeste Füllzeit unter Ausschaltung des Überschwingens bei einer möglichst ruhigen Schiffslage ermittelt werden.

Das Überschwingen wird durch Schließen des Regelungsorganes vermieden, wobei der vollständige Abschluß genau zum Zeitpunkt des Ausspiegelns erfolgen muß.

Die ruhige Schiffslage wird durch eine Vielzahl von nur ungenau zu erfassenden Einflüssen bestimmt.

Durch die Beschränkung der Maximalwerte verschiedener hydromechanischer Parameter, welche die Schiffsbewegungen charakterisieren, kann die erwünschte ruhige Schiffslage erreicht werden [3], [5].

Die Untersuchung erfolgt für Beschränkungen der Wasserspiegelneigung I_p, der Zuflußänderung A_p oder des Zuflusses V_p.

5.2 Die Zielfunktion

Gesucht ist eine Steuerung, bei der zu einem möglichst frühen Zeitpunkt (T_{30}) das Regelungsorgan geschlossen ist (R_{30}=Ø) und der Schleusenwasserspiegel ohne Überschwingen ausgeglichen ist (Z_{30}=Ø). Dabei ist außerdem als Nebenbedingung die Beschränkung des Maximalwertes der Parameter I_p, A_p oder V_p, und der Schließgeschwindigkeit U_{11} einzuhalten.

Als Steuergröße tritt der Zeitpunkt für den Beginn des Schließvorganges T_{11} auf.

Das Optimierungsproblem ist wieder ein Parameterproblem, da die Struktur des optimalen Steuerungsgesetzes bekannt ist, das nur durch den noch unbekannten Parameter T_{11} bestimmt wird.

Die Zielfunktion, die minimiert werden soll, ist daher

$$J = Z_{30}$$

Die Forderung, daß auch T_{30} ein Minimum werden soll, muß indirekt durch einen geeigneten, möglichst kleinen Startwert von T_{11} erfüllt werden.

5.3 Die Lösung des Optimierungsproblemes am Hybridrechner

Die Zustandsdifferentialgleichungen können mit den gegebenen Anfangsbedingungen unter Einhaltung aller Nebenbedingungen durch eine entsprechende Schaltung am Analogrechner gelöst werden. Entsprechend der Vorgangsweise bei der Eichung, erfolgt die Lösung der Optimierungsaufgabe wieder durch den Overlay ZERO. Dabei reduzieren sich die Komponenten des Vektors g auf das Optimierungskriterium J.

Bei der Simulation des Öffnungsgesetzes R(T) am Hybridrechner besteht die Möglichkeit, durch Vergleich mit vorgegebenen Grenzwerten von Parametern eine Regelung des Öffnungsvorganges durch direkte Rückkoppelung in der Schaltung zu erreichen. Das gesuchte Öffnungsgesetz ergibt sich aus dem Verlauf des Rechenvorganges. Am Analogteil des Rechners wird diese Regelungsschaltung aufgebaut. Dabei wird eine "bang-bang-Schaltung" verwendet, bei der die Regelungsgröße dR/dT nur die zwei Grenzwerte Ø bzw. die vorgegebene Öffnungsgeschwindigkeit U annimmt, jedoch keine Zwischenwerte.

5.4 Beispiel einer Optimierung mit Beschränkung der Wasserspiegelneigung I

Als Ausgangsbasis dient wieder der Füllvorgang F-15 der Südschleuse des Donaukraftwerkes Aschach, für den auch die Eichung untersucht wurde.

$$T_{20} = 760 \text{ s}$$
$$\max|I| = 0.00047$$
$$\max Q = 12.4\ 10^{-6}$$
$$Q_m = 6.3\ 10^{-6} \qquad \text{(Abb. 5)}$$

Um das Überschwingen auszuschalten wird das Regelungsorgan vollständig geschlossen.

$$T_{30} = 813 \text{ s}$$
$$\max Q = 12.3\ 10^{-6}$$
$$Q_m = 5.9\ 10^{-6} \qquad \text{(Abb. 6)}$$

Wenn der ursprüngliche Maximalwert der Wasserspiegelneigung I = 0.00047 eingehalten wird, jedoch die maximale Öffnungsgeschwindigkeit U_o schon zum Beginn des Schleusungsvorganges zugelassen wird (U_o = 0.003041), dann wirkt sich die Beschränkung der maximalen Wasserspiegelneigung schon aus: Das Öffnungsgesetz zeigt einen nichtlinearen Verlauf. Die charakteristischen Werte werden:

$$T_{30} = 605 \text{ s} \qquad (-20\%)$$
$$\max Q = 15.1\ 10^{-6} \qquad (+22\%)$$
$$Q_m = 7.9\ 10^{-6} \qquad (+25\%)$$

Die Füllzeit verringert sich, maximaler und auch mittlerer Zufluß steigen (Abb. 7).

Bei Einhaltung der ursprünglichen Füllzeit T_{30} = 760 s und Verringerung des Parameters I_p für die Beschränkung der Wasserspiegelneigung ergeben sich folgende Werte:

$$I_p = 0.00030 \qquad (-36\%)$$
$$\max Q = 12.1\ 10^{-6} \qquad (-2\%)$$
$$Q_m = 6.3\ 10^{-6}$$

Bei sonst gleichbleibenden Werten kann die Wasserspiegelneigung als Gütekriterium für die Schiffskräfte um 36% verbessert werden (Abb. 8).

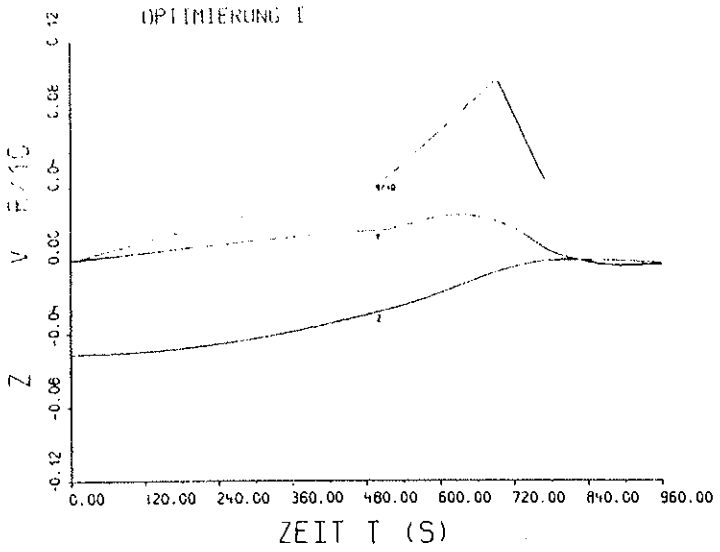


Abbildung 5

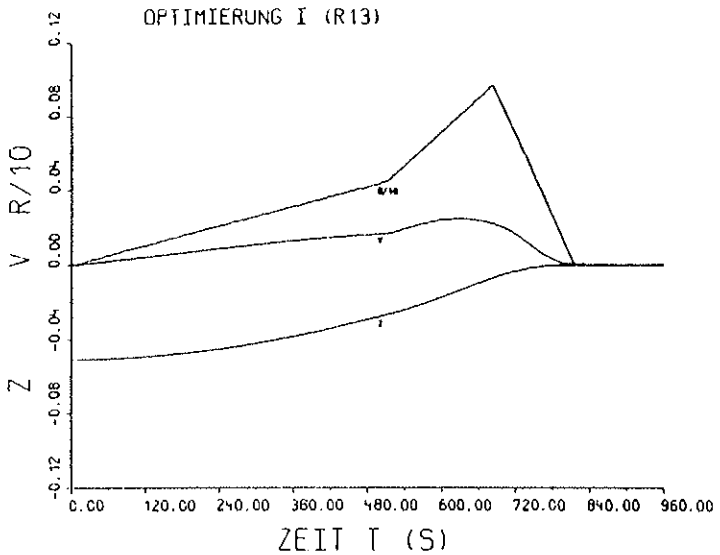


Abbildung 6

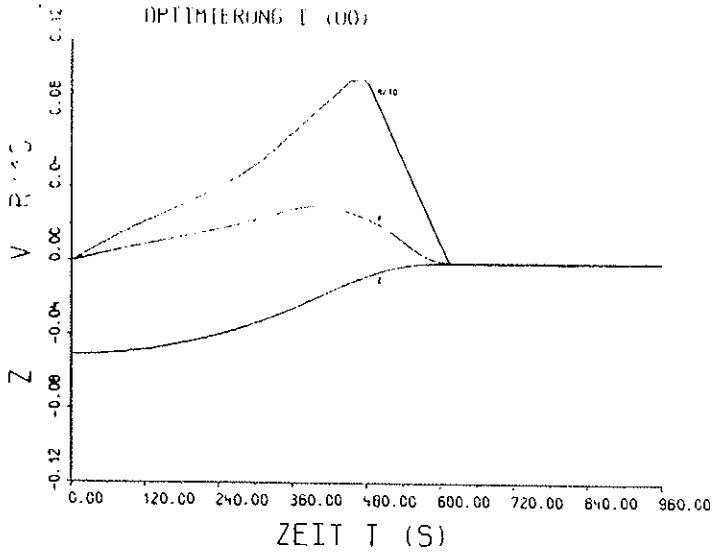


Abbildung 7

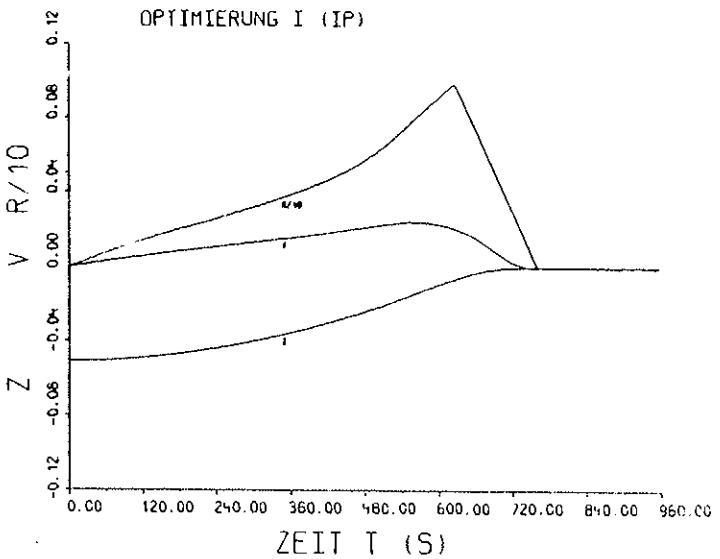


Abbildung 8

5.5 Ergebnisse der Optimierung

Die Ausschaltung des Überschwingens durch vollständiges Schließen des Regelungsorganes führt bei ungeänderten sonstigen Bedingungen zu einer Verlängerung der Füllzeit, eine Vergrößerung der Öffnungs- bzw. Schließgeschwindigkeit des Regelungsorganes zu einer Verkürzung der Füllzeit auch bei Einhaltung derselben Parameterwerte für die Beschränkungen. Bei Einhaltung derselben Füllzeit können viel niedrigere Parameterwerte für die Beschränkungen und damit eine viel ruhigere Schiffslage erzielt werden.

Charakteristische Öffnungsgesetze für das Regelungsorgan ergeben sich bei Berücksichtigung verschiedener Parameterwerte für die Beschränkungen:

- "I_p" (Wasserspiegelneigung) Abb. 9
- "A_p" (Zuflußänderung) Abb. 10
- "V_p" (Zufluß) Abb. 11

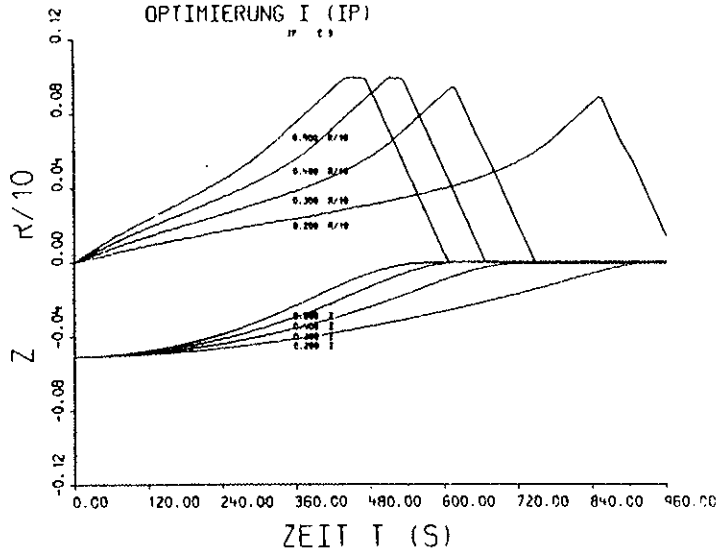


Abbildung 9

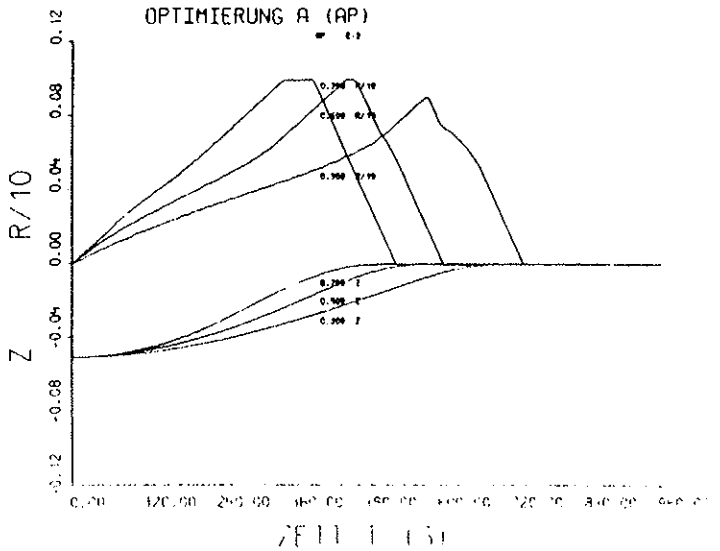


Abbildung 10

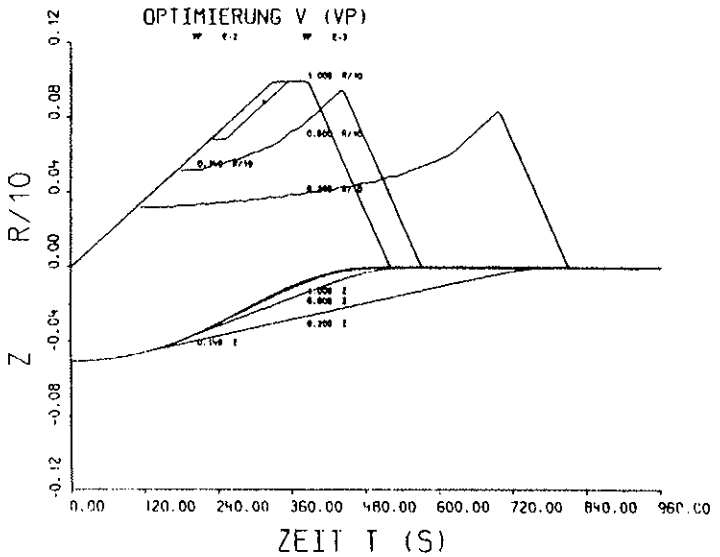


Abbildung 11

6. SCHLUSZFOLGERUNGEN

Durch eine nichtlineare Steuerung des Regelungsorganes im Füll- bzw. Entleerungssystem einer Schleuse kann der Schleusungsvorgang verbessert werden. Die Verbesserung kann eine Verkürzung der Füllzeit oder eine Verringerung der zulässigen Maximalwerte von Parametern sein, die als Gütekriterien eine ruhige Lage der Schiffe in der Schleusenkammer garantieren sollen. Das vorliegende mathematische Modell ermöglicht die Berechnung einer optimalen Steuerung; dabei können jedoch gewisse Ungenauigkeiten nicht ausgeschaltet werden, weil eine vollkommen exakte Zuordnung der Parameterwerte zu den Gütekriterien nicht möglich ist.

SUMMARY

In planning ship locks of increasing size and lifting height difficulties arise from the search of a suitable design considering short filling time and also reducing unfavourable inertia forces of the ship.

By a mathematical model the effect of parameters of quality which limit unfavourable inertia forces and ship motions on several hydromechanical factors is investigated.

The set of equations is solved by the use of a hybrid computer system (HYBSYS). The verification of the mathematical model is given for an existing ship lock.

To avoid the effect of over-oscillation - i.e. the rise of the water level in the lock above the up-stream water level - optimal control of the filling valve operation is calculated. Characteristic graphs of the resulting nonlinear control are given for different parameters of quality. In that way either the filling time or the unfavourable inertia forces of the ship can be reduced.

LITERATUR

[1] DIETRICHS, E.: Verfahren zur Berechnung der Schleusenfüllung vom Oberhaupt aus, Wasserwirtschaft - Wassertechnik 19 (1969), 200-203

[2] KAYSER, G.: Nichtlineare Steuerung des Schleusenfüllvorganges, Dissertation der TU Wien, 1981

[3] MOSONYI, E. und BAKOWIES, F.: Modellversuche für Schiffsschleusen, Zeitschrift für Binnenschiffahrt und Wasserstraßen, (1975), 451 - 458

[4] PRESS, H. und SCHRÖDER, R.: Hydromechanik im Wasserbau, Berlin/München: W. Ernst, 1966

[5] WICKERT, G.: Abhängigkeit und Größe der Schiffskräfte bei Kammerfüllung durch das Obertor mit tiefliegendem Drempel, Dissertation der TH Karlsruhe, 1951

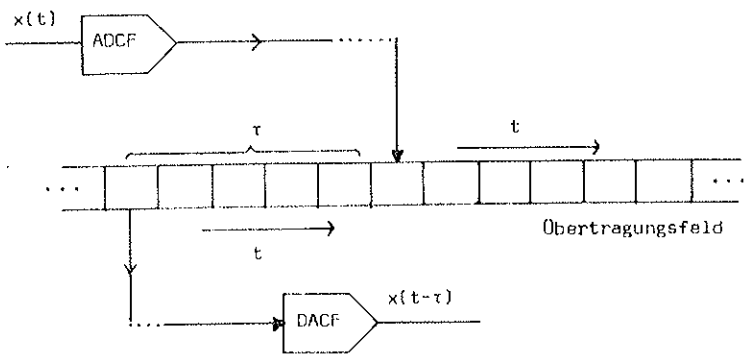
HYBSYS-OVERLAYS ZUR ZEITVERZÖGERUNG

Praktikum für Hybridrechner I (P. Götzl)
F. Breitenecker
Institut für Analysis, Technische Mathematik und Versicherungsmathematik
Arbeitsbereich Regelungstheorie und Hybridrechentechnik (Prof.I. Troch)
Technische Universität Wien

Es wurden drei zusammengehörige Overlays - INDEL, INDELF, DELEY - entwickelt und installiert, die es ermöglichen, eine oder mehrere abhängige Variable einer HYBSYS-Schaltung um eine Zeit τ zu verzögern. Diese Overlays stehen allen Benützern auf District 19 zur Verfügung.

FUNKTIONSWEISE

A/D- und D/A-Übertragung der zu verzögernden (k) Funktion(en) mit Einspeicherung bzw. Entnahme der Funktionswerte an verschiedenen Plätzen desselben Übertragungsfeldes.



Die maximale Anzahl N der Diskretisierungspunkte errechnet sich daher folgendermaßen:

N= Min {(2ØØ2)/K,(TAU+TEND-TØ).BETA.1ØØØ)/MINDIS}

MINDIS= Mindestzeitabstand zwischen zwei Diskretisierungspunkten in µsec (mit 4ØØ implementiert).

Demnach ergeben sich für die Verzögerungszeit τ N_V= N.TAU/(TAU+TEND-TØ) Diskretisierungspunkte. Im Intervall [TØ-TAU,TØ] müssen entweder diese N_V Punkte vorgegeben oder durch Angabe einer Funktion initialisiert werden.

VERWENDUNG

Der Overlay INDEL initialisiert die Zeitverzögerung. Der Aufruf

```
INDEL:X1AD,X2AD BY Ø.8 ↵
```

initialisiert z.B. die Verzögerung der Variablen X1 und X2 um TAU=Ø.8.

Dabei müssen in der Schaltung zu X1,X2 die ADCF-Variablen X1AD,X2AD (zu verzögernde Funktionen) und die DACF-Variablen X1DA, X2DA (verzögerte Funktionen) mit entsprechenden Verbindungen zusätzlich definiert werden. INDEL überprüft die Richtigkeit der Zuordnungen X1-X1AD-X1DA und X2-X2AD-X2DA. (Eine Fehlermeldung erfolgt auch, wenn ADCF-Variable und DACF-Variable in unterschiedlicher Reihenfolge definiert wurden.)

INDEL fordert als erstes den Aufruf des Overlays INDELF. (Aus Speicherplatzgründen mußte diese Teilung erfolgen.)

```
INDELF ↵
```

Der Overlay gibt die Anzahl der Diskretisierungspunkte N_V in [TØ-TAU,TØ] aus und verlangt für jede einzelne zu verzögernde Funktion die Eingabe der Vorgabeart in [TØ-TAU,TØ].
Die Frage kann mit P↵, F↵ oder BRK↵ beantwortet werden:

a) P↵ (Points): X1 wird punkteweise vorgegeben; der Reihe nach werden zur Nummer des Diskretisierungspunktes die Funktionswerte vorgegeben:

```
z.B.:
1. Punkt: 4.7 ↵
2. Punkt: 5.1 ↵
   .
   .
```

b) F↵ (Function): X1 wird als Funktion vorgegeben. Es ist der Index der Funktion vorzugeben (siehe HYBSYS-Manual Kap. 11.5) und die Funktionsparameter A,B,C,D.

```
z.B.:
INDEX:5 ↵
A,B,C,D:
7.2, 8.1, Ø., 1.↵
```

c) BRK↵ (Breaktaste): Die Initialisierung wird abgebrochen (nützlich, wenn N_V zu klein ist, was durch Änderung von BETA berichtigt werden kann).

Wird INDELF ohne Fehlermeldung abgebrochen (es existieren 24 selbstdokumentierende Fehlermeldungen), so kann nun beliebig oft ein (hybrider) Rechenlauf durch Aufruf des Overlays DELEY*)

```
DELEY ↵
```

durchgeführt werden, wobei der Rechner vorher in HYBRID-Mode zu setzen ist.

---

*)Der Name beinhaltet keinen orthographischen Fehler; ein Systemoverlay namens DELAY existiert bereits, weswegen ein anderer Name gewählt werden mußte.

BEMERKUNGEN

1) Für X1 und X1DA müssen geeignete (üblicherweise gleiche) Skalierungsfaktoren eingegeben werden; INDELF bricht ab, wenn die (skalierte) Vorgabefunktion zu groß ist.

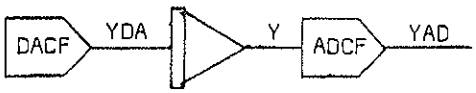
2) Mit DELEY! kann eine Art REPOP-Mode erzeugt werden (vgl. SINGLE!).

3) Jede beliebige Variable der Schaltung kann mit PLOT-Befehlen über [TØ,TEND] gezeichnet werden (HYBRID-Mode!).
X1DA wird durch PLOT X1DA in [TØ-TAU,TEND-TAU] gezeichnet, obwohl die Achsenbeschriftung sich auf [TØ,TEND] bezieht (Da das Plotten einer DACF-Variablen keinen hybriden Lauf startet, muß vorher mindestens einmal DELEY aufgerufen werden.).

4) Da derzeit nur zwei ADCF-Makros zur Verfügung stehen, können nur zwei Funktionen einer Schaltung verzögert werden.

BEISPIEL

$y' = y(t-\tau)$



Nach Deklaration der Schaltung (TØ=Ø,TEND=2) wird die Zeitverzögerung mit

```
INDEL:YAD BY Ø.4 ↵
```

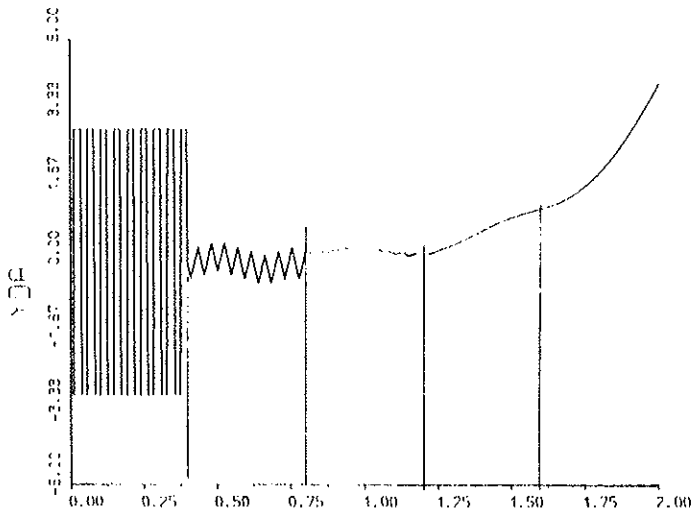
initialisiert.
Die Fortsetzung mit INDELF legt y(t) auf [-τ,Ø] mit

```
INDEX:3 ↵
A,B,C,D:
3.,1ØØ.,Ø.Ø,Ø.Ø ↵
```

fest. Die Befehlsfolge

```
HYBRID; DELEY; PLOT YDA
```

zeichnet y(t) auf [-τ,2-τ].



INTERFACE Jänner 1982