

Ansätze zu funktionsorientierten Prozessrechnerstrukturen

DISSERTATION
zur Erlangung des Grades des
Doktors der Naturwissenschaften
der Universität Dortmund
an der Abteilung Informatik

von
Wolfgang A. Halang

Dortmund
1980

Ansätze zu funktionsorientierten Prozessrechnerstrukturen

**DISSERTATION
zur Erlangung des Grades des
Doktors der Naturwissenschaften
der Universität Dortmund
an der Abteilung Informatik**

**von
Wolfgang A. Halang**

**Dortmund
1980**

Tag der mündlichen Prüfung: 25.4.1980

Dekan: Prof. Dr. V. Claus

Gutachter: Prof. Dr. L. Richter

Prof. Dr. Y. Wallach

Prof. Dr. M. Reimer

Inhaltsverzeichnis

§ 0	Einleitung	1
<u>Kapitel I</u>	Eine funktionsorientierte Prozeßrechner- struktur	10
§ 1	Aufteilung der Systemfunktionen auf einzelne Module	10
§ 2	Verarbeitungs- , Speicher- und Datentransfereinheiten	21
<u>Kapitel II</u>	Ablaufsteuernde Funktionseinheiten	67
§ 3	Der Task-Scheduler	67
§ 4	Der Dispatcher	78
<u>Kapitel III</u>	Beispiele für den Einsatz speziali- sierter Prozessoren	108
§ 5	Mathematische Grundlagen zur Erfassung und Darstellung empirischer Funktionen mit Hilfe lokaler Integrale	109
§ 6	Module zur integrierenden Erfassung und Ausgabe kontinuierlicher Zeitfunktionen	128
Anhang zu § 5		154
Literaturverzeichnis		160

§ 0 Einleitung

Prozeßrechensysteme haben die Aufgabe, zeitlich synchron mit den Ereignissen in technischen Prozessen diesen zugeordnete Programme auszuführen.

Bisher werden im wesentlichen busorientierte, modular organisierte Einprozessoranlagen mit der von-Neumann-Struktur als Prozeßrechner eingesetzt, die um geeignete Prozeßperipheriegeräte erweitert sind und die eine Unterbrechungseingabeeinheit als das den Realzeitbetrieb ermöglichende Element besitzen. Bei prinzipiell unveränderter Architektur, die auch für die Mikrocomputer übernommen wurde, fanden die Entwicklungen der Halbleitertechnologie jeweils schnell Eingang in den Hardwareaufbau. Deshalb kann man davon ausgehen, daß Prozeßrechner z. Zt. mit Hilfe von Bit-Slice-Mikroprozessoren konstruiert und mikroprogrammiert sind sowie eine Wortlänge von 16 Bits besitzen.

Da die Peripheriekomponenten nur recht elementare Operationen selbständig durchführen können, ist die Belastung der Zentralprozessoren durch Betriebssystemfunktionen sehr groß. Prozeßrechnerbetriebssysteme sind zumeist schichtenweise organisiert [3], wobei die Unterbrechungsverwaltung als wesentliches Element der Ablaufsteuerung den Kern bildet. Darauf baut der Scheduler auf, der beim Eintreten bestimmter Ereignisse die mit diesen assoziierten Taskoperationen einleitet. Die Betriebsmittelverteilung an lauffähige Tasks nimmt auf Grund i.a. fest vorgegebener

Prioritäten der Dispatcher vor. Damit kann man trotz richtiger Wahl der Prioritäten im ungünstigsten Falle eine maximale Prozessorausnutzung von nur 70% erreichen [21]. Diese verschlechtert sich noch, wenn man - wie es oft geschieht - den Treibern langsamer Peripheriegeräte zu deren besserer Auslastung die höchsten Prioritäten zuordnet. In der Literatur [8,14-17,19,20] wurden leistungsfähigere und sogar optimale Dispatching-Algorithmen angegeben, die jedoch wegen des sehr großen Bearbeitungsaufwandes, der auch den Einsatz spezieller Hardwaremodule erfordert, bisher noch nicht implementiert wurden. Darüberhinaus nehmen die Betriebssysteme Peripherietreiber-, Synchronisations- und Kommunikationsfunktionen wahr. Alle diese Aufgaben werden zusammen mit den Anwendertasks im Mehrprogrammbetrieb bearbeitet.

Für das oben umrissene Konzept waren vor allem Kostengründe entscheidend. Jedoch haben sich mit der fortschreitenden Erhöhung der Integrationsdichte von Halbleiterbausteinen und der Entwicklung von Mikroprozessoren die Kostenverhältnisse umgekehrt: Datenübertragungseinrichtungen und die Verkabelung einzelner Komponenten sind wesentlich teurer als Verarbeitungs- und Speicherelemente. Darum erscheint es angebracht zu sein, nach neuen und leistungsfähigeren Strukturen für Prozeßrechner zu suchen, die die ihren Aufgabenstellungen innewohnende Parallelität widerspiegeln und zu einer Senkung der Gesamtkosten beitragen.

Das Ziel der vorliegenden Arbeit ist es deshalb, eine diesen Kriterien genügende dezentrale Rechnerstruktur zu beschreiben und einen Satz von Funktionseinheiten zu definieren, aus denen sich für jede Anwendung ein geeignetes asymmetrisches Mehrprozessorsystem konfigurieren läßt.

Den Ausgangspunkt bildet hierbei die in der Prozeßdatenverarbeitung auf der Ebene der Teilaufgaben (Tasks) gegebene Möglichkeit zur Paralleltätigkeit. Dementsprechend sollen verschiedene allgemeine und spezialisierte Prozessoren vorgesehen werden, von denen die ersteren sich noch in ihrer Leistungsfähigkeit unterscheiden können und zur Bearbeitung der Benutzertasks dienen, während den anderen bestimmte Betriebssystem- und Anwenderaufgaben fest zugeordnet sind.

So wird ein besonderes Modul für die wesentliche Aufgabe der Prozeßdatenverarbeitung, das Task-Scheduling und die Ereignisverwaltung, bereitgestellt, die die Verwendung ganz allgemeiner Einplanungsbedingungen gestattet. Der Dispatcher verwaltet die ablaufbereiten Tasks nach einem sehr einfachen Zustandsmodell [9] und nimmt die Betriebsmittelvergabe vor, wobei auf Deadlockfreiheit durch a priori Verhinderung geachtet wird. Die angewandten Algorithmen, Antwortzeitsteuerung und Vorhaltestrategie sowie eine Erweiterung letzterer für asymmetrische Mehrprozessorsysteme, die auch Lastverlagerungen an schnellere Verarbeitungseinheiten erlaubt, gewährleisten die Einhaltung vorgegebener Reaktionszeiten, sofern dieses überhaupt möglich ist. Schließlich werden auch für die Prozeßkommunikation und

zum Treiben der Peripherie spezielle Module eingerichtet und die Dateiverwaltung in den Speichereinheiten angesiedelt, an die auch die Hintergrundspeicher angeschlossen werden. Durch diese Gliederung werden die allgemeinen Verarbeitungsmodule zum größten Teil vom Verwaltungsaufwand des Betriebssystems entlastet und damit die meisten Context-Switching-Operationen eingespart.

Dem gleichen Zweck, nämlich der Reduzierung des internen Übertragungsaufwandes, der zum bestimmenden Faktor des Hardwarepreises werden kann, dienen auch die Verlagerung komplexer Operationen, vornehmlich zur Datenkonzentration, an die Entstehungsorte der Daten in anwendungsspezifische Module und die Einsparung der Instruktionsadressentransfers. Letzteres wird durch die Integration der Programmadressenverwaltung, die eine entsprechende Vorinterpretation der Befehle erfordert, in die Speichereinheiten erzielt. Zur Vermeidung von Zugriffskonflikten und zur Erhöhung der Arbeitsgeschwindigkeit werden in verschiedenen der eingeführten Funktionsmodule lokale Speicher, darunter auch sequentiell organisierte, vorgesehen.

Die vorgeschlagene Rechnerstruktur ermöglicht eine klare Zuordnung der einzelnen parallelen Prozesse, insbesondere der voneinander unabhängigen, zu verschiedenen Subsystemen und fördert somit die Übersichtlichkeit und deshalb auch die Zuverlässigkeit der Software. Da im Gegensatz zu Einprozessoranlagen die Betriebsmittelmengen paralleler Prozesse in dezentral organisierten Rechnern i.a. disjunkt

sind, wird durch Senkung der Ausfallwahrscheinlichkeit der Gesamtsysteme zu einer erhöhten Verfügbarkeit beigetragen.

Ansätze zu funktionsorientierten Rechnerstrukturen sind in der Form von Ein-/Ausgabeprozessoren z.B. bei den Rechnern der Serien 6000 und Cyber 170 der Control Data Corporation sowie bei verschiedenen Mikrocomputersystemen zu finden. In dieser Arbeit werden Programm- und Datenspeichermodule eingeführt werden, die einen erheblichen Teil der üblicherweise von Zentralprozessoren durchgeführten Aufgaben übernehmen. Einfache Vorformen solcher Einheiten weist der Mikrorechner F8 von Fairchild auf.

Im ersten Paragraphen der vorliegenden Arbeit werden acht Typen von Funktionseinheiten definiert, aus denen sich modular strukturierte Mehrprozessorsysteme aufbauen lassen. Den Einheiten wird nach den oben aufgeführten Gesichtspunkten jeweils die Zuständigkeit für die Durchführung eines Teiles der in der Prozeßdatenverarbeitung benötigten Grundoperationen und Betriebssystemdienste übertragen. Die nicht für Prozeßrechner typischen Verarbeitungseinheiten, Programm- und Datenspeichermodule, Peripherieadapter, Verbindungs- sowie Nachrichtenübertragungseinheiten werden im zweiten Paragraphen beschrieben. Nach dieser Darlegung der Grundzüge einer funktionsorientierten Rechnerstruktur im ersten Kapitel, behandeln die folgenden Kapitel eingehend die Arbeitsweise der den internen Ablauf eines Systems

steuernden Einheiten und auf bestimmte Aufgaben spezialisierter Prozessoren. Der dritte Paragraph ist dem Task-Scheduler gewidmet, der die im System erzeugten Signale verarbeitet und die Einplanungsbedingungen aller Teilprozesse, für die eine theoretische Systematisierung angegeben wird, auswertet. Lauffähige Tasks werden an den Dispatcher zur Betriebsmittelvergabe und zum Start der Ausführung weitergeleitet. Die entsprechenden Bedingungen werden im Paragraphen 4 formuliert. Weiterhin hat die letztgenannte Einheit die Aufgabe, den Ablauf der einzelnen Prozesse zu überwachen und ihre Bearbeitungsreihenfolge derart zu wählen, daß nach Möglichkeit strikte Antwortzeitbedingungen eingehalten werden können.

Am Beispiel der Erfassung, Darstellung und Ausgabe kontinuierlicher Zeitfunktionen sollen im dritten Kapitel die Einsatzmöglichkeiten für spezialisierte Prozessoren aufgezeigt werden. Dafür werden im Paragraphen 5 die mathematischen Grundlagen von mit Hilfe lokaler Integrale konstruierter Spline-Approximationsoperatoren entwickelt, die sich durch gute Rauschunterdrückungseigenschaften auszeichnen. Der folgende Paragraph beschreibt dann die Arbeitsweise entsprechender Module zur fortlaufenden Bildung und Weiterverarbeitung lokaler Integrale von Zeitfunktionen und zur Rückgewinnung und späteren Ausgabe der Funktionsverläufe aus den erfaßten Daten.

Herrn Prof. Dr. Lutz Richter möchte ich an dieser Stelle
für wertvolle Ratschläge und Hinweise bei der Anfertigung
dieser Arbeit sehr herzlich danken.

Zur Darstellungsweise der Algorithmen

Die Arbeitsweise der hier vorgestellten Funktionseinheiten wird zum größten Teil auch durch die Angabe von Algorithmen beschrieben. Den einzelnen Programmstücken sind jeweils Texte vorangestellt, in denen die Bedeutungen der verwendeten Daten behandelt und die Abläufe der Prozeduren verbal umrissen werden.

Zur Steigerung der Übersichtlichkeit und um das Wesentliche deutlich zu machen, werden wir die Programme in "Pidgin"-Algol 68 (vgl. [1]) formulieren, da Algol 68 die Darstellung paralleler Aktivitäten gestattet. Die Routinen werden jeweils als Reaktion auf ein Signal ausgeführt, was durch das Sprachmittel

on <signal> do <anweisungsfolge> od

ausgedrückt werden soll. Wir werden Mengenoperationen verwenden und benötigen die Konstruktionen

for all <elementrelation> [with <bedingung>] do
 <anweisungsfolge> od

und

<skalare variable> <menge> with <bedingung>

um die Elemente einer Menge verarbeiten zu können, die einer gewissen Bedingung genügen bzw. zur Auswahl eines durch die angegebene Bedingung eindeutig bestimmten Elementes einer Menge. Mit cont bezeichnen wir den Inhaltsoperator. Die Anweisungen clear, trigger und start dienen zum Löschen gewisser Objekte, zum Aussenden von Signalen bzw.

zur Freigabe von Taktfrequenzen.

Die angegebenen Algorithmen wurden zur Verifikation auch in der Simulationssprache SIMULA formuliert und anschließend getestet.

Kapitel I Eine funktionsorientierte Prozeßrechnerstruktur

§ 1 Aufteilung der Systemfunktionen auf einzelne Module

Ausgehend von den Einsatzbereichen für Prozeßrechner werden wir in diesem Abschnitt die Funktionen ihrer Betriebssysteme und die Elemente der Anwendungsalgorithmen umreißen. Daran schließt sich die Definition eines Satzes von Funktionseinheiten an, mit denen sich jeweils ein der Aufgabenstellung angepaßter, dezentral strukturierter Prozeßrechner konfigurieren läßt. Die einzelnen Tätigkeiten werden dann allgemeinen Modulen - Verarbeitungs- und Verbindungseinheiten sowie Programm- und Datenspeichermodule - und auf die Betriebssystemdienste Task-Scheduling, Dispatching, Peripherieanpassung sowie Nachrichtenaustausch spezialisierten Prozessoren zugewiesen. Weiterhin werden Beispiele dafür angeben, auch häufig vorkommende und mithin standardisierbare anwendungsbezogene Aufgabenstellungen mit speziellen Modulen abzudecken. Der Paragraph schließt mit einer Übersicht über die Funktionen der eingeführten Module und der Angabe einer Beispielkonfiguration.

1.1 Prozeßrechnerfunktionen

Um die von Prozeßrechnern auszuführenden Tätigkeiten herauszuarbeiten, ist es notwendig, zuerst ihre Einsatzbereiche näher zu charakterisieren. Entsprechende Untersuchungen [18] haben ergeben, daß Prozeßrechner im wesentlichen für die folgenden, in der Reihenfolge steigender Komplexität aufgeführten, Aufgaben eingesetzt werden:

- Meßwerterfassung
- Überwachung und Protokollierung technischer Prozesse
- Steuern von Abläufen und Regeln technischer Prozesse
- Sollwertführung anhand mathematischer Prozeßmodelle und entsprechendes Regeln technischer Prozesse
- Prozeßführung auf der Grundlage mathematischer Modelle deren Parameter in Echtzeit optimiert werden
- Disposition und Steuerung größerer Prozeßsysteme.

Die Bearbeitung der entsprechenden Einzelaufgaben läßt sich jeweils auf die Ausführung einer Reihe von Systemfunktionen zurückführen, die sich in anwendungs- und betriebssystemorientierte Funktionen einteilen lassen. Letztere ergeben sich aus den Anforderungen, die die Implementierungen virtueller Maschinen und höherer Programmiersprachen auf realen Maschinen stellen, und wurden bereits hinreichend genau definiert [4,5,18]. Danach können wir ein Prozeßrechner-Betriebssystem in die folgenden Systemdienste aufgliedern:

- B1. Unterbrechungsbearbeitung
- B2. Prozeßverwaltung
- B3. Prozeßkommunikation
- B4. Prozeßsynchronisation
- B5. Zeitverwaltung
- B6. Durchführung von Ein- und Ausgaben
- B7. Mensch-Maschine-Kommunikation
- B8. Datei- und Datenbankverwaltung
- B9. Speicherplatzverwaltung
- B10. Fehlerbehandlung
- B11. Bearbeitung von Ausnahmeständen.

Wegen der Fülle der möglichen Prozeßrechner-Anwendungen läßt sich auf der anderen Seite allerdings kein relativ kleiner Satz anwendungsorientierter Systemfunktionen hohen Komplexitätsniveaus angeben. Wie die Analyse höherer, technisch-wissenschaftlicher Programmiersprachen zeigt, können wir jedoch festhalten, daß die entsprechenden Algorithmen aus den folgenden Elementen aufgebaut sind:

- A1. Abbildungen - insbesondere arithmetische und logische Verknüpfungen
- A2. Speichern und Wiederverfügbarmachen von Daten
- A3. logische Entscheidungen
- A4. Ablaufsteuerung
- A5. Datentransfer
- A6. Kommunikation mit dem Betriebssystem.

1.2 Moduldefinition

Die Ausführung der unter Punkt A1, A3 und A4 genannten Operationen übertragen wir - wie auch bisher schon üblich - allgemeinen Verarbeitungseinheiten. Um die Zahl der zur Programmbearbeitung notwendigen Datenübertragungen zu reduzieren, statten wir diese Prozessoren mit kleineren Speichern für häufig benötigte Daten aus.

Im Gegensatz zum klassischen von Neumann'schen Konzept unterscheiden wir zwischen Programmspeichermodulen zum Bereitstellen von Instruktionen und Datenspeichermodulen zur Aufnahme von Operanden. Unter der Voraussetzung, daß die Verarbeitungseinheiten von möglichst vielen Routineaufgaben entlastet werden sollen, erscheint diese Aufteilung sinnvoll, da die beiden Speichertypen unterschiedlich arbeiten. Während nämlich die Programmspeichermodule Befehle in vorwiegend sequentieller Reihenfolge ausgeben, unterstützen Datenspeichermodule verschiedene Adressierungsarten und übertragen Daten in beiden Richtungen, was auch mit Darstellungstransformationen verbunden sein kann. Weiterhin vereinfacht die Unterscheidung der Speichertypen die Programm- und Datensicherung und ermöglicht die Verwendung verschiedener Wortlängen. Die zu einer Speicherhierarchie gehörenden Massenspeichereinheiten werden direkt mit den entsprechenden Speichermodulen verbunden, die mithin die notwendigen Verwaltungs-, Treiber- und Dateifunktionen wahrzunehmen haben.

Zum Anschluß der übrigen Peripheriegeräte und zur Durchführung der jeweiligen Anpassungen sollen Peripherieadapter dienen. Auf Anstoß von Verarbeitungseinheiten führen sie den Datentransfer mit externen Systemkomponenten selbständig durch. Die Kommunikation zwischen den bisher genannten Komponenten und den das Betriebssystem beherbergenden Modulen vollzieht sich über Verbindungseinheiten, an die die übrigen Module angeschlossen werden.

Die Definition weiterer Funktionseinheiten, die nicht überall eingesetzt werden können, erscheint für solche Aufgabenstellungen gerechtfertigt zu sein, die sich sehr häufig wiederholen und die hinreichend gut standardisierbar sind. Aus der letztgenannten Bedingung folgt, daß sich vor allem Aufgaben aus den drei oben erwähnten Einsatzbereichen mit geringerer Komplexität dafür anbieten.

Neben den in Kapitel III näher behandelten Funktionsmodulen sollen an dieser Stelle exemplarisch noch einige mögliche spezialisierte Prozessoren genannt werden:

- arithmetische (Feld-) Prozessoren
- Datenerfassungs- und Vorverarbeitungseinheiten
- Einheiten zum Steuern, Regeln und zur Parameterüberwachung
- Bedienkommandointerpretierer
- Umsetzer für Datendarstellungen
- Einheiten zur Listenaufbereitung und -ausgabe
- Einheiten zur graphischen Ausgabe
- Datenkomprimierende und -expandierende Übertragungseinheiten.

Während die Betriebssystemprogramme herkömmlicher Digitalrechner von denselben Verarbeitungseinheiten ausgeführt werden, die auch die Anwenderprogramme abarbeiten, soll nun eine Reihe von Systemkomponenten definiert werden, die ausschließlich die entsprechenden Aufgaben wahrnehmen.

Im Stapel- und Teilnehmerbetrieb werden Aufträge an eine Rechenanlage übergeben, die dieser i.a. vorher nicht bekannt sind und die so schnell wie möglich erledigt werden sollen. Dagegen sind die Aufträge, die ein Prozeßrechner zu bearbeiten hat, bereits im System vorhanden. Aufgabe des Task-Schedulers ist es nun, ständig die von der Zeit, von Signalen und inneren Zuständen abhängigen Bedingungen zu überprüfen, an die die Ausführung der einzelnen Aufträge geknüpft ist. Da die allgemeinen Verarbeitungseinheiten durch die Wahrnehmung der genannten Prozeßverwaltungsaufgaben erheblich belastet werden würden, erscheint es angebracht zu sein, das Task-Scheduling von einem selbständigen Funktionsmodul ausführen zu lassen, zumal die Systemdienste und die Anwenderaufträge logisch nicht zusammengehören. Sind die Laufbedingungen der Benutzerprozesse erfüllt, dann müssen ihnen Betriebsmittel zugeteilt werden, eine Reihenfolge für die Abarbeitung der einzelnen Tasks ist aufzustellen und schließlich muß deren Ausführung überwacht werden, wobei jederzeit sichergestellt sein muß, daß es nicht zu Systemverklemmungen kommen kann. Hier liegt eine weitere in sich abgeschlossene, komplexe und umfangreiche Aufgabenstellung vor, die wir einer beson-

deren Funktionseinheit, dem Dispatcher, übertragen.

Zur Durchführung des Datenaustausches mit Peripherieleggeräten wurden bereits oben die Peripherieadapter eingeführt. Es liegt nahe, hier die Treiberfunktionen des Betriebssystems anzusiedeln, wodurch die Verarbeitungseinheiten von diesen Routineaufgaben entlastet werden und einheitliche Schnittstellen zur Peripherie geschaffen werden können. An die Adapter sollen vor allem die Prozeßperipherie, die mit dem Menschen korrespondierenden Geräte und spezialisierte Prozessoren angeschlossen werden. Dagegen werden die Massenspeicher, wie bereits oben erwähnt, i.a. von Speichermodulen gesteuert.

Neben den Verbindungseinheiten zum Transfer größerer Datenmengen sind alle Systemkomponenten an eine Nachrichtenübertragungseinheit angeschlossen, die zur Prozeßkommunikation und zum Austausch kurzer Mitteilungen dient.

Eine zusammenfassende Darstellung der vorgeschlagenen Zuordnung der Systemfunktionen von Prozeßrechnern zu den verschiedenen Typen von Funktionsmodulen ist in Tabelle 1 gegeben. Dabei wurden einige Funktionen gleichzeitig mehreren Einheiten zugewiesen. Der Grund dafür liegt in der Anordnung der verschiedenen Speicher und darin, daß jede Funktionseinheit die in ihrem Bereich anfallenden Fehler und Ausnahmestände sowie die notwendigen Sicherungs- und Synchronisationsaufgaben selber behandeln soll.

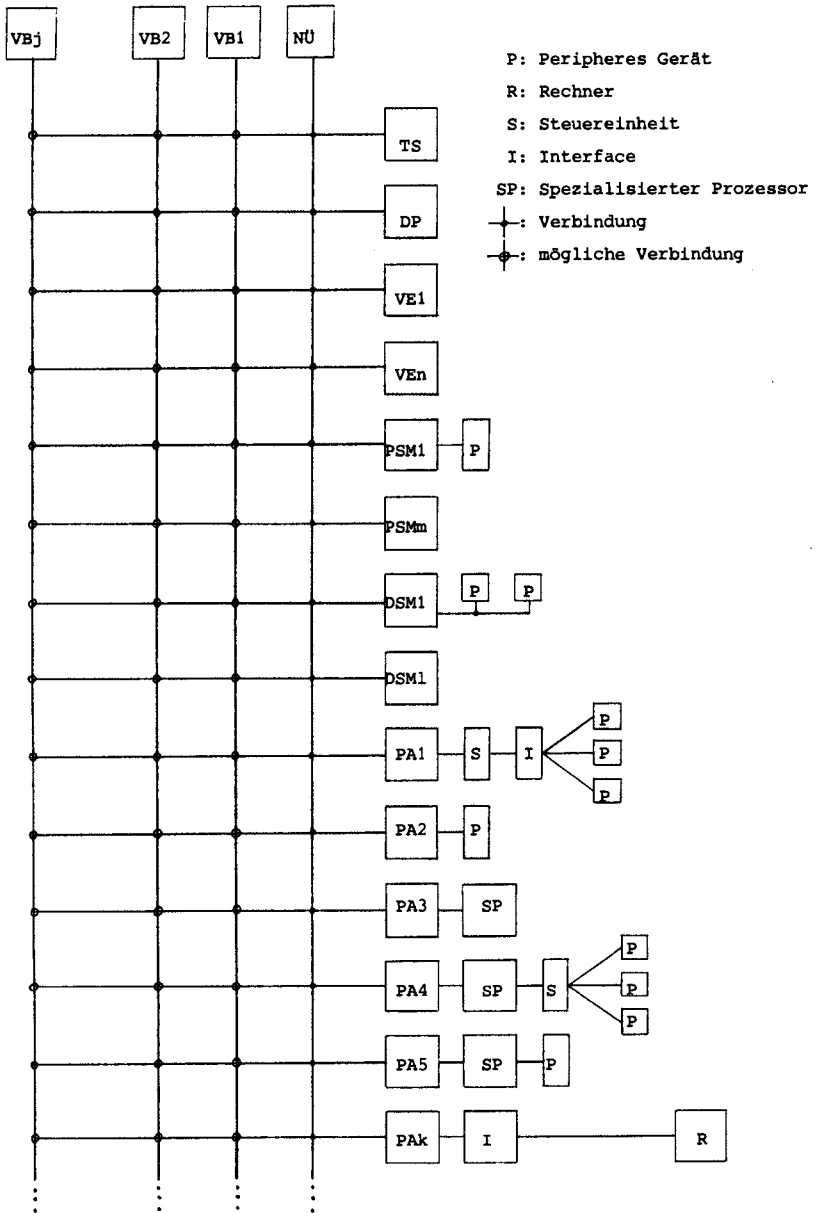
Tabelle 1. Zuordnung der Systemfunktionen von Prozeß-
rechnern zu einzelnen Funktionseinheiten

Module	Funktionen
1. Verarbeitungseinheiten (VE)	Programmausführung Speichern häufig benötigter Daten Instruktionszwischenspeicherung
2. Programmspeichermodule (PSM)	Bereitstellen von Instruktionen Ausführen von Programmsprüngen Speicherhierarchieverwaltung Peripherietreiberfunktionen Zugriffskontrolle und -synchronisation Urladen
3. Datenspeichermodule (DSM)	Speichern und Bereitstellen von Operanden Adressengenerierung Speicherhierarchieverwaltung Peripherietreiberfunktionen Datei- und Datenbankverwaltung Darstellungstransformationen Zugriffskontrolle und -synchronisation
4. Peripherieadapter (PA)	Durchführung des Datenaustausches mit externen Komponenten Treiberfunktionen Darstellungstransformationen

Module	Funktionen
5. Verbindungseinheiten (VB)	Datentransfer Leistungssteuerung
6. Nachrichtenübertragungs- einheit (NÜ)	Nachrichtenaustausch zur Prozeßkommunikation und -synchronisation
7. Task-Scheduler (TS)	Unterbrechungs- und Signal- verarbeitung Zeitverwaltung Task-Einplanung Prozeßkoordinierung Prozeßzustandstransfer (Wieder-) Anlauf des Systems
8. Dispatcher (DP)	Prozeßzustandstransfer Betriebsmittelvergabe Parameterübergabe Laufzeitkontrolle Verwaltung des Pausierens von Tasks Antwortzeitsteuerung Synchronisation

1.3 Eine Beispielkonfiguration

Die Figur 1 zeigt als Beispiel die Struktur eines Rechners, der aus den hier eingeführten Einheiten zusammengestellt wurde. Grundsätzlich ist es möglich, jedes Modul an jede vorhandene Verbindungseinheit anzuschließen. Von der jeweiligen Anwendung hängt es ab, welche Verbindungen benötigt und deshalb tatsächlich realisiert werden. Mithin muß die dargestellte Verschaltungsstruktur deutlich von einem Kreuzschienensystem unterschieden werden.



Figur 1. Struktur eines modular aufgebauten Prozeßrechners

§ 2 Verarbeitungs- , Speicher- und Datentransfereinheiten

Nachdem wir im letzten Paragraphen aus den Aufgabenstellungen acht Funktionsmodultypen zum Aufbau von Prozeßrechnern abgeleitet haben, wollen wir nun alle die Einheiten näher beschreiben, denen nicht die Task-Ablaufsteuerung obliegt. Das wird zu einem großen Teil durch die Angabe entsprechender Prozeduren geschehen. Bei der Konzeption der Module und ihres Zusammenwirkens wurde immer von der Forderung ausgegangen, den Übertragungsaufwand zwischen ihnen so gering wie möglich zu halten, um so den Kostenverhältnissen Rechnung zu tragen. Deshalb konnte bei der Definition der Verbindungseinheiten, die in einem System mehrfach vorhanden sein können, am bislang üblichen Buskonzept festgehalten werden. Jedoch soll die Bussteuerung auf Grund dynamisch veränderlicher Prioritäten erfolgen. Daneben steht mit der Nachrichtenübertragungseinheit ein zweites Kommunikationssystem zur Übermittlung von Nachrichten zwischen parallelen Prozessen zur Verfügung, das wegen der Verwendung von Fifo-Speichern synchronisationsfrei arbeitet. Das Modul kann auch als Datenfernübertragungseinrichtung zwischen räumlich verteilt angeordneten Komponenten dienen und wird dann i.a. mehrere Netzknoten besitzen. Die Speicherung und Bereitstellung von Informationen, die Datensicherung sowie eventuell die Dateiverwaltung auf angeschlossenen Massenspeichern werden von den Speichereinheiten weitgehend selbständig durchgeführt. Darüberhinaus übernehmen die Datenspeichermodule auch einige Routine-

aufgaben und unterstützen verschiedene Datenstrukturen und Adressierungsverfahren. Durch die Integration von Programmzählern, Befehlsinterpretierern und Stapelzeigern in die Programmspeichermodule zur Behandlung auch nichtsequentieller Befehlsabfolgen werden die Instruktionsadressentransfers völlig eingespart. Nach Erkennen bedingter Verzweigungen werden vorausschauend jeweils zwei alternative Befehlsströme an die Verarbeitungseinheiten übertragen, wo sie ebenso wie kurze Programmschleifen in geeigneten Zwischenspeichersystemen Aufnahme finden. Der Erhöhung der Arbeitsgeschwindigkeit dienen auch in den Verarbeitungseinheiten vorgesehene Scratch-Pads, die den parallelen Zugriff auf die Argumente dyadischer Operatoren erlauben. Die Peripherieadapter entsprechen weitgehend Front-End-Prozessoren und sind der Sitz der Treiberprogramme des Betriebssystems: sie führen den Datentransfer mit den und die Anpassung an die verschiedenen Peripheriegeräte durch. Hinsichtlich der Zahl der internen Datenübertragungen, der Ausfallwahrscheinlichkeiten und der Möglichkeiten zur Parallelarbeit wird schließlich das hier vorgeschlagene Konzept mit der konventionellen Einprozessorstruktur verglichen.

2.1 Verbindungseinheiten

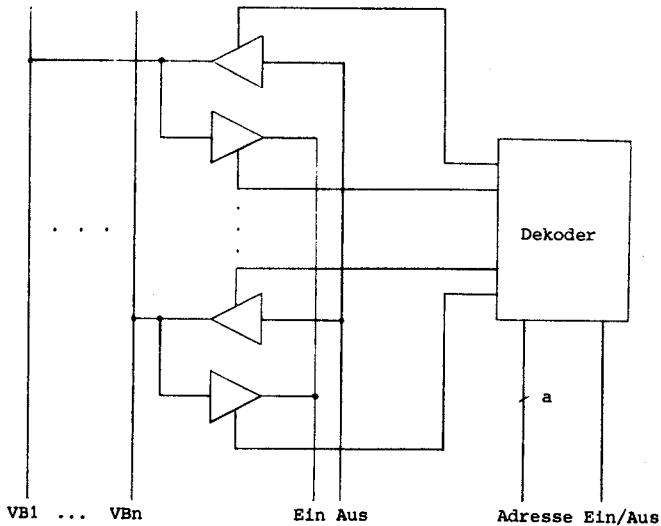
Zum Austausch von Informationen, insbesondere größerer Blöcke, sind alle Module, wie Figur 1 zeigt, an Verbindungseinheiten angeschlossen. Diese Funktion wird in konventionellen Prozeßrechnern i.a. von einem Bus wahrgenommen. Oft sind auch mehrere, unterschiedlich breite, Busse, nämlich Adreß-, Daten- und Kontrollbus oder Programmadreß- und Befehlsbus sowie Daten- und Datenadreibus usw., vorhanden. Das Buskonzept besitzt gegenüber anderen Verbindungsstrukturen den Vorteil minimalen Verkabelungsaufwandes, erzeugt jedoch andererseits Flaschenhälse, weil jeweils nur eine Einheit Informationen über einen Bus senden kann. Eine netzwerkartige Verbindung der einzelnen Funktionsmodule, die den Erfordernissen der jeweiligen Anwendung und Programmstruktur entspricht, wird dagegen i.a. zu aufwendig sein und widerspricht unserem Optimierungskriterium, nämlich mit möglichst wenig Verkabelung auszukommen. Daher halten wir am Buskonzept zur Definition der Verbindungseinheiten fest.

Die letzteren setzen sich mithin jeweils aus dem eigentlichen Leitungssystem und der Bussteuerung zusammen, die für die Synchronisation der Buszugriffe sorgt. Neben den Datenübertragungsleitungen besteht jeder Bus aus Leitungen, auf denen die einzelnen Einheiten Buszugriffe anfordern können, aus einem Adreßbus, auf den die Nummern der Module geschaltet werden, die Buszugriffe erhalten bzw. zu denen Daten übertragen werden sollen, und weiteren Leitungen zum

Austausch der ablaufsteuernden Signale. Fordern mehrere Einheiten gleichzeitig Buszugriffe an, so löst die Bussteuerung diesen Konflikt dadurch auf, daß sie den Zugriff dem Modul mit der höchsten Priorität gewährt. Um diese Prioritätssteuerung variabel zu gestalten, besitze jede Verbindungseinheit einen Registersatz, in den der Dispatcher die Prioritätsreihenfolge einschreibe und anhand dessen die Bussteuerung arbeite. Damit beim Eintreten von Fehlersituationen auf eventuell vorhandene weitere Verbindungseinheiten umgeschaltet werden kann, sei jede VB über eine Signalleitung mit dem Task-Scheduler verbunden. Nachdem der Dispatcher einer Task die benötigten Betriebsmittel zugeordnet hat, muß er dafür sorgen, daß diese über eine oder mehrere Verbindungseinheiten Daten miteinander austauschen können. Dazu teilt er den einzelnen Modulen mit, über welche Busse ihre Ein-/Ausgabeschnittstellen Datentransfers durchführen sollen. Die Art des Anschlusses einer solchen Schnittstelle an mehrere VB's ist für eine Ein-/Ausgabelleitung in Figur 2 skizziert.

In den später angegebenen Algorithmen stehen die Aufrufe der Prozeduren put und get stellvertretend für die Übertragung eines Datenwortes mittels einer Verbindungseinheit. Durch get(<Datenwort>) werde das nächste an die aufrufende Einheit abgeschickte Datenwort eingelesen, während mit put(<Modul>,<Datenwort>) ein solches zu einem bestimmten Modul übertragen werde.

Engpässe beim Datentransfer dürften sich im wesentlichen vermeiden lassen, da jedes Funktionsmodul an mehrere Ver-



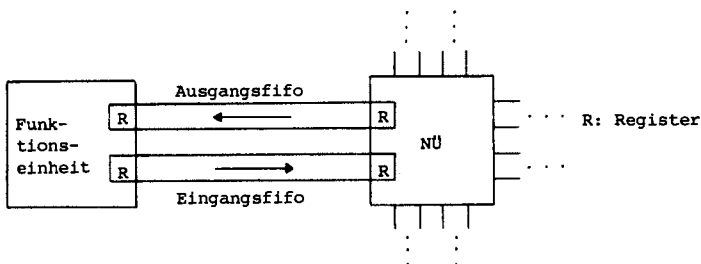
Figur 2. Anschluß einer Ein-/Ausgabeschnittstelle an mehrere Busse

bindungseinheiten angeschlossen werden kann, wodurch die Auswahl zwischen verschiedenen Übertragungswegen ermöglicht wird, und der Umfang der Datenübertragungen ohnehin durch strukturelle Maßnahmen erheblich reduziert werden soll. Durch die Verfügbarkeit mehrerer Kommunikationswege wird die Gesamtausfallwahrscheinlichkeit eines Systems gesenkt. Ohne Mitwirkung und ohne Behinderung von Verarbeitungseinheiten können verschiedene Module, z.B. Speicher und Peripherieadapter, Daten austauschen, eventuell sogar parallel auf mehreren Bussen zur Geschwindigkeitssteigerung. Mehrere Prozessoren können gleichzeitig arbeiten und parallel auf die ihnen zugeordneten Programm- und Datenspeichermodule zugreifen, wenn jeder Speicher über mindestens einen Bus allein verfügen kann. Schließlich lassen sich auf der Basis

des Mehrbuskonzeptes auch hierarchische Rechnerstrukturen verwirklichen, wenn jeweils nur wenige Einheiten an einen Bus angeschlossen werden und die Kommunikation zwischen beliebigen Komponenten des Gesamtsystems i.a. die Mitwirkung weiterer Module erfordert.

2.2 Die Nachrichtenübertragungseinheit

Zum Austausch kurzer Nachrichten zwischen parallel ablaufenden Prozessen sind, wie in Figur 1 dargestellt, alle weiteren Funktionseinheiten an ein zweites Kommunikationssystem, die Nachrichtenübertragungseinheit, angeschlossen. Da diese Einrichtung im wesentlichen zur Prozesssynchronisation dienen soll, ist es vorteilhaft, wenn ihre Arbeit nicht mit anderen Prozessen koordiniert werden muß. Das läßt sich erreichen, wenn die $n \geq 1$ Module mit der eigentlichen Übertragungseinheit über jeweils zwei in entgegengesetzten Richtungen arbeitenden First-in-first-out-Speichern (Fifos) verbunden werden (vgl. Figur 3).



Figur 3. Anschluß der Nachrichtenübertragungseinheit

Durch Kopplung mehrerer als Netzknoten fungierender Transfereinheiten mit Hilfe jeweils zweier Fifos lassen sich Kommunikationsnetzwerke aufbauen, die nach außen wie eine Nachrichtenübertragungseinheit erscheinen. Die Aufgabe der in den Netzknoten ablaufenden und unten angegebenen Prozedur transfer ist es, die durch die Eingangsfifos eintreffenden Informationen an die ihnen direkt angeschlossenen Empfänger oder an geeignete Nachbarknoten weiterzuleiten, die den Endempfängern näher liegen. Dabei muß die Reihenfolge der Absendezeitpunkte der einzelnen Nachrichten berücksichtigt werden. Tritt der Fall ein, daß die Nachrichten nicht mehr weitergeleitet werden können, weil die Ausgangsfifos gefüllt sind, so werden keine weiteren Daten mehr entgegengenommen. Dadurch stauen sich die Nachrichten in den Eingangsfifos an. Wenn letztere gefüllt sind, können keine weiteren Eingaben mehr vorgenommen werden und die Übertragungseinheit hat die Möglichkeit, den Stau abzubauen, wenn die angeschlossenen Module die Ausgangsfifos zu leeren beginnen.

Besitzt die Nachrichtenübertragungseinheit mehr als einen Netzknoten, so läßt sich mit ihr die Verbindung zwischen örtlich verteilt angeordneten Teilsystemen, die über eigene Busse verfügen, bzw. Einzelkomponenten eines Prozeßrechners herstellen. In Abhängigkeit der Entfernungen zwischen den Netzknoten und des zu übertragenden Datenvolumens kann der eigentliche Datentransfer auch seriell erfolgen.

Bevor wir die in den Netzknoten ablaufende Prozedur transfer angeben werden, wollen wir nun ihre Arbeitsweise zusammen mit den von ihr bearbeiteten Datenstrukturen verbal beschreiben.

Die Ein- und Ausgänge der Fifos in den angeschlossenen Einheiten erscheinen dort als Register oder Speicherplätze.

In der NÜ seien $\text{fifoein}[1:n]$ die Ausgangsregister der Eingangsfifos und $\text{fifoaus}[1:n]$ die Eingangsregister der Ausgangsfifos. Um den Belegungszustand dieser Speicher abfragen zu können, seien die Nicht-leer-Signale der Eingangsfifos als $\text{fnl}[1:n]$ und die Anzahlen der jeweils in den Ausgangsfifos freien Speicherplätze als $\text{fsp}[1:n]$ verfügbar.

In jedem Netzknoten sei die Nummer des Ausgangs, in den eine an die Einheit e_i adressierte Nachricht geladen werden muß, als $u[e_i]$ gespeichert. Die Konstante dim sei eine durch die Anzahl der an die NÜ angeschlossenen Module und die Struktur des jeweiligen Rechner- und Programmsystems bestimmte Felddimension.

Will nun eine Einheit eine Nachricht senden, dann gibt sie diese in der Form $(s, t, l, e_1, \dots, e_l, m, d_1, \dots, d_m)$ in den von ihr ausgehenden Eingangsfifo ein, falls letzterer Daten aufnehmen kann. Dabei seien s die Adresse des Senders, t der Absendezeitpunkt der Nachricht, l die Anzahl der Empfänger und e_i , $i=1, \dots, l$, ihre Adressen sowie m die Anzahl der eigentlichen Datenworte d_j , $j=1, \dots, m$. Durch den Fifo-Speicher gelangt die Nachricht zum nächsten Netzknoten der Übertragungseinheit, wo beim nächsten Abfragezyklus s , t und l eingelesen und in den der sendenden Einheit zugeord-

neten Elementen der Felder $o[1:n]$, $z[1:n]$ bzw. $le[1:n]$ abgelegt werden. Danach liest die Prozedur transfer die Empfängeradressen e_1, \dots, e_l ein und stellt fest, an welche Empfänger - Funktionsmodule und Netzknoten - die Nachricht zu senden ist. Ihre Anzahl und ihre Adressen werden in den Feldern $la[1:n]$ und $a[1:n, 1:dim]$ eingetragen. Zu jedem Element des letzteren gehört eine Reihe von Endempfängern, deren Adressen in $b[1:n, 1:dim, 1:dim]$ und deren Anzahlen in $lb[1:n, 1:dim]$ gespeichert werden. Schließlich wird noch m eingelesen und im Feld $ld[1:n]$ eingetragen. Jedem Ausgangsfifo ist ein Element von $v[1:n]$ zugeordnet, das jeweils die kleinste der Absendezeiten derjenigen Nachrichten enthält, die in diesen Fifo zu laden sind. Nachdem t mit allen in Frage kommenden Werten in v verglichen wurde und diese gegebenenfalls modifiziert worden sind, überprüft transfer, ob die anstehende Nachricht ausgegeben werden kann. Dazu wird festgestellt, ob die benötigten Ausgangsfifos hinreichend aufnahmefähig sind und ob die entsprechenden Elemente von v den Wert t haben. Falls diese Bedingung erfüllt ist, werden auch die Datenworte d_1, \dots, d_m eingelesen und die Nachricht in der Form $(s, t, l_k, e_1, \dots, e_{l_k}, m, d_1, \dots, d_m)$, $k=1, \dots, la[i]$, weitergeleitet, wobei die e_1, \dots, e_{l_k} die auf den jeweiligen Wegen erreichbaren Endempfänger adressieren. Die übrigen in transfer vorkommenden Namen bezeichnen Lauf- und Hilfsvariable. Bevor diese Prozedur in einem Netzknoten gestartet werde, sind alle Fifos zu löschen, wodurch die Elemente von fnl den Wert false und die von fsp ihren Maximalwert annehmen. Die Felder le und v sind mit 0 bzw. ∞ vorzubesetzen.

proc transfer=:

```

[α: for i from 1 by 1 to n do
  if le[i]=0 then
    if fnl[i] then
      o[i]:=fifoein[i] ; z[i]:=fifoein[i] ; le[i]:=fifoein[i];
      la[i]:=0 , for j from 1 by 1 to dim do lb[i,j]:=0 od ;
      for j from 1 by 1 to le[i] do
        w:=fifoein[i] ; q:=u[w] ;
        for k from 1 by 1 to la[i] do
          if a[i,k]=q then goto β fi
        od ;
        k:=la[i]+1 ; a[i,k]:=q ; la[i]:=k ;
β:   lb[i,k]:=lb[i,k]+1 ; b[i,k,lb[i,k]]:=w
      od ;
      ld[i]:=fifoein[i]
    else goto γ fi ;
  fi ;
  log:=true ;
  for j from 1 by 1 to la[i] do
    v[a[i,j]]:=min{z[i],v[a[i,j]]} ;
    log:=log $\wedge$ v[a[i,j]]=z[i] $\wedge$ fsp[a[i,j]] $\geq$ ld[i]+lb[i,j]+4
  od ;
  if log then
    le[i]:=0 ;
    for j from 1 by 1 to la[i] do
      v[a[i,j]]:= $\infty$  ; fifoaus[a[i,j]]:=o[i] ;
      fifoaus[a[i,j]]:=z[i] ; fifoaus[a[i,j]]:=lb[i,j] ;
      for k from 1 by 1 to lb[i,j] do

```

```
        fifoaus[a[i,j]]:=b[i,j,k]
    od ;
    fifoaus[a[i,j]]:=ld[i]
od ;
for j from 1 by 1 to ld[i] do
    w:=fifoein[i] ;
    for k from 1 by 1 to la[i] do fifoaus[a[i,k]]:=w od
od
fi ;
γ:od ; goto α ] .
```

Sei M eine obere Schranke für die Anzahl m der eigentlichen Datenworte, die in einer Nachricht enthalten sein können, dann hat die obige Prozedur für jeden Durchlauf eine Komplexität von der Ordnung $O(n \cdot \dim \cdot (\dim + M))$.

2.3 Speichereinheiten

Im letzten Paragraphen haben wir Speichermodule als die Einheiten eingeführt, welche die mit der Speicherung und Bereitstellung von Informationen zusammenhängenden Aufgaben selbständig bearbeiten. Unter besonderer Berücksichtigung der Schnittstellen zu den Verarbeitungseinheiten, die derart gewählt werden, daß die Zahl der notwendigen Datenübertragungen zwischen den einzelnen Modulen möglichst klein wird, wollen wir die beiden Typen von Speichereinheiten nun im Detail beschreiben.

2.3.1 Datenspeichermodule

Die Datensätze und privaten Dateien eines oder mehrerer Programme bzw. allgemein zugängliche Dateien sollen von Datenspeichermodulen bereitgehalten und verwaltet werden. Im wesentlichen bestehe ein DSM aus einem Arbeits- und eventuell mehreren Massenspeichern, besitze jedoch nur einen logischen Adreßraum. Die Verwaltung dieser Speicherhierarchie kann nach dem Prinzip des virtuellen Speichers erfolgen, so daß die Steuerung eines DSM's auch entsprechende Peripherietreiberfunktionen durchzuführen hat.

Um die Anzahl der Datenübertragungen zwischen DSM und Verarbeitungseinheit so gering wie möglich zu halten, sehen wir für letztere eine Speicher-Speicher- bzw. Multiregister-Architektur mit einem integrierten Scratch-Pad vor. In diesem finden häufig benötigte Skalare und Hilfsgrößen sowie kleinere Felder Platz. Werden die Scratch-Pad-Register am Anfang der Ausführung einer Programmeinheit, d.h. eines Blockes, einer Task oder einer Prozedur, aus dem DSM geladen und ihr Inhalt am Ende eventuell zurücktransferiert, dann sind Zugriffe zum DSM im wesentlichen nur noch zur Übertragung der Elemente größerer Datenstrukturen erforderlich. Das insbesondere nach Programmunterbrechungen notwendige Retten und Laden des Scratch-Pads kann durch geeignete Blocktransferbefehle veranlaßt werden.

Außerdem hat jedes DSM seine Daten zu sichern, indem es vor der Zuweisung an eine andere Task oder beim Eintreten einer Fehlersituation alle Informationen aus dem Arbeits- auf

einen nichtflüchtigen Massenspeicher verlagert.

Wir haben die Verarbeitungs- und Speichermodule als weitgehend selbständige Einheiten eingeführt. Um diese Eigenständigkeit aufrechtzuerhalten, ist es sinnvoll, in den von PSM's gehaltenen Instruktionen nur logische Datenadressen zu verwenden und deren Abbildung auf virtuelle Adressen den DSM's zu übertragen. In jeder Programmeinheit ist die Anzahl der Namen von Daten relativ klein, insbesondere verglichen mit dem Adreßvolumen eines DSM's. Ordnet man die Namen in einer Liste an, dann lassen sich die entsprechenden Platznummern, zu deren binärer Darstellung mithin wenige Bits ausreichen, als logische Adressen benutzen. Letztere können im DSM nun zur Adressierung eines kleinen Schreib-Lese-Speichers herangezogen werden, der die virtuellen Datenadressen enthält.

Nimmt der Dispatcher die Zuordnung eines DSM's zu einer bestimmten Aufgabe vor, dann ist dieser Adreßspeicher *a* neu zu laden. Weiterhin müssen in diesem Zusammenhang mehrere Felder mit Größen initialisiert werden, die die einzelnen Datenblöcke beschreiben, wie die Blocklängen *b* und die Schreibsperrern *s*, und Variable in den direkt zugreifbaren Speicher gebracht werden, die zur Überprüfung der Parameter benötigt werden, mit denen ein DSM zur Ausführung bestimmter Aufgaben beauftragt wird. Schließlich wird durch Setzen der logischen Variablen *zs*, die das Anforderungssignal maskiert, der Zugriff anderer Einheiten auf die

Daten des DSM's ermöglicht. Auf ein solches Signal ps eines anderen Moduls hin liest das DSM, sofern zs den Wert true hat, die übergebenen Parameter ein, interpretiert sie und prüft, ob die gewünschte Operation zulässig ist. Gegebenenfalls wird zs zurückgesetzt, um weitere Anforderungen bis zur Beendigung der laufenden Operation zurückzuweisen, und dann mit der selbständigen Durchführung der Aufgabe begonnen. Dabei kann es sich um die Übertragung von Datenblöcken, das Anlegen und Löschen von Dateien oder das Modifizieren von Schreibsperrern handeln. In den letzteren Fällen sind neben dem Belegen bzw. Freigeben von Speicherbereichen nur Eintragungen in den Feldern a, b und s vorzunehmen.

Die Datenspeichermodule können den Verarbeitungseinheiten eine Reihe von Routineaufgaben abnehmen. So wird es unter Umständen angebracht sein, Daten nicht in dem Format abzuspeichern, in dem sie verarbeitet werden. In diesen Fällen können die DSM's die Umformatierungen im Zusammenhang mit den Datenübertragungen durch die Abarbeitung entsprechender Funktionsprozeduren vornehmen und so die Programmlogik von derartigen Transformationen entlasten. Darüberhinaus führt die Übertragung solcher häufig vorkommender Teilaufgaben wie der Initialisierung von Blöcken mit Konstanten, dem Kopieren von Informationen oder dem sequentiellen Suchen bestimmter Werte in Datenfeldern an die DSM's zu einer erheblichen Reduzierung des Datentransfers und damit auch zu einer Entlastung der Verbindungseinheiten.

Da die Algorithmen zur Durchführung der bisher erwähnten

Operationen recht trivial sind, wollen wir die Arbeitsweise der Datenspeichermodule nur am Beispiel der Unterstützung dreier Datenstrukturen darstellen, nämlich der sequentiellen Datei, des Stacks und des Ringpuffers. Dazu müssen wir den den einzelnen Programmeinheiten zugeordneten Dateibeschreibungen die Felder v und h hinzufügen, die die Indizes der ersten bzw. letzten gültigen Datenworte in den Ringpuffern halten. In v werden im Falle der anderen beiden Adressierungsarten die fortlaufend zu inkrementierenden Satznummern bzw. die Stackpointer gespeichert. Als Parameter liest die folgende Routine von einem zugriffsberechtigten Modul m die Operationsauswahlvariable i, die logische Adresse n des zu bearbeitenden Puffers, die die Übertragsrichtung angezeigende Boole'sche Variable d und den Namen f einer Datentransformationsprozedur ein. Durch Aufruf der Prozedur check werden diese Parameter dann auf ihre Zulässigkeit hin überprüft. Sollte sie nicht gegeben sein, so wird der anfordernden Einheit mit der Anweisung "trigger zw" ein Zurückweisungssignal gesandt. Die unten angegebene Task endet nach Ausführung der gewünschten Operationen mit dem Setzen von zs, wodurch die vorher wirksam gewesene Zugriffssperre wieder aufgehoben wird. Die übrigen, aber bisher noch nicht definierten, Namen bezeichnen Hilfsvariable.

on ps[^]zs do

zs:=false , get(m,i,n,d,f) ;

if ¬ check(m,i,n,d,f) then goto a fi ;

case i in

[co Initialisierung des sequentiellen, Stack- oder
Ringpuffermodus co

v[n]:=h[n]:=0 ,

[co Sequentieller Zugriff co

if d as [n] v b [n] < v [n] then goto a fi ;

if d then get (w) ; cont a [n] + v [n] := f (w)

else w:=f(cont a[n]+v[n]) ; put (m,w) fi ;

v[n]:=v[n]+1 ,

[co Stackzugriff co

if d then if s [n] v b [n] < v [n] then goto a fi ;

get (w) ; cont a [n] + v [n] := f (w) ; v [n] := v [n] + 1

else if v [n] < 0 then goto a fi ;

v [n] := v [n] - 1 ; w:=f(cont a[n]+v[n]) ; put (m,w)

fi ,

[co Ringpufferzugriff co

if d then if s [n] v h [n] + 1 = v [n] v h [n] = b [n] ^ v [n] = 1 then goto a fi;

h [n] := if h [n] = b [n] then 1 else h [n] + 1 fi ;

get (w) ; cont a [n] + h [n] - 1 := f (w)

else if v [n] = 0 then goto a fi ;

w:=f(cont a[n]+v[n]-1) ; put (m,w) ;

if v [n] = h [n] then v [n] := h [n] := 0

else v [n] := if v [n] = b [n] then 1

else v [n] + 1 fi

fi

fi

out goto a esac ; goto β ;

α:trigger zw ; β:zs:=true

od

2.3.2 Das Befehlsbereitstellungssystem

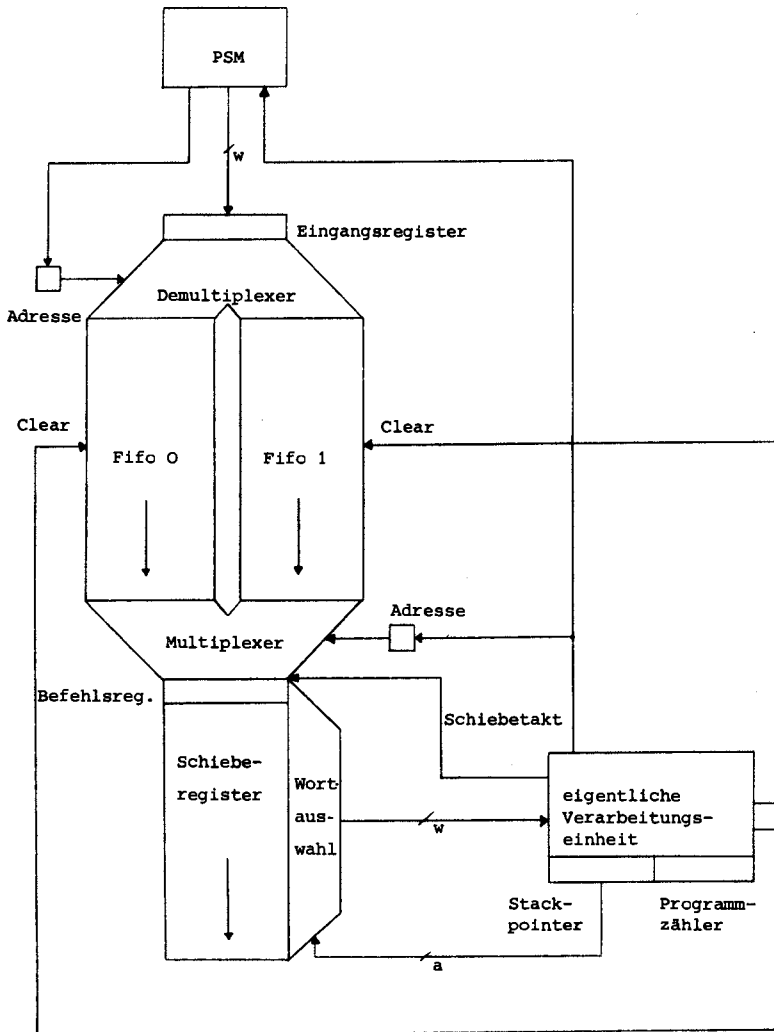
Wegen ihres engen Zusammenwirkens soll nun die Funktionsweise der Programmspeichermodule und der Befehlseingabesysteme der Verarbeitungseinheiten gemeinsam behandelt werden. Wir beginnen mit einer Analyse der Instruktionstypen, aus der wir die Aufgabenstellung und den Aufbau beider Komponenten entwickeln werden.

2.3.2.1 Die Struktur des Befehlsbereitstellungssystems

Bei konventionellen Rechnern wird die von einem in der CPU befindlichen Programmzähler gehaltene Adresse eines Befehlswortes zum Arbeitsspeicher übertragen, wenn dieses geladen werden soll. Für den Fall der rein sequentiellen Befehlsabarbeitung lassen sich die Befehlsadressentransfers durch die Integration von Programmzählern in die PSM's einsparen. Diese sequentielle Abfolge in der Ausführung von Instruktionen wird jedoch durch vier Befehlstypen aufgehoben:

1. Unbedingte Verzweigungen
2. Unterprogrammsprünge
3. Unterprogrammrücksprünge
4. Bedingte Verzweigungen.

Die Befehle der ersten drei Klassen haben dynamisch jeweils genau einen Nachfolger. Da alle zur Ausführung der entsprechenden Operationen benötigten Informationen in den PSM's zur Verfügung stehen, können die Befehle auch dort bearbeitet werden, wenn die PSM's um Dekodierer zur Befehls-



Figur 4. Befehlswörterzwischenspeichersystem der Verarbeitungseinheiten

interpretation und um Stacks zur Speicherung der Rücksprungadressen aus Unterprogrammen erweitert werden.

Die bedingten Verzweigungen haben dagegen zwei mögliche Nachfolgeinstruktionen, deren Auswahl von Verarbeitungsergebnissen abhängt. Wird ein bedingter Sprungbefehl erkannt, so kann das PSM der VE einige Nachfolgebefehle bzgl. beider möglicher Alternativen vorausschauend zur Verfügung stellen, um Verzögerungen in der Instruktionsausführung durch Warten auf die Übertragung des sich aus der Verarbeitung ergebenden Anschlußbefehls einer bedingten Verzweigung zu vermeiden. Dafür werden ein zweiter Programmzähler und Rückmeldeleitungen benötigt, über die dem PSM die Ausführung dieser Anweisungen und die Werte der Bedingungen mitgeteilt werden. Die nächsten Befehle werden dann von den Speicherplätzen gelesen, auf die die den eingetretenen Alternativen zugeordneten Programmzähler weisen.

Von den PSM's werden die Befehlsdaten zu den Verarbeitungseinheiten übertragen. Als Empfänger führen wir dort das in Figur 4 dargestellte Speichersystem zur vorübergehenden Aufnahme der Instruktionen ein, das wir nun vorstellen wollen.

Die vom zugeordneten PSM ausgegebenen Befehlswörter gelangen über eine Verbindungseinheit in das Eingangsregister dieses Speichersystems und von dort gemäß einer vom PSM eingestellten Adresse dm durch den Demultiplexer in einen der beiden First-in-first-out-Speicher $\text{fifo}[i]$, $i \in \{0,1\}$, wo sie sich an die Schlange der abzuarbeitenden Befehle anschließen.

Diese Fifos zwischen PSM und VE stellen sicher, daß die jeweils nächste Instruktion bei Bedarf sofort zur Verfügung steht. Die Übertragung der Befehle vom PSM zur VE erfolgt völlig asynchron zu deren späterer Ausführung. Bis zu einem gewissen, durch die Kapazität der Fifos gegebenen, Grade können durch vorübergehende Blockierungen des Übertragungsbusses oder durch Nachladen von Programmteilen von Massenspeichern verursachte Verzögerungen derart ausgeglichen werden. Nach Bearbeitung einer Instruktion erzeugt die VE einen Schiebetakt so, wodurch der nächste Befehl aus dem jeweils mit ma adressierten Fifo durch den Multiplexer in das Befehlsregister br gelangt. Die von der VE eingestellte Multiplexeradresse ma bleibt solange konstant, wie keine bedingten Verzweigungen auszuführen sind. Hält nun das Befehlsregister eine Instruktion dieser Klasse, dann können die ihr sequentiell folgenden Befehle dem zur Zeit adressierten Fifo entnommen werden, während der andere Fifo die ersten Befehle der alternativen Instruktionssequenz enthält. Es ist Aufgabe des PSM's, nach Erkennen einer bedingten Verzweigung die beiden Fifos entsprechend zu füllen. Die maximale Anzahl der Befehle, die dabei übertragen werden, hängt entweder von der Fifokapazität oder vom Auftreten solcher Instruktionen ab, vor deren Ausgabe die Ausführung der bedingten Verzweigung abgewartet werden muß. Auf Grund der eingetretenen Bedingung, deren Wert im Status-Register v gespeichert wird, stellt die VE bei der Bearbeitung einer bedingten Verzweigung die Multiplexeradresse neu ein und löscht den Fifo, der die nicht benötigten In-

struktionen enthält. Durch das Signal psm wird dem PSM angezeigt, daß eine neue Multiplexeradresse ansteht. Mit dieser Information werden einer der beiden Befehlszähler für die Weiterarbeit ausgewählt und die Demultiplexeradresse für die Eingabe der nächsten Befehlskörper neu gesetzt.

An das Befehlsregister schließt sich ein Schieberegister $sr[0:n]$ mit $n=2^m-1$, $m \in \mathbb{N}$, und $br=sr[n]$ an, in dem sich i.a. die abgearbeiteten Instruktionen noch eine gewisse Zeit lang befinden. Der Sinn dieses Speichers jedoch ist, alle Befehle von Programmschleifen bzw. Schleifennestern bis zu einer durch die Kapazität des Registers gegebenen Länge gleichzeitig in der VE zur Verfügung zu halten. Solche Programmschleifen werden recht häufig zur Formulierung von Algorithmen angewandt. Da die einzelnen Befehle nicht bei jedem Schleifendurchlauf, sondern nur einmal, vom PSM zur VE transferiert werden müssen, lassen sich erhebliche Einsparungen bei der Befehlsdatenübertragung erzielen. Die Instruktionssequenz eines Schleifennestes habe die folgende Form:

Schleifenstartbefehl

α :erster Befehl	}	zur Auswertung der Schleifenabbruchbedingung
·		
·		
letzter Befehl		

innere Befehle der Schleife

ω :unbedingter Sprung nach α .

Dabei werde im Schleifenstartbefehl die Anzahl s_1 der folgenden Wörter einschließlich des Rücksprungs angegeben, die natürlich die Kapazität des Schieberegisters nicht über-

steigen darf. Werden in der Schleife Unterprogramme aufgerufen, dann muß deren Länge im Startbefehl mit berücksichtigt werden. Letzterer wird sowohl vom PSM wie auch von der VE benötigt: bedingte und unbedingte Verzweigungen, die aus dem Bereich der Schleife nicht hinausführen, werden unbearbeitet zusammen mit den anderen Befehlen zur VE weitergeleitet, die entsprechend der Angabe im Startbefehl Schieberegister erzeugt, um die folgenden Instruktionen in das Schieberegister zu laden. Damit die Ausführung von Schleifen im Rahmen des beschriebenen Zwischenspeichersystems möglich ist, darf ein Schleifennest höchstens zwei alternative

- Nachfolgebefehle haben. Dies stellen die beiden Bedingungen
- (1) Eine Schleife darf keine (ohnehin sinnlose) unbedingten Sprünge enthalten, die aus ihr hinausführen.
 - (2) Eine Schleife darf höchstens eine bedingte Verzweigung enthalten, die aus ihr hinausführt und nicht den auf das Nest direkt folgenden Befehl adressiert.

sicher. Bei der Bearbeitung des Startbefehls werde auch der Stackpointer sp der VE initialisiert, der zur Auswahl des nächsten auszuführenden Befehls im Schieberegister dient. Bis zum Verlassen der Schleife ist er dann nach jeder abgearbeiteten Instruktion wie ein Programmzähler zu modifizieren. Ist der bei Beendigung der Schleife vom Stackpointer gehaltene Wert gleich $n+1$, der Anzahl der Speicherplätze des Schieberegisters, so steht der nächste Befehl im zu dieser Zeit adressierten Fifo.

2.3.2.2 Das Befehlszwischenspeichersystem der Verarbeitungseinheiten

Im vorausgegangenen Abschnitt haben wir bereits den Aufbau des in den Verarbeitungseinheiten vorgesehenen Befehlszwischenspeichersystems dargestellt und seine Funktionsweise umrissen. Diese Ausführungen sollen jetzt durch die Angabe einer Routine präzisiert werden, welche den Ablauf der Befehlsbearbeitung in den VE's und die Steuerung des Zwischenspeichersystems beschreibt. Dabei genügt es, diejenigen Operationen bei der Instruktionsausführung näher zu betrachten, welche der Verwaltung der Befehlsdaten dienen.

In den Verarbeitungseinheiten befindet sich jeweils auch ein Programmadreßzähler pz. Diese dienen dazu, für den Fall von Unterbrechungen die vollständigen Stati der jeweiligen Programmausführungen in den VE's zur Verfügung zu haben. Zur richtigen Fortschaltung der Programmzähler sind demnach auch die Befehle, die bereits in den PSM's bearbeitet werden, zu den VE's weiterzuleiten. Um auf Unterprogrammrücksprungadressenstacks in den VE's verzichten zu können, müssen die PSM's die Rückkehradressen zusammen mit den entsprechenden Rücksprungbefehlen zu den VE's übertragen, wo diese dann genauso wie Unterprogrammaufrufe als einfache Sprünge behandelt werden. Möchte man die Möglichkeit haben, innerhalb von Schleifen Unterprogramme zu verwenden, deren Code sich ebenfalls im Schieberegister befinden soll, dann benötigt man in den VE's ebenfalls Stacks für pz und sp, für die jedoch eine

sehr geringe Kapazität ausreicht. Weiterhin sind ein spezieller Unterprogrammsprungbefehl, der statt einer absoluten Adresse ein Adresseninkrement enthalten sollte, das zu pz und sp addiert werden kann, und ein entsprechender Rückkehrbefehl erforderlich.

Die im folgenden angegebene Steuerungsroutine werde durch ein Signal ds des Dispatchers gestartet. Zuerst werden einige Initialisierungen vorgenommen: die internen und Scratch-Pad-Register werden geladen und die beiden Fifos gelöscht, der Stackpointer sp wird auf n und die Multiplexeradresse ma auf 0 gesetzt. Die Boole'schen Variablen v und l zeigen an, ob eine bedingte Verzweigung ausgeführt werden muß bzw. ob gerade keine Schleife bearbeitet wird. Diese Größen werden mit false bzw. true vorbesetzt. Das Programm behandelt die Größen ma und v sowohl als ganzzahlige als auch als logische Variablen. Der Aufruf der durch

proc in:=[while fl do od ; trigger so]

definierten Prozedur "in", die das Leer-Signal fl des mit ma adressierten Fifos solange abfragt, bis Instruktionen angekommen sind, bringt ein Befehlswort in das Schieberegister. Dieses wird dann in das erste Wort bhr[1] des Befehlshalteregisters transferiert, aus dessen Inhalt die Funktionsprozeduren bc und bl den Befehlscode bzw. die Befehlslänge bestimmen. Im Falle, daß gerade keine Schleife in Bearbeitung ist, werden danach die restlichen Befehlsworte eingelesen. Ansonsten ist der nächste Befehl schon im Schieberegister verfügbar. Dann folgt die eigentliche Instruktionsausführung, wozu auch die Modifikation von pz und sp

gehört. Bei einigen, für unsere Betrachtung wesentlichen, Befehlen haben wir unten die Operationen angegeben, die bei ihrer Bearbeitung durchzuführen sind. So werden nach einem Schleifenstartbefehl u.a. die Statusvariable 1 zurückgesetzt und alle zur Schleife gehörenden Befehle ins Schieberegister gebracht. Bei Verzweigungen wird das aus dem Befehl entnommene Inkrement Δ zu sp und pz addiert. Schließlich werden nach bedingten Verzweigungen außerhalb von Schleifen und solchen, die aus letzteren hinausführen, jeweils ein Fifo gelöscht, die Multiplexeradresse modifiziert und dem angeschlossenen PSM das Signal psm gesandt.

on ds do

sp:=n , ma:=0 , v:=false , l:=true , clear fifo[0],fifo[1],

Lade das Scratch-Pad und die internen Register ;

α :in ;

β :bhr[1]:=sr[sp] ; pz:=pz+b1 ;

for i from 2 by 1 to b1 do

if 1 then in else sp:=sp+1 fi ; bhr[i]:=sr[sp]

od ;

if \neg 1 then sp:=sp+1 fi ;

if bc $\hat{=}$ Haltbefehl then Beende die Befehlsausführung

elsif bc $\hat{=}$ Schleifenstartbefehl then

Sichere die im Befehl stehende Schleifenlänge s1 ;

sp:=n-s1+1 , l:=false ,

for i from 1 by 1 to s1 do in od

elsif bc $\hat{=}$ Schleifenrücksprung then sp:=sp-s1 , pz:=pz-s1

elsif bc $\hat{=}$ bedingte Verzweigung then

Führe den Befehl aus und lege das Ergebnis der Bedingung

```

in v ab ;

i:=if v then Δ else 0 fi ; pz:=pz+1 ;

if l then ma:=ma⊕v ; trigger psm , clear fifo[¬ma]
    else sp:=sp+1 ;
        if sp<0vsp⊗n+1 then
            if sp⊗n+1 then ma:=¬ma fi ;
            sp:=n , l:=true , trigger psm , clear fifo[¬ma]
        fi
    fi

else Führe den Befehl aus, modifiziere ggf. pz und sp
fi ; goto if l then α else β fi
od .

```

2.3.2.3 Die Funktionsweise von Programmspeichermodulen

Aus der vorangegangenen Beschreibung des Befehlseingabesystems der Verarbeitungseinheiten sind bereits die wesentlichen Funktionen deutlich geworden, die von Programmspeichermodulen durchzuführen sind. Neben den schon oben genannten vier Befehlstypen müssen die PSM's Haltbefehle, nach denen die weitere Instruktionsausgabe einzustellen ist, und Schleifenstartbefehle erkennen und gesondert behandeln. Die Einhaltung der für den letzteren Befehlstyp geltenden Einschränkungen haben die Compiler bzw. der Assembler zu überprüfen.

Die weiter unten angegebene, in den PSM's ablaufende Befehlsausgaberoutine werde durch ein vom Dispatcher erzeugtes Signal ds gestartet und beende ihre Tätigkeit nach der Über-

tragung eines Haltbefehls zur angeschlossenen VE. Die gleiche Operation möge auch auf Grund eines entsprechenden Signals dieser VE erfolgen. Um die Auftragsvergabe an ein PSM zu synchronisieren, brauchen dann nur noch etwaige während aktiver Phasen der Befehlsausgaberoutine ankommende Startsignale des Dispatchers zurückgewiesen zu werden.

Zum Schutze der Programme unterscheiden wir zwei Zustände, in denen sich Rechnersysteme befinden können: den Test- und den Normalbetriebsmodus. Der Letztere sei dadurch gekennzeichnet, daß keine Programme in die PSM's übertragen werden können. Im einfachsten Falle sind alle Befehle eines PSM's in Nur-Lese-Speichern abgelegt. Ansonsten müssen die benötigten Programme vor der Inbetriebnahme von Prozeßrechnerinstallationen im Testmodus in die PSM's geladen werden. Dazu werden für jedes Programm die Parameter Anfangsadresse und Programmlänge eingelesen. Anschließend können die Instruktionen wortweise eingegeben und an die entsprechenden Stellen im Adreßraum geschrieben werden. Mithin erübrigen sich sowohl für die Befehlseingabe als auch für die Befehlsausgabe Befehlsadreibusse.

Auf Umladen kann i.a. verzichtet werden, wenn die Betriebsroutinen der einzelnen Funktionsmodule dort in Hard- oder Firmware realisiert werden. Somit ist eine Rechenanlage im Normalbetriebsmodus nach dem Einschalten sofort funktionsfähig.

Ist nicht der ganze Adreßraum eines PSM's direkt ansprechbar, dann werde die Verwaltung der entsprechenden Speicherhierarchie nach dem Prinzip des virtuellen Speichers vorge-

nommen. Die Adressierung nicht im Arbeitsspeicher verfügbarer Befehle führt somit zum Nachladen von Programmseiten von dem an das PSM angeschlossenen Massenspeicher. Deshalb sind von einem PSM gegebenenfalls auch Peripherietreiberfunktionen durchzuführen. Wegen des im Vergleich mit ihren Datensätzen relativ geringen Umfangs von Prozeßrechnerprogrammen bzw. Programmgruppen und der in PSM's vorliegenden Geschwindigkeitsanforderungen, empfiehlt es sich, die Massenspeicher für diese Anwendung aus als Halbleiterbauelemente zur Verfügung stehenden Schieberegistern aufzubauen.

Zur präzisen Beschreibung der Arbeitsweise von PSM's, die wir nun wieder in Form eines Programmes vornehmen wollen, sehen wir jedoch von diesen Speicherhierarchieverwaltungsfunktionen ab, um das Wesentliche deutlich werden zu lassen.

Nach dem Start der Routine durch das Dispatchersignal ds wird die Nummer ve der zu bedienenden VE und die Anfangsadresse anf des auszugebenden Programmes im Adreßraum der VE eingelesen. Diese Adresse wird dem ersten der beiden Befehlsadressenzähler $pz[0:1]$ zugewiesen. Entsprechend erhält der Programmzählerindex p den Anfangswert 0. Als weitere Initialisierungen setzen die VE und das PSM ihre Multiplexeradresse ma und die Demultiplexeradresse dm jeweils auf 0 und werden das die Ausführung bedingter Verzweigungen anzeigende Signal psm sowie der mit den Prozeduren $push$ und pop zu programmierende Rückkehradressenstack sp gelöscht. Die eigentliche Befehlsbearbeitung finde

erst dann statt, wenn der durch dm in der VE adressierte Fifo mittels des Signals fab anzeige, daß er für mindestens einen Befehl maximaler Länge aufnahmebereit sei. Dieses Signal habe außerdem die Eigenschaft, daß es nach Annahme des Wertes false erst dann wieder true werde, wenn der Fifo vorher bis zu einem gewissen Grade entleert wurde. Aus dem Arbeitsspeicher des PSM's werden die einzelnen Instruktionen in das Befehlshalteregister bhr[1:max] gebracht, dessen Länge der des längsten Befehls entspreche, und dann von dort zur angeschlossenen VE übertragen. Durch Aufruf der Funktionsprozeduren bc, bl, adr, Δ , sl und v werden aus dem Inhalt von bhr der Befehlscode, die Befehlslänge, eine absolute Adresse, ein Adresseninkrement, die Anzahl der zu einer Schleife gehörenden Befehlsworte bzw. eine Boole'sche Größe abgeleitet, die anzeigt, ob eine Schleife eine aus ihr hinaus und nicht auf den nächsten Befehl führende bedingte Verzweigung enthält. Nachdem das PSM den gerade benutzten Programmzähler auf die Adresse des nächsten Befehls gesetzt hat, wird der soeben übertragene interpretiert. Mit einem Haltbefehl endet die Arbeit der Routine. Bei Sprüngen wird die Adresse des nächsten zu holenden Befehls neu berechnet. Nach dem Speichern der Rückkehradresse im Stack wird auch bei einem Unterprogrammsprung der Programmzähler neu geladen. Liegt ein Unterprogrammrücksprung vor, dann wird die zuletzt eingegebene Adresse dem Stack entnommen und auch zur VE übertragen, um das dortige Programmadressregister zu redefinieren. Die sl einem Schleifenstartbefehl folgenden Worte werden uninterpretiert an die VE weitergeleitet.

Nach einer bedingten Verzweigung oder nach einer Schleife, die einer solchen entspricht, was durch v=true gekennzeichnet ist, wird die Anfangsadresse der alternativen Befehlssequenz berechnet und dem gerade nicht benutzten Programmzähler zugewiesen. Sofern sie aufnahmebereit sind, werden dann die beiden Fifos der VE abwechselnd mit den Worten der alternativen Befehlsströme gefüllt, bis das Signal psm gesetzt wird. Letzteres wird quittiert und der Wert der Multiplexeradresse wird zur Adressierung eines Programmzählers und des Demultiplexers übernommen. Da ihre Bearbeitung unmöglich ist bzw. zu Schwierigkeiten führen könnte, wird das Füllen der Fifos auch dann eingestellt, wenn in einem der Befehlsströme ein Halt-, bedingter Verzweigungs-, Schleifenstart- oder ein Unterprogramm(rück)sprungbefehl erkannt wird.

on ds do

get(ve) ; get(anf) , p:=dm:=0 , clear st , psm:=false ;

pz[p]:=anf ;

a:while ~ fab do od ;

bhr[1]:=cont pz[p] ;

for i from 2 by 1 to bl do

pz[p]:=pz[p]+1 ; bhr[i]:=cont pz[p]

od ;

for i from 1 by 1 to bl do put(ve,bhr[i]) od ,

pz[p]:=pz[p]+1 ;

if bc^ΔHaltbefehl then Beende die Programmausgabe

elsif bc^Δunbedingte(r) Verzweigung (Sprung) then

pz[p]:=if bc^Δrelative Verzweigung then pz[p]^Δ else adr fi

```
elsif bc $\hat{=}$ Unterprogrammsprung then push(pz[p]) ; pz[p]:=adr
elsif bc $\hat{=}$ Unterprogrammrücksprung then
    pz[p]:=pop ; put(ve,pz[p])
elsif bc $\hat{=}$ bedingte Verzweigung then pz[ $\neg$ p]:=pz[p]+ $\Delta$  ; goto  $\beta$ 
elsif bc $\hat{=}$ Schleifenstartbefehl then
    for i from 1 by 1 to s1 do
        bhr[1]:=cont pz[p] ; pz[p]:=pz[p]+1 ;
        while  $\neg$  fab do od ; put(ve,bhr[1])
    od ;
    if v then pz[ $\neg$ p]:=pz[p]+ $\Delta$  ; goto  $\beta$  fi
fi ; goto  $\alpha$  ;
 $\beta$ :sw:= $\delta$  ;
 $\gamma$ :if psm then p:=dm:=ma , psm:=false ; goto  $\alpha$  else goto sw fi ;
 $\delta$ :if  $\neg$  fab then goto  $\gamma$  fi ;
bhr[1]:=cont pz[p] ;
if Befehle{Halt-, Unterprogramm(rück)sprung-, bedingter
    Verzweigungs-, Schleifenstartbefehl} then sw:= $\gamma$ 
else
    for i from 2 by 1 to b1 do
        pz[p]:=pz[p]+1 ; bhr[i]:=cont pz[p]
    od ;
    for i from 1 by 1 to b1 do put(ve,bhr[i]) od ,
    pz[p]:=pz[p]+1 ;
    if bc $\hat{=}$ unbedingte(r) Verzweigung (Sprung) then
        pz[p]:=if bc $\hat{=}$ relative Verzweigung then pz[p]+ $\Delta$  else adr fi
    fi ;
    p:=dm:= $\neg$ p
fi ; goto  $\gamma$ 
od .
```

2.4 Verarbeitungseinheiten

In konventionellen Prozeßrechnern werden alle System- und Anwendertasks von einer zentralen Verarbeitungseinheit ausgeführt. Dagegen sehen wir, wie in Figur 1 eingezeichnet, i.a. mehrere VE's vor.

Auf Grund der Kostenentwicklung besteht keine ökonomische Notwendigkeit mehr dafür, daß die Zahl der VE's wesentlich geringer sein sollte als die der Speichereinheiten und mithin der Task-Gruppen. Im Gegenteil kann es aus Sicherheitsgründen sogar wünschenswert sein, mehr Prozessoren als Speichermodule zur Verfügung zu haben.

Dabei ist allerdings nicht an symmetrische Mehrprozessor-systeme gedacht, sondern an Rechner mit einer Hierarchie von Prozessoren verschiedener Leistungsstufen. Die einzelnen Leistungsklassen sollten sich in den Verarbeitungsgeschwindigkeiten und den Mächtigkeiten der Befehlssätze unterscheiden. Allen Prozessoren sollten jedoch die Struktur und die Befehlscodes gemeinsam sein. Weniger leistungsfähige und damit billigere Einheiten brauchen nur echte Teilmengen des gesamten Befehlssatzes verarbeiten zu können. Es reicht aus, wenn die VE's der untersten Leistungsklasse Befehle bis etwa zur Komplexität der Festkommaaddition ausführen können. Da Festkommanultiplikationen und -divisionen sowie Gleitkomma- und Vektoroperationen usw. in der Prozeßdatenverarbeitung relativ selten vorkommen, brauchen nur recht wenige und entsprechend schnelle VE's diese Instruktionen interpretieren und ausführen zu können. Wir werden

später sehen, daß es eine wesentliche Aufgabe des Dispatchers ist, die leistungsfähigeren Prozessoren vorübergehend mit den PSM's zu verbinden, deren Programme sehr schnell abgearbeitet werden müssen bzw. die Ausführung komplexer Operationen erfordern. Dabei ist es möglich, bereits beim Taskstart geeignete Prozessoren zuzuordnen oder diese erst dann anzufordern, wenn durch Unterbrechungen angezeigt wird, daß gewisse Befehle nicht bearbeitet werden können.

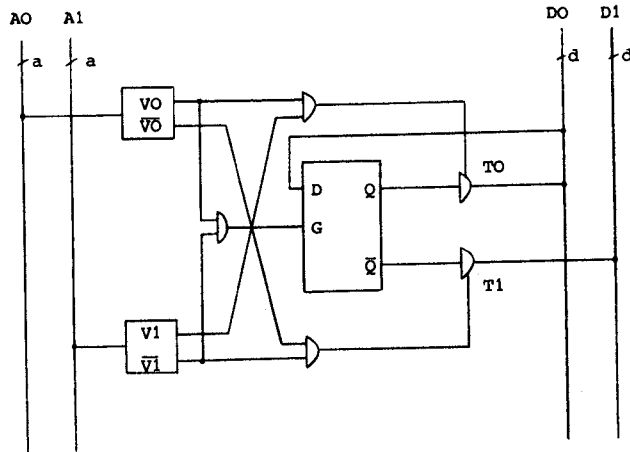
2.4.1 Die Registerstruktur der Verarbeitungseinheiten

Den internen Aufbau der Verarbeitungseinheiten haben wir, soweit es die Befehlseingabe und -bearbeitungssteuerung anbelangt, bereits oben behandelt. Darum brauchen wir an dieser Stelle nur die Beschreibung der internen Registerstruktur nachzutragen.

Es wurde schon darauf hingewiesen, daß die VE's integrierte Scratch-Pads besitzen sollen. Dieser Ansatz wurde gewählt, um eine größere Anzahl häufig benötigter Daten in den VE's zur Verfügung zu haben und derart sehr viele Übertragungen zu und von Datenspeichereinheiten einzusparen.

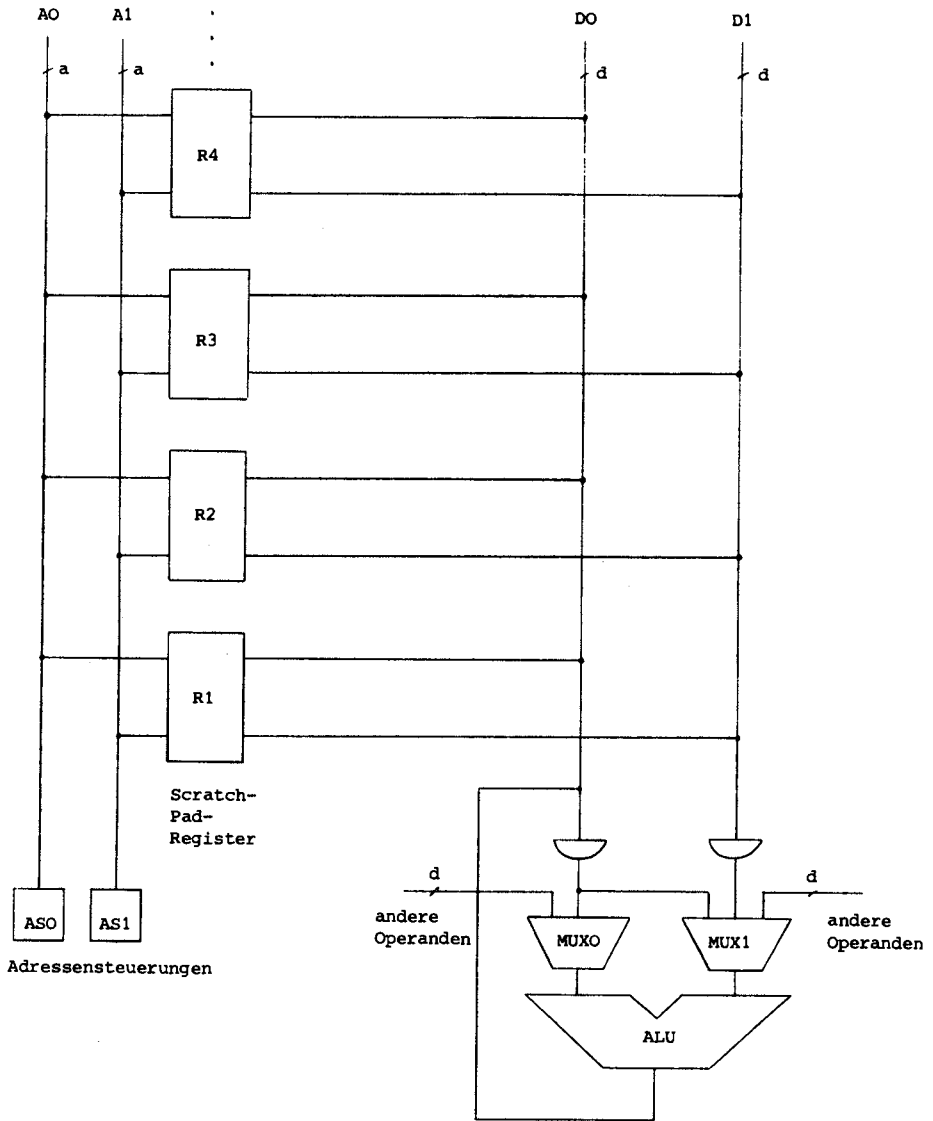
Die Struktur des hier vorgestellten Scratch-Pads erlaubt es, auf die Operanden zweistelliger Operatoren parallel zuzugreifen. Da i.a. eine recht geringe Zahl von Scratch-Pad-Registern ausreichen wird, z.B. 64 oder 256, die mit 6 bzw. 8 Bits adressiert werden können, lassen sich hier effektiv Dreiadreibefehle anwenden.

Den prinzipiellen Aufbau eines solchen Registers zeigt



Figur 5. Aufbau eines Scratch-Pad-Registers

Figur 5. Jede Zelle ist an zwei Adreßbusse AO und A1 der Breite a und an zwei Datenbusse DO und D1 der Breite d angeschlossen. Mithin kann auf zwei Operanden parallel zugegriffen werden. Die zeitlich später zur Verfügung stehenden Resultate werden über DO in das Ergebnisregister geschrieben. Dazu wird dessen Adresse auf beide Adreßbusse geschaltet, wodurch sich eine Schreib/Lese-Leitung ergibt. Um den Inhalt einer Speicherzelle auf beiden Bussen verfügbar zu machen, sind daher besondere Maßnahmen erforderlich (vgl. Figur 6). Die Adressendekodierung erfolgt jeweils durch zwei Vergleiche VO, V1. Da diese immer einem Register mit fester Adresse zugeordnet sind, reduzieren sie sich jeweils auf eine Reihe von Invertern und ein Ungatter. Durch Verzicht auf eine mögliche Adresse,



Figur 6. Struktur eines Rechnerkerns mit Scratch-Pad

z.B. 0, können Taktleitungen eingespart werden, wenn sie dazu verwendet wird, den nicht aktiven Zustand eines Adreßbusses zu kennzeichnen. Über die drei Undgatter in Figur 5 werden Lesegates G und die Bustreiber T0, T1 der jeweils d Speicherflipflops eines Registers angesteuert. Die Zusammenschaltung des Scratch-Pads mit den übrigen Komponenten einer VE, mit denen direkter Datenaustausch stattfindet, zeigt Figur 6.

2.5 Peripherieadapter

Der Name "Peripherieadapter" bringt bereits zum Ausdruck, daß es die Aufgabe solcher Funktionseinheiten ist, den Datentransfer zwischen einer Rechanlage und den unterschiedlichsten peripheren Geräten - Daten-, Datenend- und Prozeßperipherie sowie spezielle Hardware und weitere Rechner - abzuwickeln und die Anpassungen an diese Komponenten mit Hilfe entsprechender Programme vorzunehmen. Dadurch erübrigt es sich, für jeden einzelnen Fall ein besonderes Interface zu entwickeln.

Rechnerseitig werden die PA's wie alle anderen Module an Verbindungseinheiten angeschlossen, während der peripherie-seitige Datenaustausch über nach Möglichkeit standardisierte Außenschnittstellen erfolge. Die Adapter haben die Struktur einfacher Rechner und besitzen eigene Schreib/Lesespeicher zur Aufnahme beim Betrieb benötigter Variablen und zur Pufferung von Datenblöcken.

Die Arbeitsweise der PA's ist der der DSM's sehr ähnlich:

insbesondere wird das Einlesen von Parametern durch ein Signal veranlaßt, das mit einer logischen Variablen des Adapters maskiert ist. Die Parameter werden interpretiert und auf Zulässigkeit hin überprüft. Bei Nichtausführbarkeit der gewünschten Operation wird an das aufrufende Modul ein Zurückweisungssignal gesandt. Ansonsten beginnt mit dem Laden der zu übertragenden Daten, die von einer Verbindung- bzw. Peripherieeinheit eingelesen werden, in den Pufferspeicher die eigentliche Arbeit des Adapters. Falls gewünscht kann nun eine Darstellungstransformation dieser Daten durchgeführt werden, bevor sie schließlich an ihren Bestimmungsort transferiert werden. Die Kommunikation mit den anderen Modulen eines Rechnersystems über Verbindungseinheiten erfolgt auch hier mittels der Prozeduren put und get. Dagegen müssen in den PA's zum Verkehr mit der Peripherie jeweils spezielle Peripherietreiberrouninen implementiert werden, die die eigentliche Hardwareprogrammierung vornehmen, den Datenfluß überwachen, Fehler erkennen und nach Möglichkeit beheben. Als weitere Funktionen führt die Steueroutine eines PA's auf Grund entsprechender Kommando- parameter die Ausgabe von Kontrolldaten an die Peripherie und von Statusinformationen an Verarbeitungseinheiten durch. Den Abschluß von Operationen zeigen die PA's den aufrufen- den Modulen durch Signale an, die zusammen mit den oben erwähnten logischen Variablen gesetzt werden.

Aus dem bisher Gesagten folgt, daß nur die Kenntnis der Außenschnittstellen des Adapters und das Schreiben jeweils eines Steuer- und Peripherietreiberprogrammes erforderlich

sind, wenn ein Gerät über einen PA an einen Rechner angeschlossen werden soll. Da alle PA's in der gleichen Weise mit den übrigen Systemkomponenten kommunizieren, wird auf deren Seite nur ein Unterprogramm zur Durchführung von Ein- und Ausgaben gebraucht. Kann dieses direkt in der Anwendersoftware aufgerufen werden bzw. werden die E/A-Anweisungen einer höheren Programmiersprache vom Compiler in solche Aufrufe umgesetzt, dann ist der "Durchgriff" zur Hardware und zur Treiberoutine ohne Eingriffe in Betriebssysteme und Compiler wie bei herkömmlichen Rechnern möglich.

2.6 Strukturvergleich mit Einprozessorsystemen

Um die Vorteile des hier vorgeschlagenen Konzeptes deutlich zu machen, wollen wir jetzt letzteres mit einer konventionellen Einprozessorstruktur hinsichtlich der Übertragungskomplexität, der Möglichkeit zur Parallelarbeit sowie Kosten und Zuverlässigkeit vergleichen und auf einige Eigenschaften hinweisen, die das Konzept gegenüber klassischen Prozeßrechnern auszeichnen.

2.6.1 Vergleich der Übertragungskomplexität

Bei der von-Neumann-Struktur muß jedes zu bearbeitende Befehlswort und seine Adresse übertragen werden. Demgegenüber sind keine Befehlsadressentransfers zu PSM's erforderlich und die Anzahl n der zwischen einem PSM und der zugeordneten VE zu übertragenden Befehlswörter ist

$$n = p \cdot (m - (q-1) \cdot i + j + r \cdot k) + 1 ,$$

wobei m die Anzahl der auszuführenden Befehle,
 i die Anzahl der Instruktionen von im Befehlsschieberegister
Platz findenden Schleifen,
 j die Anzahl ihrer Startbefehle,
 q ihre mittlere Durchlaufhäufigkeit,
 k die Anzahl der bedingten Verzweigungen außerhalb dieser
Schleifen bzw. der Schleifen mit zwei möglichen Folge-
befehlen,
 r die mittlere Anzahl der alternativen, nicht benötigten
Befehlswörter,
 p die mittlere Anzahl der Wörter einer Instruktion und
 l die Anzahl der zusätzlich zu Übertragenden Unterprogramm-
rückkehradressen sei.

Diesen n stehen also insgesamt $2 \cdot p \cdot m$ Übertragungen bei der
konventionellen Struktur gegenüber, vorausgesetzt, beide
Rechner haben den gleichen Befehlssatz und das gleiche
Befehlsformat. Da aber zur namentlichen und Scratch-Pad-
Adressierung weniger Bits erforderlich sind als zur Angabe
absoluter Speicheradressen, kann i.a. der in die Berechnung
von n eingehende Faktor p gesenkt werden. Zusammen mit der
Dreiadreßstruktur wird bei Ausführung von Operationen des
Typs $a := b \cdot c$ eine Reduktion der Befehlswörter um den Faktor
3 und der Übertragungen um den Faktor 6 gegenüber Ein-
und Zweiadreßmaschinen erzielt.

Bei Zugriffen auf in einem Datenspeichermodul gespeicherte
Daten können Einsparungen nur hinsichtlich der Zahl der

Adressentransfers erreicht werden. Im günstigsten Fall läßt sich diese Zahl wegen der Verwendung von Datennamen in Dreiadreßbefehlen auf ein Drittel senken. Übertragungen können in erheblichem Umfang vermieden werden, wenn die DSM's in der Lage sind, einfache und häufig wiederkehrende Aufgaben selbständig zu erledigen. Als Beispiele seien die Initialisierung von Feldern mit Konstanten, das Suchen nach Feldelementen mit bestimmten Werten und das Umspeichern von Datenblöcken erwähnt. Zum Aufruf dieser Funktionen genügt dann die Übergabe weniger Parameter.

Zum quantitativen Vergleich der Anzahl von Datenübertragungen betrachten wir nun ein Programm, das n mal Daten im Speicher referenziert und q mal unterbrochen wird. In einem Rechner mit r Registern sind also

$$2n+4qr$$

Adressen und Daten zu übertragen. Besitzt eine VE $s+r$ Scratch-Pad-Register, dann können wir davon ausgehen, daß die Zahl der Zugriffe auf Daten n nicht übersteigt. Sei p die Wahrscheinlichkeit, daß sich ein Datenwort im Scratch-Pad befindet, und j die mittlere Anzahl der mit einem Adreßwort zum DSM übertragenen Datennamen, so erfordert die Programmausführung

$$n(1-p) \cdot (1 + \frac{1}{j}) + 2q(s+1)$$

Bustransfers zum DSM unter der Voraussetzung, daß letzteres zum Laden bzw. Retten des Scratch-Pads jeweils nur ein Parameterwort benötigt. Die Wahrscheinlichkeit p wird i.a. recht groß sein, da die meisten Zugriffe auf relativ wenige Größen, z.B. Schleifenvariable, erfolgen, die dauernd im

Scratch-Pad resident sein können.

2.6.2 Parallelität bei der Taskbearbeitung

Die hier beschriebene Rechnerstruktur ermöglicht es, daß mehrere Funktionseinheiten parallel an einer Aufgabe arbeiten können. Unter letzterer wollen wir eine Task, jedoch nicht eine Vektoroperation verstehen, bei der eine gewisse Anzahl gleichartiger Verknüpfungen zur selben Zeit ausgeführt werden.

Der Grad der Parallelität hängt natürlich von der jeweiligen Programmsituation ab. Im günstigsten Falle jedoch sind Module fünf verschiedener Typen gemeinsam mit der Bearbeitung einer Task beschäftigt. Während nämlich eine VE das Programm ausführt, verwalten die Speichermodule ihre Speicherhierarchien und Peripherieadapter tauschen Daten mit externen Geräten aus. Parallel dazu kann schließlich die NÜ zur Synchronisation der Task mit anderen Prozessen Meldungen übermitteln. Auch die Bearbeitung von (Unter-) Programmsprüngen und die Inkrementierung des Programmzählers im PSM sowie die Generierung von Adressen und die Darstellungstransformation im DSM erfolgen weitgehend gleichzeitig zur Programmausführung in der VE. Im Scratch-Pad können die Argumente dyadischer Operatoren zusammen verfügbar gemacht werden. Dadurch wird der Grad der Parallelität ebenfalls erhöht. Entsprechend der Anzahl parallel arbeitender Module und der zur gleichen Zeit vorgenommenen Übertragungen zwischen PSM und VE bzw. zwischen DSM und VE verkürzt sich

die zur Bearbeitung einer Task benötigte Zeit.

2.6.3 Kostenvergleich

Zum Vergleich der Kosten setzen wir voraus, daß zwei Anlagen der beiden betrachteten Strukturen bzgl. der Anwendungen die gleiche Leistung erbringen sollen. Bei der Definition der einzelnen Funktionseinheiten haben wir darauf geachtet, die Anzahl der Datenübertragungen so gering wie möglich zu halten, denn Datentransfereinrichtungen machen zur Zeit den größten Kostenanteil bei der Hardware aus. Auf Grund der Einsparung von Übertragungen kann die Zahl der Busse bzw. ihre Breite verringert werden. Auch durch die Senkung ihrer Leistungsfähigkeit, insbesondere der oberen Grenzfrequenz, lassen sich die Kosten in diesem Bereich senken. Die Anforderungen an die Arbeitsgeschwindigkeiten der Funktionsmodule verhalten sich indirekt proportional zur Anzahl parallel arbeitender Einheiten. Dabei muß auch berücksichtigt werden, daß ein Prozeßrechner mit nur einer CPU i.a. mehrere Programme quasigleichzeitig ausführt, wohingegen diese Tasks in der dezentralen Struktur von verschiedenen VE's bearbeitet werden. Wegen der geringeren Geschwindigkeitsanforderungen können zur Konstruktion der Funktionseinheiten relativ langsame Technologien eingesetzt werden, die jedoch einen hohen Integrationsgrad zulassen, was zu erheblichen Kostensenkungen führt. In jeder Rechananlage sind i.a. alle Modultypen mehrfach vorhanden. Von den entsprechenden Bauelementen können deshalb große Stückzahlen

hergestellt werden, wodurch sich diese weiter verbilligen. Wir werden unten einige Struktureigenschaften erwähnen, die die Anwenderprogramme einfacher und übersichtlicher machen als in Einprozessoranlagen. Das senkt die Fehlermöglichkeiten in der Programmierung und somit die Softwarekosten und verbessert gleichzeitig die Zuverlässigkeit der Programme.

2.6.4 Vergleich der Zuverlässigkeit

Wegen der Redundanz der Komponenten und der Dezentralisierung der Systemfunktionen hat die hier vorgeschlagene Rechnerstruktur auch hinsichtlich der Hardwarezuverlässigkeit Vorteile. So stehen immer mehrere parallele Übertragungswege zur Verfügung: die NÜ und mindestens eine VB. In einer konventionellen Anlage werden zur Durchführung der Funktionen der Verarbeitungs-, Speicher-, Peripherieadapter- und Nachrichtenübertragungsmodule die CPU, der Hauptspeicher und das Bussystem benötigt. Der Ausfall einer dieser Einheiten macht die gesamte Anlage funktionsunfähig. Dagegen kann die Arbeit defekter Module i.a. von anderen übernommen werden. Auch die Verteilung der Peripheriegerätesteuerung auf die verschiedenen Komponenten trägt zur Erhöhung der Zuverlässigkeit bei.

Sei B die Menge der zur Ausführung einer Gruppe von Tasks notwendigen Betriebsmittel und $p(b)$ die Wahrscheinlichkeit, daß $b \in B$ defekt ist. Mithin ist

$$P_1 = \sum_{b \in B} p(b)$$

die Wahrscheinlichkeit, daß die Tasks wegen Hardwareausfalls auf einer Einprozessoranlage nicht bearbeitet werden können. Entsprechend gilt für die andere Rechnerstruktur

$$P_2 = \sum_{\substack{B \cap BK_i \neq \emptyset \\ i=1, \dots, n}} \prod_{b \in BK_i} p(b) ,$$

wobei BK_i , $i=1, \dots, n$, die Klassen gleichartiger Module bezeichnen. Haben die Elemente dieser Klassen jeweils die gleichen Ausfallwahrscheinlichkeiten, so vereinfacht sich obiger Ausdruck zu

$$P_2 = \sum_{\substack{B \cap BK_i \neq \emptyset \\ i=1, \dots, n}} p(b: b \in BK_i)^{|BK_i|} .$$

Unter der Voraussetzung, daß die in beiden Fällen eingesetzte Technologie die gleiche Fehlerhäufigkeit besitzt, gilt

$$P_2 \leq P_1 ,$$

denn ein DSM wird z.B. oft nicht während der gesamten Dauer einer Programmausführung benötigt. Wegen $0 < p(b) < 1$ für alle $b \in B$ läßt sich der Wert von P_2 durch Vergrößerung der Redundanz beliebig senken. Damit wird auch erreicht, daß mehr Module zur Bearbeitung externer Ereignisse bereitstehen, was zu kürzeren und besser vorherbestimmbaren Reaktionszeiten führt.

2.6.5 Gegenüberstellung einiger weiterer Eigenschaften

In konventionellen Rechnern macht der Speicherzyklus zum Ladens eines Befehls einen bedeutenden Anteil der Zeit zu seiner Abarbeitung aus. Da das PSM in unserem Konzept der VE ihre Befehle vorausschauend zur Verfügung stellt, stehen sie bei Bedarf schon in der VE bereit. Aus den Fifos bzw. aus dem Schieberegister können sie dann wesentlich schneller als aus einem Programmspeicher gelesen werden.

Die CPU einer Einprozessoranlage muß alle Systemfunktionen wahrnehmen, was zu häufigem Context-Switching führt. Die dafür aufzuwendende Zeit und die notwendigen Datenübertragungen werden durch die Struktur unseres Rechners eingespart.

Außerdem entfallen eine Reihe von Synchronisations- und Sicherungsfunktionen, die durch die gemeinsame Nutzung der gleichen Betriebsmittel von verschiedenen Prozessen bedingt sind. So kann z.B. jede Task auf ihre privaten Dateien zugreifen, ohne einen Ein/Ausgabe-Dispatcher einschalten zu müssen, da diese Dateien von den DSM's und nicht von einer einzigen Treiberroutine und einer Steuereinheit verwaltet werden, und die Programme brauchen nicht geschützt zu werden, da sie in Speichermodulen stehen, in die nicht geschrieben werden kann.

Durch die Übertragung größerer Aufgaben an die verschiedenen Funktionsmodule haben diese eine höhere Komplexität als die Subsysteme herkömmlicher Rechner. Andererseits vereinfachen sich jedoch die Anwender- und Systemprogramme, denn die

PSM's sichern z.B. automatisch die Rückkehradressen nach Unterprogrammaufrufen und die DSM's entlasten die Programme von Adreßberechnungen und E/A-Operationen. Da Programmvariable direkt den Scratch-Pad-Registern zugewiesen werden können, lassen sich in Verbindung mit den Dreiadreibefehlen übersichtlichere Programme schreiben, als im Falle, daß nur wenige Akkumulatoren zur Verfügung stehen.

Damit das Betriebssystem einer Einprozessoranlage Aufträge der Anwenderprogramme bearbeiten kann, muß es nach seinem Aufruf (Supervisor-Call) und dem Umladen der Register vorher vorbereitete Auftragsparameter interpretieren. Die Funktionsmodule nehmen dagegen jeweils nur wenige Systemdienste wahr. Deshalb und weil die entsprechenden Routinen dauernd aktiv sind, erübrigen sich die genannten, bei Einprozessoranlagen nötigen, Operationen.

Kapitel II Ablaufsteuernde Funktionseinheiten

§ 3 Der Task-Scheduler

Die Ausführung von Task-Operationen ist i.a. an Einplanungsbedingungen (Schedules) S geknüpft, die ihrerseits von der Zeit und von binären Größen abhängen, die wir "Ereignisse" nennen wollen.

Ausgehend von einer mathematischen Theorie dieser Schedules und der Bedingungen für ihr Erfülltsein, wollen wir in diesem Paragraphen den Task-Scheduler beschreiben, in dem zur Entlastung der Verarbeitungseinheiten der in sich abgeschlossene Aufgabenkomplex der Beobachtung der Ereignisse und der Auswertung der Schedules, die als Boole'sche Prozeduren gegeben sind, zusammengefaßt ist. Die als Funktionsprozeduren realisierten Zeiteinplanungen liefern bei jedem Aufruf den jeweils nächsten kritischen Zeitpunkt, der dann in einen "Wecker" geladen wird. Die Arbeitsweise des Task-Schedulers wird durch eine Reihe von Routinen angegeben, die als Reaktion auf eingetretene Ereignisse ablaufen und deren Komplexität im wesentlichen linear von der Anzahl der zu überwachenden Schedules abhängt. Sind letztere erfüllt, dann werden die entsprechenden Tasks an den Dispatcher zur Betriebsmittelvergabe und zur Ausführung übergeben.

3.1 Formale Beschreibung der Aufgaben des Task-Schedulers

In [9] wurde gezeigt, daß vier Zustände ausreichen, um den augenblicklichen Stand der Bearbeitung einer Task zu charakterisieren. Jede Task befindet sich jeweils in genau einem der folgendermaßen definierten Zustände:

1. "bekannt" : die Task ist dem System bekannt, ihre Einplanungsbedingung ist z.Zt. nicht erfüllt;
2. "ablaufbereit" : die Task wartet auf die Zuteilung der benötigten Betriebsmittel;
3. "ablaufend" : die Task wird ausgeführt; und
4. "zurückgestellt" : die Bearbeitung der Task ist unterbrochen.

Nach diesem Modell verwaltet der Task-Scheduler die bekannten Tasks und führt gegebenenfalls den Übergang zum Zustand "ablaufbereit" durch. Die übrigen Zustände und Statustransfers werden vom Dispatcher organisiert und deshalb im nächsten Paragraphen beschrieben, wo auch ein Übergangsdiagramm angegeben werden wird. Weiterhin überwacht der Task-Scheduler auch die Einplanungsbedingungen aller anderen Task-Operationen.

Zur Beschreibung der Aufgaben des Task-Schedulers betrachten wir den Zeitpunkt t_0 , zu dem die $m(t_0) \in \mathbb{N}$ Task-Operationen T_i , $i=1, \dots, m$, bekannt seien, die wir zur Menge T zusammenfassen. Die Elemente E_j der Ereignismenge $E := \{E_j \mid j=1, \dots, n\}$, deren Mächtigkeit $n \in \mathbb{N}$ nicht von der

Zeit abhängen, seien Konjunktionen der eigentlichen Ereignisse \hat{E}_j und eventuell vorhandener Maskierungen $M_k^{(j)}$:

$$E_j = \hat{E}_j \bigwedge_{k=1}^{l_j} M_k^{(j)}, \quad l_j \in \mathbb{N}, \quad j=1, \dots, n.$$

Vermöge $F: T \rightarrow S$ werde die Menge der Task-Operationen T surjektiv auf die Menge $S := \{S_k(t, E) \mid k=1, \dots, l(t_0)\}$, $l(t_0) \in \mathbb{N}$, der Einplanungsbedingungen abgebildet. Zwischen den Ereignissen und den Laufbedingungen für die Tasks besteht die Relation

$$R_1 := \{(E_j, S_k) \mid j=1, \dots, n, k=1, \dots, l, S_k \text{ hängt explizit von } E_j \text{ ab}\} \subset E \times S.$$

Jeder der explizit zeitabhängigen Schedules aus S impliziert eine Folge von Zeitpunkten, zu denen die entsprechenden Task-Operationen gegebenenfalls auszuführen sind. Wir vereinigen alle diese Zeitpunkte zu der abzählbaren, streng monoton aufsteigend geordneten Menge $Z := \{t_1 < t_2 < \dots\} \subset [t_0, \infty)$, zwischen der und der Menge der Einplanungsbedingungen eine weitere Relation

$$R_2 := \{(t_i, S_k) \mid i=1, \dots, |Z|, k=1, \dots, l, S_k \text{ hängt explizit von } t \text{ ab und impliziert } t_i\} \subset Z \times S$$

existiert. Zur Reduzierung des Verarbeitungsaufwandes des Task-Schedulers setzen wir o.B.d.A. voraus, daß falls

$(E_j, S_k) \in R_1$ ist, $j \in \{1, \dots, n\}$ und $k \in \{1, \dots, l\}$, nur E_j , jedoch nicht $\overline{E_j}$, in den Ausdruck für S_k eingeht. Das stellt keine Einschränkung dar, weil E_j durch das Ereignis $\overline{E_j}$ ersetzt werden kann bzw. sowohl E_j als auch $\overline{E_j}$ Elemente von E sein können. Daraus folgt, daß die logische Funktion S_k , $k \in \{1, \dots, l\}$, zum Zeitpunkt $t \in [t_0, \infty)$ ihren Wert nur dann

von 0 nach L ändern kann, wenn

1. $(t, S_k) \in R_2$ ist, oder
2. mindestens ein E_j , $j \in \{1, \dots, n\}$, mit $(E_j, S_k) \in R_1$ seinen Zustand von 0 nach L verändert hat.

Mithin hat der Task-Scheduler die Aufgabe, alle Einplanungsbedingungen $S_k \in S$, $1 \leq k \leq l$, auszuwerten, für die gilt:

1. $(t, S_k) \in R_2$, falls ein $t_i \in Z$ mit $t = t_i$, $i \in \{1, \dots, |Z|\}$, existiert, oder
2. $(E_j, S_k) \in R_1$, falls das Ereignis E_j , $1 \leq j \leq n$, zum Zeitpunkt t den Zustand L angenommen hat.

Schließlich müssen jeweils die in $F^{-1}(S^{(t)})$ enthaltenen

Task-Operationen ausgeführt werden, wobei

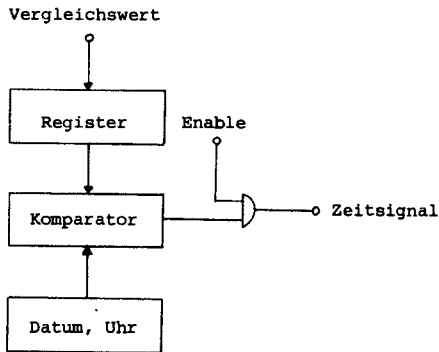
$$S^{(t)} := \{S_k \in S \mid S_k(t-0) = 0 \wedge S_k(t+0) = L, k=1, \dots, l\}$$

und $t \in [t_0, \infty)$ seien. Um die genannten Verarbeitungen vornehmen zu können, benötigt der Task-Scheduler die Einplanungsbedingungen in der Form Boole'scher Prozeduren. Hängt der Schedule S_k explizit von der Zeit ab, dann wird außerdem eine Funktionsprozedur s_k mit der Eigenschaft

$$s_k(t) = \begin{cases} \min\{Z_k \cap (t, \infty)\} & , \text{ falls } Z_k \cap (t, \infty) \neq \emptyset \\ \text{"Erschöpft-Kennung"} & , \text{ sonst} \end{cases}$$

gebraucht, wobei Z_k die von S_k implizierte Teilmenge von Z bezeichne. Auf Grund dieser Konstruktion ist es nicht erforderlich, alle Elemente von Z und die ganze Relation R_2 jederzeit zur Verfügung zu haben. Es genügt, zum Zeitpunkt t_0 alle Prozeduren s_k auszuwerten, die Ergebnisse zur Menge \hat{Z} zusammenzufassen und aufsteigend zu ordnen sowie die Teilmenge

$$\hat{R}_2 := \{(t_i, S_k) \in R_2 \mid t_i \in \hat{Z}, i=1, \dots, |Z|, k=1, \dots, l\}$$



Figur 7. "Wecker"

von R_2 zu speichern. Das kleinste Element von \hat{Z} wird in das Vergleichsregister des in Figur 7 skizzierten "Weckers" geladen, der bei Erreichen dieses Zeitpunktes ein Signal erzeugt. Gegenüber der Verwendung eines Differenzzeitzählers brauchen bei dieser Methode keine Programmlaufzeiten des bearbeitenden Prozessors berücksichtigt zu werden. Neben den bereits oben erwähnten Aufgaben hat der Task-Scheduler beim Eintreffen des Zeitsignals zum Zeitpunkt $t_i \in Z$, $i \in \{1, \dots, |Z|\}$, folgendes zu tun:

1. Entfernen des kleinsten Elementes aus der Menge \hat{Z} :
 $\hat{Z} := \hat{Z} \setminus \{t_i\}$;
2. Auswerten aller Prozeduren s_k mit $(t_i, s_k) \in R_2$, $1 \leq k \leq l$;
3. Vereinigen der in (2) erhaltenen Zeitwerte mit \hat{Z} und Sortieren der Ergebnismenge \hat{Z} ;
4. Prüfen, ob die erschöpften Zeiteinplanungen s_k zugeordneten Schedules S_k nicht mehr den Wert L annehmen können, und gegebenenfalls Entfernen aller entsprechenden Ele-

- mente aus R_1 ;
5. Bilden der aktuellen Teilmenge \hat{R}_2 von R_2 ;
 6. Laden des Vergleichsregisters des Weckers mit dem Minimum von \hat{Z} .

3.2 Task-Scheduling-Algorithmen

Nachdem wir im letzten Abschnitt die Aufgaben des Task-Schedulers herausgearbeitet und seine wesentlichen Datenstrukturen beschrieben haben, wollen wir nun die Arbeitsweise des Moduls durch die Angabe einer Reihe von Algorithmen darlegen. Alle diese Routinen sind selbst Tasks, die in der Einheit ablaufen und durch externe Signale aktiviert werden.

Neben den bereits oben definierten Mengen tritt im folgenden die Menge $s := \{s_k(t) \mid k=1, \dots, l\}$ der Zeitprozeduren als Operand auf. Die jedem Schedule $S_k \in S$ zugeordnete Prozedur s_k liefert die Abbildung $zp: S \rightarrow s$. Mit vr werde das Vergleichsregister des Weckers bezeichnet, \uparrow stehe für den O-L-Übergang eines Signals. Durch den Aufruf der Prozedur $sort$ werde eine Menge in aufsteigender Reihenfolge sortiert. Das Minimum einer Menge stelle die Funktion min zur Verfügung. Das Signal eof zeige an, daß keine vom Task-Scheduler einzulesenden Informationen anstehen. Die Parameter auszuführender Task-Operationen werden an den Dispatcher übertragen, der die Adresse dp habe. Die übrigen in den folgenden Programmen vorkommenden Größen sind Hilfsvariable, insbesondere bezeichne t die aktuelle Zeit.

Ein reset-Signal versetzt den Task-Scheduler in die Grundstellung und initialisiert seine Datenstrukturen:

on reset do

$T:=F:=S:=s:=zp:=\hat{Z}:=R_1:=\hat{R}_2:=\emptyset$, enable:=false , vr:= ∞

od .

Seine Tätigkeit beginnt der Task-Scheduler mit dem Laden des Weckers nach einem start-Signal:

on start do

if $\hat{Z} \neq \emptyset$ then vr:=min(\hat{Z}) fi ; enable:=true

od .

Jederzeit nach dem reset-Signal können die Parameter von Task-Operationen und Einplanungsbedingungen an den Task-Scheduler übergeben werden, der zur Übernahme der entsprechenden Daten durch ein load-Signal aufgefordert wird. Die folgende Routine führt das Einordnen der eingelesenen Parameter in die Datenstrukturen des Schedulers durch. Werden neue Einplanungsbedingungen für schon bekannte Task-Operationen angegeben, dann werden die alten Schedules überschrieben. Die zum Löschen dieser Bedingungen dienende Prozedur delete wird weiter unten definiert.

on load do

while $\neg \text{eof}$ do

get(TN,SN,sn,EN) ;

if $TN \in T$ then $\Sigma := F(TN)$; $F := F \setminus \{(TN, \Sigma)\}$;

if $F^{-1}(\Sigma) = \emptyset$ then delete(Σ) fi fi ;

$T := T \cup \{TN\}$, $S := S \cup \{SN\}$, $s := s \cup \{sn\}$, $F := F \cup \{(TN, SN)\}$,

$zp := zp \cup \{(SN, sn)\}$, $R_1 := R_1 \cup \{(e, SN) \mid e \in EN\}$;

```

if  $sn \neq nil$  then
   $tn := sn(t)$  ;
  if  $tn = "erschöpft"$  then
    if  $tz > t \rightarrow SN(tz, E) = false$  then  $delete(SN)$  fi
  else  $\hat{Z} := \hat{Z} \cup \{tn\}$  ,  $\hat{R}_2 := \hat{R}_2 \cup \{(tn, SN)\}$  ;  $sort(\hat{Z})$  ;
    if  $tn = \min(\hat{Z})$  then  $vr := tn$  fi
  fi
fi
od
od .

```

Nach einem prevent-Signal entfernt der Task-Scheduler eingeplante Task-Operationen aus seinen Listen, er führt also die zu load inverse Operation aus. Ist mit dem Schedule einer auszuplanenden Task-Operation keine weitere mehr assoziiert, dann wird auch der Schedule durch Aufruf von delete gelöscht.

```

on prevent do
  while  $\neg eof$  do
     $get(TN)$  ;
     $T := T \setminus \{TN\}$  ,  $SN := F(TN)$  ;  $F := F \setminus \{(TN, SN)\}$  ;
    if  $F^{-1}(SN) = \emptyset$  then  $delete(SN)$  fi ;
  od
od .

```

Die folgende Prozedur delete entfernt eine Einplanungsbedingung und alle mit ihr verknüpften Informationen aus den Dateien des Task-Schedulers und setzt gegebenenfalls einen neuen Wert in das Vergleichsregister des Weckers.

```

proc(proc bool) delete=(proc bool  $\Sigma$ ):
 $\bar{\sigma} := zp(\Sigma)$  ;  $S := S \setminus \{\Sigma\}$  ,  $s := s \setminus \{\sigma\}$  ,  $zp := zp \setminus \{(\Sigma, \sigma)\}$  ,
for all  $e \in E$  with  $(e, \Sigma) \in R_1$  do  $R_1 := R_1 \setminus \{(e, \Sigma)\}$  od ;
 $M := F^{-1}(\Sigma)$  ;  $T := T \setminus M$  ,
for all  $m \in M$  do  $F := F \setminus \{(m, \Sigma)\}$  od ;
if  $\sigma \neq nil$  then
for all  $tz \in \hat{Z}$  with  $(tz, \Sigma) \in \hat{R}_2$  do
 $\hat{R}_2 := \hat{R}_2 \setminus \{(tz, \Sigma)\}$  ;
if  $\hat{R}_2 \cap \{(tz, SN) \mid SN \in S\} = \emptyset$  then  $\hat{Z} := \hat{Z} \setminus \{tz\}$  fi
od ;
vr := if  $\hat{Z} \neq \emptyset$  then  $\min(\hat{Z})$  else  $\infty$  fi
fi .

```

Wir kommen nun zu den eigentlichen Aufgaben des Task-Schedulers, der Auswertung der Einplanungsbedingungen beim Eintreten von Ereignissen. Bei jedem O-L-Übergang eines $e \in E$ wird das folgende Programm ausgeführt:

```

on  $e = \dagger$  do
for all  $SN \in S$  with  $(e, SN) \in R_1$  do check(SN) od
od ,

```

wobei alle mit dem Ereignis e in Relation stehenden Schedules überprüft werden. Nimmt eine Einplanungsbedingung den logischen Wert L an, dann werden die mit ihr assoziierten Task-Operationen an den Dispatcher zur Ausführung übertragen:

```

proc(proc(real, [1:n] bool) bool) check=
(proc(real t, [1:n] bool E) bool  $\Sigma$ ):
if  $\Sigma(t, E)$  then

```

for all $TN \in T$ with $(TN, L) \in F$ do put(dp, TN) od
fi .

Auf das Zeitsignal des Weckers hin werden alle mit dem jeweiligen Minimum von \hat{Z} bzgl. der Relation \hat{R}_2 in Beziehung stehenden Schedules wie oben überprüft. Aus \hat{Z} werden das kleinste Element und aus \hat{R}_2 die entsprechenden Tupel entfernt. Die den überprüften Task-Ablaufbedingungen zugeordneten Zeiteinplanungen werden ausgewertet und mit den sich ergebenden Zeitpunkten werden die Mengen \hat{Z} und \hat{R}_2 aktualisiert. Schedules, die mit erschöpften Zeiteinplanungen assoziiert sind, werden aus den Listen des Task-Schedulers gestrichen, sofern sie nicht mehr den Wert L annehmen können. Schließlich wird das Vergleichsregister mit dem neuen Minimum von \hat{Z} geladen.

on zeitsignal do
 $tmin := \min(\hat{Z})$; $\hat{Z} := \hat{Z} \setminus \{tmin\}$;
for all $SN \in S$ with $(tmin, SN) \in \hat{R}_2$ do
 $check(SN)$, $tn := zp(SN)(t)$, $\hat{R}_2 := \hat{R}_2 \setminus \{(tmin, SN)\}$;
if $tn = \text{"erschöpft"}$ then
if $tz > t * SN(tz, E) = \text{false}$ then delete(SN) fi
else $\hat{Z} := \hat{Z} \cup \{tn\}$, $\hat{R}_2 := \hat{R}_2 \cup \{(tn, SN)\}$
fi
od ;
 $sort(\hat{Z})$; $vr := \text{if } \hat{Z} \neq \emptyset \text{ then } \min(\hat{Z}) \text{ else } =$ fi
od .

Unter der Voraussetzung, daß die Anzahl einem Schedule zugeordneter Task-Operationen klein ist - im Regelfall wird sie eins sein - , ist die Komplexität der Routine, die das

Eintreten eines Ereignisses aus E bearbeitet, der Mächtigkeit von S proportional. Zum Sortieren von \hat{Z} seien $N(|\hat{Z}|)$ Operationen notwendig. Dann erfordert das zeitsignal-Programm die Bearbeitung von $O(|s| + N(|\hat{Z}|))$ Anweisungen. Die Komplexität der prevent-Routine hängt linear von der Anzahl der Eingabedaten ab, die der load-Routine ist dem Produkt aus $N(|\hat{Z}|)$ und der Zahl der übergebenen Datensätze proportional.

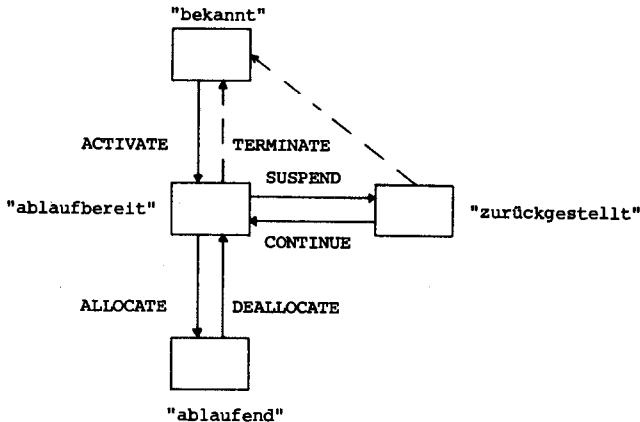
Die Signale lassen sich als positiv flankengetriggerte Flipflops realisieren. Wie bereits erwähnt, bestehen die Mengen S und s aus Prozeduren, die dem Task-Scheduler zur Verfügung gestellt werden müssen. Die Elemente von T enthalten neben einer Aufgabenkennung Parameter für den Dispatcher und die spätere Programmausführung. Durch eindimensionale Felder lassen sich \hat{Z} und die Abbildungen F und z_p darstellen. Die drei Relationen R_1 , \hat{R}_2 und F^{-1} können als invertierte Dateien oder als zweidimensionale Bitfelder implementiert werden. Im zweiten Fall ist eine Reihe der oben vorkommenden Anweisungen durch Bitkettenoperationen realisierbar.

§ 4 Der Dispatcher

Tasks, deren Ablaufbedingungen erfüllt sind, werden vom Task-Scheduler dem Dispatcher gemeldet, der nun ihre weitere Bearbeitung überwacht und insbesondere die Betriebsmittel zuteilt. Da einige Betriebsmittel exklusiven Zugriff verlangen und nicht unterbrechbar sind, besteht die Möglichkeit von Systemverklemmungen, die wir durch Anwendung des betriebsmittelhierarchischen Dispatchings von vornherein ausschließen wollen, um kurze Reaktionszeiten im Realzeitbetrieb zu gewährleisten. Die Verarbeitungseinheiten einer Leistungsklasse fassen wir zu einem symmetrischen Teilmehrprozessorsystem zusammen, deren Belegungen nach der zeitgerechten Vorhaltestrategie bestimmt werden. Diese gestattet die Prüfung, ob eine Taskmenge innerhalb der vorgegebenen Zeitgrenzen mit den Prozessoren einer bestimmten Leistungsklasse bearbeitbar ist. Gegebenenfalls muß die Ausführung von Tasks auf schnellere Verarbeitungseinheiten verlagert werden. Wir werden im folgenden die Bedingungen für den Aufruf des Vorhaltealgorithmus und die Überführung von Tasks in die Ablaufphase, die von dem jeweiligen Belegungszustand der Betriebsmittel abhängt, formulieren. Ablaufbereite Tasks können solchen mit höherer Dringlichkeit in der Ausführung vorgezogen werden, sofern andere nicht behindert werden und die benötigten Betriebsmittel frei sind und später unter Umständen auch wieder entzogen werden können. Dieser Abschnitt schließt mit der Beschreibung der vom Dispatcher geführten Datensätze und der Angabe der von ihm abzuarbei-

tenden Prozeduren, die alle polynomiale Komplexität besitzen.

4.1 Task-Zustände und Zustandsübergänge



Die obenstehende Skizze zeigt die im letzten Paragraphen eingeführten Task-Zustände zusammen mit den sinnvollen Zustandsübergängen. Letztere werden durch Task-Operationen bewirkt, mit deren Ausführung der Dispatcher vom Task-Scheduler beauftragt wird, wenn die entsprechenden Einplanungsbedingungen erfüllt sind.

Die Operation **ACTIVATE** überführt ruhende Tasks in den Zustand **"ablaufbereit"**. Nun vergibt der Dispatcher an diese Tasks die benötigten Betriebsmittel und versetzt sie mittels **ALLOCATE** in die Ausführungsphase. Die zu **ALLOCATE** inverse Operation ist **DEALLOCATE**, die Tasks, deren Ablauf beendet, ab- oder unterbrochen werden soll, aus dem Zustand **"ablaufend"** in den Zustand **"ablaufbereit"** transferiert. Der Task-

Zustand "zurückgestellt" ist dadurch gekennzeichnet, daß die sich in ihm befindlichen Tasks auf Grund von Wartebedingungen in ihrem Ablauf vorübergehend unterbrochen sind, die nicht durch die Dispatching-Strategie verursacht sind. Gründe dafür können das Abwarten von Zeitbedingungen oder der Aufhebung von Zugriffsbeschränkungen für gemeinsame Datenbereiche mehrerer Tasks sein. Die Prozedur SUSPEND vollzieht den Zustandsübergang von "ablaufbereit" nach "zurückgestellt", CONTINUE den in der umgekehrten Richtung. Schließlich versetzt die Ausführung der Operation TERMINATE ablaufbereite bzw. zurückgestellte Tasks in den Zustand "bekannt" zurück, der vom Task-Scheduler verwaltet wird.

4.2 Behandlung von Deadlocks

Bei der Vergabe von Betriebsmitteln ist die Möglichkeit des Auftretens von Systemverklemmungen zu berücksichtigen. Es gibt drei Vorgehensweisen zur Behandlung solcher Deadlocks: sie können a priori verhindert, vermieden oder erkannt und aufgelöst werden. Für harte Realzeitanwendungen ist die Verhinderung die einzige praktikable Alternative, da ein zyklisch durchzuführender Deadlocktest sehr aufwendig und die Auflösung von Verklemmungen durch Betriebsmittelentzug und Prozeßneubeginn i.a. unmöglich ist. Auch ein zur Vermeidung von Deadlocks laufend anzuwendender Sicherheitstest bindet zu viel Verarbeitungskapazität und erübrigt sich in den meisten Fällen, denn das gesamte Prozeßsystem einer Prozeßrechnerinstallation ist a priori bekannt und über-

prüfbar. Das Auftreten von Deadlocks setzt wesentlich die Notwendigkeit des exklusiven Zugriffs auf bestimmte Betriebsmittel voraus. Dieses Problem tritt in der Prozeßdatenverarbeitung jedoch nicht so stark in Erscheinung wie in anderen Anwendungen, weil die meisten Peripheriegeräte wegen ihrer festen Verbindung mit dem zu steuernden Prozeß ohnehin mit jeweils nur einer Task korrespondieren. Außerdem wird durch die Einführung von Speichermodulen ein großer Teil der Zugriffskonflikte zu Massenspeichern vermieden. Es bleibt der Zugriff zu gemeinsamen Daten, wobei logisch nicht zwischen haupt- und massenspeicherresidenten Daten bzw. Ein/Ausgabewerten unterschieden werden muß, wenn diese mit einem anderen Modul ausgetauscht werden.

Zur Verhinderung von Deadlocks sind die betriebsmittelfreigebende und die betriebsmittelhierarchische Methode bekannt. Die im folgenden näher behandelte Dispatching-Strategie erfordert, daß Tasks Betriebsmittel vorübergehend entzogen werden können. Bevor diese später wieder zurückgefordert werden können, müßten die Tasks gemäß der betriebsmittelfreigebenden Methode zuerst alle noch in ihrem Besitz befindlichen Betriebsmittel freigeben, um dann den gesamten Betriebsmittelbedarf zusammen anzufordern. Da Tasks z.B. während kritischer Phasen gemeinsame Daten vor dem Zugriff anderer Tasks sichern müssen, ist diese Vorgehensweise i.a. nicht praktikabel. Darum werden wir die Betriebsmittelvergabe entsprechend einer Hierarchie von Betriebsmittelklassen [13] vornehmen. Es wird i.a. unmöglich sein, eine Be-

triebsmittelhierarchie zu definieren, die allen Tasks eines Prozeßsystems erlaubt, Betriebsmittel nachzufordern, ohne vorher Teile ihres Betriebsmittelbesitzes freizugeben. Wir teilen nun die dem Dispatching unterliegenden Funktionsmodule gemäß Tabelle 2 in drei Klassen ein, wobei wir uns an der Anzahl untereinander austauschbarer Module und ihrer Unterbrechbarkeit orientieren.

Tabelle 2. Definition von Betriebsmittelklassen

Klasse	Modultyp	Anzahl austauschbarer Module	Unterbrechbar
H ₁	VE	>1	ja
H ₂	PSM	1	ja
	DSM	1	ja
	PA	1	ja
	Spez. Proz.	1	ja
H ₃	DSM	1	nein
	PA	1	nein
	Spez. Proz.	1	nein

Die Eigenschaft "Unterbrechbar" gibt an, ob ein Modul einer Task zugewiesen werden darf, wenn die Task, in deren Besitz es sich vorher befunden hat, noch nicht beendet ist. Die Unterbrechbarkeit hängt von der Belegung der Datenspeichermodule, den an den Peripherieadaptern angeschlossenen Geräten sowie von den Eigenschaften der spezialisierten Prozessoren ab. Dementsprechend werden die genannten Module in die Klassen H₂ bzw. H₃ eingeordnet. Sind in einem System

mehrere gleichartige Verarbeitungseinheiten einer bestimmten Leistungsklasse vorhanden, dann kann eine Task nacheinander von verschiedenen dieser Prozessoren bearbeitet werden. Auf Grund in den Modulen gespeicherter Zustandsinformationen und der festen Verbindung mit Peripheriegeräten ist dies bei den anderen Modultypen nicht möglich, so daß nach Unterbrechungen die schon früher benutzten Einheiten zurückgefordert werden müssen. Zur Festlegung der Betriebsmittelhierarchie definieren wir $H_1 > H_2 > H_3$. Da die Verarbeitungseinheiten die höchste Klasse bilden, muß der gesamte Betriebsmittelbedarf einer Task vor Beginn der Programmausführung angefordert und zugeteilt werden. Dieses erscheint gerechtfertigt zu sein, da Tasks in der Prozeßdatenverarbeitung meistens nur recht wenige Aktionen ausführen und deshalb nur selten längere Laufzeiten haben. Nachforderungen von Betriebsmitteln aus H_1 und H_2 finden nur dann statt, wenn diese vorher entzogen worden sind. Durch die Vermeidung von Dispatcher-Aufrufen zur Nachforderung von Betriebsmitteln, die nicht beim Task-Start zugeteilt worden sind, verkürzen sich die Bearbeitungszeiten der Tasks und die Belegungszeiten der Betriebsmittel aus H_3 . Werden Betriebsmittel nur unter gewissen Bedingungen gebraucht, so kann eine Subtask gestartet werden, die diese Betriebsmittel anfordert und die entsprechenden Operationen durchführt. Zur Effektivitätssteigerung und aus Sicherheitsüberlegungen heraus wird man bei der Entwicklung von Prozeßrechnerprogrammen ohnehin darauf achten, daß zumindest die zyklisch aktivierten Tasks konfliktfrei ablaufen können. Die Frei-

gabe belegter Betriebsmittel sollte selbstverständlich so früh wie möglich erfolgen.

4.3 Auswahl von Dispatching-Strategien

Realzeitbetrieb setzt voraus [7], daß die Verarbeitungsergebnisse innerhalb vorherbestimmter Zeiten verfügbar sind, d.h. daß die Antwortzeiten der Tasks eingehalten werden. Der Nachweis, daß ein gegebenes Task-System auf einem Prozeßrechner zeitgerecht verarbeitet werden kann, ist während der Einsatzplanung und Kapazitätsberechnung vorzunehmen. Die Aufgabe des Dispatchers besteht dann darin, die Betriebsmittel den ablaufbereiten Tasks so zuzuteilen, daß diese innerhalb ihrer Antwortzeiten ausgeführt werden können. Das erfordert im wesentlichen eine Strategie zur zeitgerechten Prozessorzuteilung. Nach Kapitel I kann ein Prozeßrechner der hier behandelten Struktur mehrere Verarbeitungseinheiten verschiedener Leistungsfähigkeit besitzen. Die VE's einer Leistungsklasse stellen dann ein symmetrisches (Teil-) Mehrprozessorsystem dar. Die ablaufbereiten Tasks, die sich um die Zuteilung von VE's einer bestimmten Leistungsstufe bewerben, bilden eine "freie Task-Menge" [15], weil sie nach der Betriebsmittelzuteilung sofort gestartet werden können. Zur zeitgerechten Prozessorzuteilung an freie Task-Mengen sind zwei Verfahren bekannt: die Strategie des minimalen Spielraums [14] und die Vorhaltestrategie [14,19]. Da erstere zu häufigem Context-Switching führt, was nach

§ 2 viele Datenübertragungen erfordert, werden wir hier die Vorhaltestrategie anwenden. Diese reduziert sich im Falle eines einzelnen Prozessors auf die optimale Strategie [20] nach aufsteigenden Antwortzeiten und gestattet die Überprüfung, ob eine zu einem bestimmten Zeitpunkt gegebene Task-Menge zeitgerecht verarbeitbar ist. Gegebenenfalls muß "Last abgeworfen" werden, was durch Übertragung von Tasks an VE's mit größerer Leistungsfähigkeit erfolgen kann. In der Testphase kann dieser Mechanismus dazu dienen festzustellen, ob ein Task-System nicht zeitgerecht bearbeitet werden kann. Da die Prozessoren die Betriebsmittelhierarchie anführen, brauchen die zugewiesenen Betriebsmittel der beiden übrigen Klassen bei den durch die Vorhaltestrategie bedingten Umverteilungen der Verarbeitungseinheiten unter den freien Tasks nicht berücksichtigt zu werden. Durch die Anwendung der Vorhaltestrategie auf jede einzelne Prozessorleistungsklasse ergibt sich insgesamt eine praktikable Dispatching-Strategie, wohingegen für asymmetrische Multiprozessorsysteme keine nichtaufzählenden Algorithmen bekannt sind [6].

4.4 Bedingungen für die Betriebsmittelvergabe

Die im übernächsten Abschnitt dargestellten Dispatcher-Algorithmen beruhen auf Betriebsmittelvergabebedingungen, die von den jeweiligen Belegungszuständen der einzelnen Betriebsmittel und den angewandten Dispatching-Strategien abhängen. Wir werden diese Bedingungen im folgenden exakt formulieren.

Gegeben sei eine Menge $BM := \{E_i \mid i=1, \dots, n\}$ von n Betriebsmitteln, die sich gemäß $H_1 := \{E_i \mid i=1, \dots, n_1\}$,

$H_2 := \{E_i \mid i=n_1+1, \dots, n_1+n_2\}$ und $H_3 := \{E_i \mid i=n_1+n_2+1, \dots, n=n_1+n_2+n_3\}$ auf die drei Betriebsmittelklassen verteilen.

Gemäß Tabelle 2 existiert auf H_1 eine Äquivalenzrelation $P := \{(E_i, E_j) \mid E_i \text{ und } E_j \text{ sind in der Benutzung austauschbar, } 1 \leq i, j \leq n_1\} \subset H_1^2$,

die auf H_1 eine Klasseneinteilung $K := \{PK_i \mid i=1, \dots, m\}$ der Mächtigkeit $m \leq n_1$ induziert. Die Elemente von K sind also die Mengen der Verarbeitungseinheiten jeweils gleicher Leistungsfähigkeit. Letztere wird durch $m-1$ Faktoren

$\{\lambda_i \mid 0 < \lambda_i < 1; i=2, \dots, m\}$ charakterisiert: sei L die Laufzeit einer Task auf einer $VE \in PK_{i-1}$, so reduziert sich diese auf $\lambda_i \cdot L$, wenn die Task von einem Element von PK_i ausgeführt wird, $i=2, \dots, m$. Zwischen der Menge $TM := \{T_i \mid i=1, \dots, \tau\}$

aller τ Tasks eines Prozeßsystems und der Menge von Betriebsmittelklassen $BK := K \cup \{E_i \mid i=n_1+1, \dots, n\}$ der Mächtigkeit $m+n_2+n_3$ besteht die Relation

$B := \{(T_i, EK_j) \mid \text{zur Ausführung von } T_i \text{ wird genau ein Element aus } EK_j \text{ benötigt; } i \in \{1, \dots, \tau\}, j \in \{1, \dots, m+n_2+n_3\}\} \subset TM \times BK$.

Eine Task T_i kann mithin nur dann ablaufen, wenn aus allen Klassen EK_j mit $(T_i, EK_j) \in B$ jeweils genau ein Betriebsmittel zur Verfügung gestellt wird. Zur Überprüfung des Betriebsmittelbelegungszustandes führen wir für jedes Element von BM durch

$$ZM := \left\{ Z_i \mid Z_i = \begin{cases} L, & E_i \text{ ist frei} \\ 0, & E_i \text{ ist belegt} \end{cases}; i=1, \dots, n \right\}$$

ein Zustandssignal ein. Unter Umständen ist es erforderlich, daß Tasks von ihnen belegte Betriebsmittel vorübergehend vor Entzug sichern. Um festzustellen, ob Betriebsmittelentzug möglich ist, definieren wir die weiteren Zustandssignale

$$ZM' := \left\{ Z_i' \mid Z_i' = \begin{cases} L, E_i & \text{kann entzogen werden} \\ 0, E_i & \text{darf nicht entzogen werden} \end{cases}; i=1, \dots, n \right\}.$$

Natürlich impliziert $Z_i = L$ den Zustand $Z_i' = L$, $i=1, \dots, n$, und aus $Z_i = 0$ folgt $Z_i' = 0$ für $i=n_1+n_2+1, \dots, n$. Eine ablaufbereite Task $T_v \in TM$, $v \in \{1, \dots, \tau\}$, kann in den Zustand "ablaufend" überführt werden, wenn alle benötigten Betriebsmittel frei sind, d.h. es gilt

$$\overline{(T_v, EK_j) \in B} \quad j=1, \dots, m+n_2+n_3 \quad i = \sum_{k=1}^{j-1} |EK_k| + 1, \dots, \sum_{k=1}^j |EK_k| \quad Z_i$$

oder wenn T_v auf Grund der Dispatching-Strategie anderen Tasks vorgezogen werden soll und wenn die benötigten Betriebsmittel entzogen werden können. Letzteres ist unter der Voraussetzung

$$\overline{(T_v, EK_j) \in B} \quad j=1, \dots, m+n_2+n_3 \quad i = \sum_{k=1}^{j-1} |EK_k| + 1, \dots, \sum_{k=1}^j |EK_k| \quad Z_i'$$

dann der Fall, wenn

$$\overline{(T_v, EK_j) \in B \cap TM \times K \cup H_2} \quad j=1, \dots, m+n_2 \quad i = \sum_{k=1}^{j-1} |EK_k| + 1, \dots, \sum_{k=1}^j |EK_k| \quad Z_i \vee$$

$$\vee [\overline{Z_i \wedge Z_i' \wedge T_v <_j (Task \in TM, \text{ die z.Zt. } E_i \text{ belegt})}] \wedge$$

$$\wedge \overline{(T_v, EK_j) \in B \cap TM \times H_3} \quad j=m+n_2+1, \dots, m+n_2+n_3 \quad Z_{j+n_1-m}$$

gilt, wobei " $<_j$ " die von der auf die Betriebsmittelklasse EK_j angewandten Dispatching-Strategie induzierte Reihen-

folgerelation der Tasks aus TM sei. Im Falle $|EK_j|=1$ wird die Strategie nach aufsteigenden Antwortzeiten, wobei für $j > m+n_2$ die Nichtunterbrechbarkeit berücksichtigt werden muß, und sonst die Vorhaltestrategie verwandt. Die zweite der oben angegebenen Bedingungen impliziert offensichtlich die erste. Sind benötigte Betriebsmittel belegt und können sie nicht entzogen werden, so sind die entsprechenden Tasks nicht sofort ausführbar. Deshalb besteht die Möglichkeit, daß Kapazitäten frei bleiben, auf denen Tasks bearbeitet werden könnten, die eigentlich noch garnicht an der Reihe sind. Um die blockierten Tasks auf keinen Fall zu behindern, ziehen wir Tasks nur unter der Voraussetzung in der Ausführung vor, daß nur solche Betriebsmittel aus H_3 zugewiesen werden, die von keiner blockierten Task benötigt werden.

Gilt für eine Task T_v

$$r := \min\{j \in \{1, \dots, m+n_2+n_3\} \mid (T_v, EK_j) \in B\} \in \{1, \dots, m\},$$

dann soll T_v auf einem Prozessor der Klasse PK_r bearbeitet werden. Kann nicht garantiert werden, daß dies zeitgerecht erfolgt, dann wird T_v für $r < m$ auf die Prozessoren der Klasse PK_{r+1} übertragen. Da diese VE's schneller sind, vergrößert sich der Spielraum von T_v .

4.5 Bedingungen für die Anwendung des Vorhaltealgorithmus

Wie bereits oben dargelegt, wollen wir die Belegung von Klassen mehrerer gleichartiger Verarbeitungseinheiten nach der Vorhaltestrategie verwalten. In diesem Abschnitt werden

die Ereignisse aufgeführt, die den Aufruf des Vorhaltealgorithmus erfordern. Letzterer impliziert eine Menge kritischer Zeitpunkte, die wir näher charakterisieren wollen.

Für jede Task $T_v \in TM$ gebe $t_z^{(v)}$ den Zeitpunkt an, zu dem T_v beendet sein muß. Die Funktionen $a_v(t) := t_z^{(v)} - t$ und $l_v(t)$ stellen für $0 \leq t < \infty$ die Antwortzeit von T_v bzw. den noch zur Ausführung von T_v benötigten Zeitraum dar. Die Differenzfunktion $s_v(t) := a_v(t) - l_v(t)$, $0 \leq t < \infty$, heie Spielraum der Task T_v . Die Task T_μ habe eine Vorhaltung vor T_v , wenn $s_\mu < a_v$ gilt.

Für jede Prozessorklasse PK_j , $j \in \{1, \dots, m\}$, muß der Vorhaltealgorithmus - er ist Teil der im folgenden Abschnitt angegebenen Prozedur *vstrat* - nicht nur dann durchlaufen werden, wenn

- (i) eine Task T_v mit $(T_v, PK_j) \in B$ in den Zustand "ablaufbereit" übergeht, $\forall v \in \{1, \dots, \tau\}$; oder
- (ii) eine ablaufende Task T_v mit $(T_v, PK_j) \in B$ den ihr zugewiesenen Prozessor freigibt, $\forall v \in \{1, \dots, \tau\}$;

sondern auch [15,19], wenn

- (iii) im Falle $|PK_j| > 1$ die Vorhaltestrategie dies erfordert:
 - a) der Spielraum einer Task T_v mit $(T_v, PK_j) \in B$, der noch kein Prozessor zugewiesen wurde, verschwindet, $\forall v \in \{1, \dots, \tau\}$;
 - b) eine Task, die eine $VE \in PK_j$ besitzt, verliert ihre Vorhaltung vor einer anderen Task, der im Schritt β der Prozedur *vstrat* ein Prozessor aus PK_j zugewiesen wurde.

Wir werden nun die letztgenannten Bedingungen näher untersuchen:

ad a) Da die Task T_v , $v \in \{1, \dots, \tau\}$, keinen Prozessor besitzt, gilt in einer Umgebung des Zeitpunktes $t \in [0, \infty)$

$$l_v(t) = \text{const.}$$

Mithin verschwindet der Spielraum von T_v wegen

$$s_v(t) = a_v(t) - l_v(t) = (t_z^{(v)} - l_v(t)) - t$$

zum Zeitpunkt $t_z^{(v)} - l_v(t)$.

ad b) Die Task T_μ verliert zum Zeitpunkt $t_{\mu v}$ ihre Vorhaltung vor T_v , $\mu, v \in \{1, \dots, \tau\}$, wenn

$$s_\mu(t_{\mu v}) = a_v(t_{\mu v}) \text{ gilt. Wegen}$$

$$s_\mu(t_{\mu v}) = t_z^{(\mu)} - t_{\mu v} - l_\mu(t_{\mu v}) = t_z^{(v)} - t_{\mu v} = a_v(t_{\mu v})$$

ist $t_{\mu v}$ aus der Gleichung $l_\mu(t_{\mu v}) = t_z^{(\mu)} - t_z^{(v)}$ zu bestimmen. Weil $l_\mu(t)$ vom Zeitpunkt t_μ der Zuweisung eines Prozessors aus PK_j an T_μ linear mit t abnimmt, erhalten wir

$$t_{\mu v} = l_\mu(t_\mu) + t_\mu - t_z^{(\mu)} + t_z^{(v)}.$$

Die unter (a) und (b) aufgetretenen kritischen Zeitpunkte können zu einer Menge zusammengefaßt werden, deren Minimum, d.h. der nächste eintretende Zeitpunkt, zu dem die Prozedur $vstrat$ aufgerufen werden muß, in das Vergleichsregister eines Weckers geladen werden kann.

4.6 Dispatching-Algorithmen

Nachdem wir bisher die Aufgaben des Dispatchers umrissen, die Vorgehensweise zur Verhinderung von Deadlocks festgelegt, für die hier vorgestellte Rechnerstruktur geeignete

Dispatching-Strategien ausgewählt und schließlich Bedingungen für die Betriebsmittelvergabe angegeben haben, können wir nun darangehen, die Dispatcher-Algorithmen zu formulieren.

Wir beginnen die Darstellung dieser Algorithmen mit der Beschreibung der zu bearbeitenden Daten und der Funktionsweisen einiger Hilfsprozeduren. Die bereits in den letzten Abschnitten eingeführten Größen und Zustandsvariablen werden in den angegebenen Bedeutungen verwendet.

Für jede zu bearbeitende Task legt der Dispatcher einen Verbund der folgenden Form an:

```
mode tcb=struct( string tid, real ta, real tl, string stat,  
                string par, integer dim, [1:dim]integer ek,  
                [1:n]boolean bm, string mt, ref tcb next) .
```

Dabei seien tid der Identifikator der Task, ta ihre Zeitbedingung, d.h. der Zeitpunkt, zu dem die Task beendet sein muß, und tl ihre maximale Laufzeit. Die Variable stat gibt an, in welchen Zustand sich die Task befindet. Bei Beginn der Ausführung wird der Parameter par an das Programm übergeben. Dazu gehören auch konfigurationsabhängige Angaben über die von der Task zu benutzenden Verbindungseinheiten. Das Feld ek hat die Dimension dim und enthält die Nummern der Klassen aus BK, aus denen jeweils genau ein Betriebsmittel zur Ausführung der Task benötigt wird. Der Task zugewiesene Betriebsmittel sind durch Eintragungen an den entsprechenden Stellen des Feldes bm gekennzeichnet. Ist die Task Subtask einer anderen, dann ist deren Identifikator als mt verfügbar. Die Verbunde der ablaufbereiten

und ablaufenden Tasks werden nach wachsendem t_a sortiert und mit next zur Liste bt verkettet, auf deren Anfang die Variable $kopf$ vom Typ ref tcb zeige.

Die Prüfung der zeitgerechten Verarbeitbarkeit einer Taskmenge setzt eine nach Antwortzeiten geordnete Liste dieser Tasks voraus. Wegen der Äquivalenz der Sortierung von Tasks nach Zeitbedingungen und nach Antwortzeiten können bt bzw. Teillisten davon für den genannten Zweck verwendet werden.

Um die zeitgerechte Verarbeitbarkeit von Taskmengen überprüfen zu können, müssen jederzeit aktuelle Angaben über die noch benötigten Restlaufzeiten zur Verfügung stehen. Darum muß der Dispatcher einen Scanner enthalten, der die Variablen t_l in den tcb's der sich in Ausführung befindlichen Tasks fortlaufend dekrementiert. Eine andere Möglichkeit, diese Funktion zu implementieren, wäre, als Speicherzellen für die Restlaufzeiten Abwärtszähler vorzusehen.

Die Operation \cup_A füge einen Task-Kontroll-Block (tcb) entsprechend der angegebenen Zeitbedingung in die Liste bt ein und modifiziere gegebenenfalls $kopf$. Ein nicht mehr benötigter tcb werde durch die Prozedur cancel gelöscht. Die zurückgestellten Tasks fassen wir zur Menge st zusammen. Im Feld $[1:m+1, 1:\tau]$ ref tcb al sind die ablaufenden Tasks eingetragen, die eine VE benötigen. Den nach ansteigendem t_a geordneten Tasks $al[i, j]$, $j=1, \dots, lg[i]$, $i \in \{1, \dots, m\}$, werden die Prozessoren der Klasse PK_i gemäß der Vorhaltestrategie zugewiesen. Die Elemente $al[m+1, 1]$ und $lg[m+1]=0$ werden nur als Hilfsvariable benutzt. Jeder Klasse PK_i , $i=1, \dots, m$, ist

ein Wecker zugeordnet, in dessen weckerregister[i] die oben behandelten kritischen Zeitpunkte geladen werden, bei deren Eintreten dann das weckersignal[i] erzeugt wird. Bei der Zuweisung des Betriebsmittels E_i , $i \in \{1, \dots, n\}$, an eine Task wird deren Zeitbedingung t_a als $az[i]$ und die Referenz zu ihrem tcb als $tk[i]$ gespeichert. Um im Falle, daß eine ablaufbereite Task nicht mehr zeitgerecht verarbeitbar ist, eine Fehlerbehandlungsprozedur starten zu können, führt der Dispatcher eine nach wachsendem kt geordnete Liste sl von Verbunden des Typs

```
mode ive=struct( real kt, integer anz, [1:anz]ref tcb lt,  
                ref ive next) ,
```

die über next verkettet sind. Die anz im Feld lt eingetragenen Tasks verlieren zum Zeitpunkt kt ihren Spielraum. Diese Zeitpunkte sind in das Vergleichsregister des Spielraumweckers zu laden, der die Signale zum Start der Fehlerprozedur erzeugt. Die Routinen insert und delete dienen zum Einfügen einer Task in die bzw. zum Entfernen einer Task aus der Datenstruktur sl und setzen gegebenenfalls das spielraumweckerregister. Die Prozedur error führe zur Behandlung verschiedener durch Parameter spezifizierter Fehlersituationen geeignete Maßnahmen durch, auf die wir hier jedoch nicht näher eingehen wollen. Das erste Element einer linearen Liste stelle die Funktionsprozedur first zur Verfügung. Vom Task-Scheduler kommende Daten werden mit get eingelesen. Durch Aufruf der Routine message kann der Dispatcher Nachrichten an andere Einheiten absenden. Die aktuelle Uhrzeit ist als t verfügbar. Die Zustands-

variable eof zeigt das Ende der vom Task-Scheduler stammenden Eingabesätze an. Als Hilfsgrößen werden die Felder $[m+1:m+n_2+n_3]$ boolean cl, $[1:m+1]$ integer lg, $[1:\tau]$ integer in, $[1:\tau]$ ref tcb tf und $[1:\tau]$ ref tcb v, die Mengen blm und pr von Elementen des Typs ref tcb, die Skalare boolean log, integer i,j,k,p,r,s, real q,i0,i1,i2,tn und ref tcb ptr,zgr sowie Variable gleichen Namens und Typs wie die Komponenten von tcb benötigt.

Beim Anlauf des Systems werden die Daten des Dispatchers durch die Ausführung des Abschnittes

```
[ZM:=ZM':=(L,...,L) , az:=(∞,...,∞) , tk:=(nil,...,nil) ,
lg:=(0,...,0) , weckerregister:=(∞,...,∞) ,
spielraumweckerregister:=∞ , kopf:=bt:=st:=nil ,
sl:={loc ive:=(∞,0,nil,nil)}]
```

initialisiert.

Die einzelnen Dispatcher-Algorithmen werden danach durch Signale aktiviert. Die von diesen Programmen aufgerufenen Prozeduren sollen nun zuerst angegeben werden.

Die Routine allocate weist ablaufbereiten Tasks die von ihnen benötigten, jedoch noch nicht in ihrem Besitz befindlichen Betriebsmittel mit Ausnahme von Verarbeitungseinheiten durch Setzen der entsprechenden Zustandsvariablen zu und überführt sie so in den Zustand "ablaufend". Die Vergabe von Prozessoren nimmt die Prozedur vstrat vor, während Tasks, die keine Verarbeitungseinheiten brauchen, schon von allocate in die Ausführungsphase überführt werden.

Es ist möglich, daß nicht alle Tasks aus einer zeitgerecht

verarbeitbaren Taskmenge, deren Prozessorzuteilung bzgl. eines symmetrischen (Teil-)Mehrprozessorsystems nach der Vorhaltestrategie durchgeführt wird, während der ganzen Zeit eine VE besitzen, die sie sich im Zustand "ablaufend" befinden. Deshalb teilen wir diesen in die beiden Unterzustände "ablaufend1" und "ablaufend2" ein. Der erstere ist dadurch gekennzeichnet, daß die sich in ihm befindlichen Tasks alle Betriebsmittel aus $H_2 \cup H_3$ besitzen und auf die Zuweisung einer VE durch die Prozedur vstrat warten, wodurch sie nach "ablaufend2" übergehen. Diese Einteilung wurde gewählt, weil die Unterzustände für solche Tasks zusammenfallen, die auf (Teil-)Einprozessorsystemen verarbeitet werden bzw. überhaupt keine VE benötigen. Die Routine allocate kennzeichnet jedoch auch die letztgenannten als "ablaufend2", damit dem Dispatcher allein durch die Zustandsangaben bekannt ist, für welche Tasks die Restlaufzeiten dekrementiert werden müssen.

```
proc(ref tcb) allocate=(ref tcb ptr):  
[co Die benötigten Betriebsmittel wurden zuvor freigemacht co  
  stat of ptr:="ablaufend1" , delete(ptr) ,  
  j:=if bm of ptr=(false,...,false) then m+n2+n3 else m+n2 fi;  
  for i from 1 by 1 to dim of ptr while ek[i] of ptr<j do  
    if ek[i] of ptr>m then  
      k:=ek[i] of ptr+n1-m ; z[k]:=z'[k]:=false ,  
      bm[k] of ptr:=true , az[k]:=ta of ptr , tk[k]:=ptr  
    fi  
  od ;
```

if ek[1] of ptr>m then

stat of ptr:="ablaufend2" ; Starte die Programmausführung

fi .

Die Ausführung einer Task wird durch die Prozedur deallocate unterbrochen. Weiterhin werden der Task die in ihrem Besitz befindlichen Betriebsmittel der Klassen H_j , $j=1, \dots, 1 \in \{1, 2, 3\}$, entzogen. Diese Betriebsmittel werden durch geeignetes Setzen der ihnen zugeordneten Zustandsvariablen als frei gemeldet.

proc(ref tcb,integer) deallocate=(ref tcb ptr,integer i):

[j:=case i in $n_1, n_1+n_2, n_1+n_2+n_3$ esac ;

Stoppe alle Einheiten E_k mit bm[k] of ptr=true und rette ihre Stati, $k=1, \dots, j$;

for k from 1 by 1 to j do

if bm[k] of ptr then

bm[k] of ptr:=false , z[k]:=z'[k]:=true ,

az[k]:=∞ , tk[k]:=nil

fi

od ;

stat of ptr:=case i in "ablaufend1", "ablaufbereit", "bekannt"

esac .

Für $k \in \{1, \dots, m\}$ überprüft das folgende Unterprogramm die zeitgerechte Verarbeitbarkeit der nach aufsteigenden Zeitbedingungen geordneten Tasks $al[k, i]$, $i=1, \dots, lg[k]+1$, auf dem Teilmultiprozessor PK_k anhand der für $|PK_k|=1$ in [14] bzw. für $|PK_k|>1$ in [14, 19] angegebenen Bedingungen. Im Falle $k=m+1$, d.h. wenn die Task $al[m+1, 1]$ keine VE benötigt, wird lediglich geprüft, ob der Spielraum dieser Task nicht negativ ist.

```
proc(integer) zeitgerecht=(integer k)boolean:
[if k>m then t+t1 of al[k,1]<ta of al[k,1] else
  i:=|PKk| , j:=lg[k]+1 , i0:=i2:=0 , log:=false ;
  if i=1 then
    for r from 1 by 1 to j do
      i0:=i0+t1 of al[k,r] ;
      if ta of al[k,r]-t<i0 then goto a fi
    od
  else
    for r from 1 by 1 to j do
      i0:=i0+t1 of al[k,r] , i1:=p:=0 ;
      for s from r+1 by 1 to j do
        q:=ta of al[k,r]-ta of al[k,s]+t1 of al[k,s] ;
        if q>0 then p:=p+1 else q:=0 fi ; i1:=i1+q
      od ;
      if r<i1+p+r<i then
        if ta of al[k,r]-t<(i0+i1)/(r+p) then goto a fi
      else
        if r>j-i+1 then
          i2:=i2+ta of al[k,r-1]-t ;
          if ta of al[k,r]-t<(i0+i1-i2)/(j-r+1) then goto a fi
        else
          if ta of al[k,r]-t<(i0+i1)/1 then goto a fi
        fi
      fi
    od
  fi ; log:=true ; a:log
fi] .
```

Unter den Voraussetzungen, daß die nach aufsteigenden Zeitbedingungen geordnete Taskmenge $\{a_l[k,r] \mid r=1,\dots,l_g[k]\}$ auf dem Teilmultiprozessor PK_k zeitgerecht verarbeitbar ist und daß ihre Elemente alle benötigten Betriebsmittel aus $H_2 \cup H_3$ bereits besitzen, führt die Prozedur $vstrat$ für $k \in \{1,\dots,m\}$ bzgl. obiger Tasks den in [15] um die Vorhal-tungsprüfung erweiterten Vorhaltealgorithmus [19] durch. Es werden die Menge pr von Tasks, denen Prozessoren zugewiesen werden sollen, und entsprechend der Ausführungen im vorangegangenen Abschnitt ein neuer kritischer Zeitpunkt bestimmt, der in das weckerregister $[k]$ geladen wird. Allen Tasks, die nicht zu pr gehören, jedoch eine VE der Klasse PK_k besitzen, wird der Prozessor entzogen. Schließlich werden allen jenen Tasks aus pr durch Setzen der Zustandsvariablen Verarbeitungseinheiten zugeordnet, die noch keine besitzen. Damit gehen diese Tasks in die Ablaufphase über.

proc(integer) vstrat=(integer k):

```

i:=1 , j:=0 , p:=lg[k] , tn:=∞ , pr:=∅ ;
  for r from 1 by 1 to p do tf[r]:=al[k,r] od ;
  β:j:=j+1 , pr:=prv{tf[i]} , q:=ta of tf[i] ; tf[i]:=nil ;
  s:=0 ;
  for r from 1 by 1 to p do
    if tf[r]≠nil∧ta of tf[r]-tl of tf[r]<q then
      s:=s+1 ; v[s]:=tf[r] , in[s]:=r
    fi
  od ;
  if i=1 then
    for r from 1 by 1 to s while j<| $PK_k$ | do

```

```
if ta of v[r]-tl of v[r]=t then
  j:=j+1 , pr:=prv(v[r]) ; v[r]:=tf[in[r]]:=nil ,
  tn:=min{tn,q}
fi
od
fi ;
for r from 1 by 1 to s while j<|PKk| do
  if v[r]≠nil then
    j:=j+1 , pr:=prv(v[r]) , tf[in[r]]:=nil ,
    tn:=min{tn,tl of v[r]+t-ta of v[r]+q}
  fi
od ;
if p>j then
  if j<|PKk| then
    while tf[i]=nil do i:=i+1 od ; goto §
  else
    for r from 1 by 1 to p do
      if tf[r]≠nil then tn:=min{tn,ta of tf[r]-tl of tf[r]} fi
    od
  fi
fi ;
weckerregister[k]:=tn ; r:= $\sum_{p=1}^{k-1} |PK_p|$  ; s:=r+|PKk| ; r:=r+1 ;
for p from r by 1 to s do
  if tk[p]≠nil then
    if tk[p]∈pr then pr:=pr\{tk[p]} else deallocate(tk[p],1) fi
  fi
od ;
for all ptrepr do
```

```
while tk[r]≠nil do r:=r+1 od ;  
tk[r]:=ptr , az[r]:=ta of ptr , z[r]:=z'[r]:=false ,  
bm[r] of ptr:=true , stat of ptr:="ablaufend2" ;  
Starte die Verarbeitungseinheit Er ; r:=r+1
```

od .

Immer wenn Tasks in den Zustand "ablaufbereit" übergehen oder wenn Betriebsmittel wieder frei oder entziehbar werden, wird die zentrale Dispatcher-Routine, die Prozedur check, aufgerufen. Auf der Grundlage der Antwortzeitordnung und des aktuellen Betriebsmittelbelegungszustandes bestimmt sie aus der Liste bt die lauffähigen und vorzuziehenden Tasks. Für jede Task wird geprüft, ob die von ihr benötigten Betriebsmittel aus $H_2 \vee H_3$ verfügbar sind. Weil die entsprechenden Betriebsmittelklassen die Mächtigkeit 1 besitzen und die Liste bt in der Reihenfolge aufsteigender Zeitbedingungen durchlaufen wird, ist letzteres genau dann der Fall,

wenn die geforderten Betriebsmittel entziehbar sind
und noch nicht für bereits vorher untersuchte Tasks
vorgemerkt wurden
oder wenn sie nicht entziehbar sind
und der betreffenden Task schon zugeteilt worden sind.

Gilt die Verfügbarkeit für alle benötigten Betriebsmittel aus $H_2 \vee H_3$, so werden diese, und ansonsten nur die aus H_3 , als vorgemerkt gekennzeichnet. Für jeden Teilmultiprozessor PK_i , $i=1, \dots, m$, erstellt check eine zeitgerecht verarbeitbare freie Taskmenge. Dabei müssen eventuell Tasks auf die nächst schnelleren Klassen von Verarbeitungseinheiten ver-

lagert werden, wodurch sich ihre Restlaufzeiten reduzieren, oder - falls solcher Lastabwurf nicht mehr möglich ist - aus der Liste bt entfernt und fehlerhaft beendet werden. Die ablauffähigen Tasks werden zur Belegungsmenge blm zusammengefaßt. Ablaufende Tasks, die nicht zu dieser Menge gehören, werden nach Entzug der von ihnen belegten Betriebsmittel aus $H_1 \cup H_2$ in die Spielraumliste sl eingefügt. Schließlich überführt $check$ die lauffähigen Tasks durch Betriebsmittelzuweisung und Aufruf von $vstrat$ in den Zustand "ablaufend".

proc $check =$

(: γ : $ptr := \text{kopf}$, $blm := \emptyset$, $lg := (0, \dots, 0)$,

for i from $m+1$ by 1 to $m+n_2+n_3$ do $cl[i] := Z'[i+n_1-m]$ od ;

while $ptr \neq nil$ do

if $\frac{}{(tid \text{ of } ptr, EK_1) \in B}$ $cl[i] \vee \neg Z'[i+n_1-m] \wedge tk[i+n_1-m] = ptr$
 $i = m+1, \dots, m+n_2+n_3$

then

$k := \min\{ek[1] \text{ of } ptr, m+1\}$; $al[k, lg[k]+1] := ptr$;

while $\neg \text{zeitgerecht}(k)$ do

$delete(ptr)$;

if $k > m$ then

$deallocate(ptr, 3)$, $bt := bt \setminus \{ptr\}$;

$error(ptr, \text{"Nicht mehr verarbeitbar"})$; goto γ

else

$k := ek[1] \text{ of } ptr := k+1$; $tl \text{ of } ptr := \lambda[k] \times tl \text{ of } ptr$;

$insert(ptr)$, $al[k, lg[k]+1] := ptr$,

if $stat \text{ of } ptr = \text{"ablaufend2"}$ then $deallocate(ptr, 1)$ fi

fi

```

od ;
    blm:=blmv{ptr} , j:=m+1 , if k≤m then lg[k]:=lg[k]+1 fi
else j:=m+n2+1
fi ;
    for i from 1 by 1 to dim of ptr do
        if ek[i] of ptr>j then cl[ek[i] of ptr]:=false fi
    od ;
    ptr:=next of ptr
od ;
    for i from 1 by 1 to n1+n2 do
        if tk[i]≠nil then
            if tk[i]≠blm then insert(tk[i]) ; deallocate(tk[i],2) fi
        fi
    od ;
    for all ptr∈blm with stat of ptr="ablaufbereit" do
        allocate(ptr)
    od ;
    for i from 1 by 1 to m do if lg[i]>0 then vstrat(i) fi od ).

```

Wir kommen nun zu den durch Signale und Zustandsänderungen aktivierten Dispatcher-Algorithmen, die die oben angegebenen Prozeduren aufrufen.

Nach einem load-Signal liest der Dispatcher die vom Task-Scheduler ausgegebenen Daten ein und führt die eingeplanten Task-Operationen durch. So werden bei einer Aktivierung ein Task-Kontroll-Block angelegt und in die Liste bt einsortiert sowie die Task in die Struktur sl eingefügt und durch Aufruf von check versucht, sie in den Zustand "ablaufend" zu über-

führen. Zu terminierenden Tasks werden die Betriebsmittel entzogen und die auf sie bezogenen Eintragungen in bt, st oder sl werden gelöscht. Zur Suspendierung werden ablaufenden Tasks die in ihrem Besitz befindlichen Betriebsmittel aus $H_1 \cup H_2$ entzogen, während die Eintragungen ablaufbereiter Tasks in sl gelöscht werden. Der tcb solcher Tasks wird aus bt entfernt und der Menge st hinzugefügt. Sollen suspendierte Tasks fortgesetzt werden, dann wird die letztgenannte Operation rückgängig gemacht und danach wie bei der Aktivierung verfahren.

on load do

while ¬eof do

get(i) ;

case i in

[co Activate co

get(tid,ta,tl,par,dim,ek,mt) ;

ref tcb ptr = loc tcb := (tid,ta,tl,"ablaufbereit",par,dim,
ek,(false,...,false),mt,nil) ;

bt := bt ∪ {ptr} , insert(ptr) ; check] ,

[co Suspend co

get(tid) ;

ptr := zgrebt with tid of zgr = tid ;

if ptr ≠ nil then

if stat of ptr = "ablaufbereit" then delete(ptr)

else deallocate(ptr,2)

fi ;

bt := bt \ {ptr} , st := st ∪ {ptr} ,

stat of ptr := "zurückgestellt" , next of ptr := nil

```
fi],  
[co Continue co  
  get(tid) ;  
  ptr:=zgrest with tid of zgr=tid ;  
  if ptr#nil then  
    st:=st\{ptr} , bt:=btA\{ptr} , stat of ptr:="ablaufbereit",  
    insert(ptr) ; check  
  fi],  
[co Terminate co  
  get(tid) ;  
  ptr:=zgrebt with tid of zgr=tid ;  
  if ptr#nil then  
    deallocate(ptr,3) , delete(ptr) , bt:=bt\{ptr}  
  else  
    ptr:=zgrest with tid of zgr=tid ;  
    if ptr#nil then deallocate(ptr,3) , st:=st\{ptr} fi  
  fi ;  
  message(tid of ptr,"Terminierung") ; cancel(ptr)]  
esac  
od  
od .  
Ablaufende Tasks können den Dispatcher durch ein Supervisor-  
Call-Signal veranlassen, sie zu terminieren bzw. zu suspen-  
dieren, wobei diese Zustandsübergänge wie nach entsprechen-  
den Aufrufen durch den Task-Scheduler ablaufen.  
on svc do  
  get(ptr,i) ;  
  case i in
```

```
[co Normales Taskende co
  deallocate(ptr,3) , bt:=bt\{ptr} ,
  message(tid of ptr,"Normales Taskende") ; cancel(ptr)] ,
[co Suspend co
  deallocate(ptr,2) , bt:=bt\{ptr} , st:=stv{ptr} ,
  stat of ptr:="zurückgestellt" , next of ptr:=nil
  co Vor dem svc muß die Task ihre Fortsetzungsbedingung
    eingeplant haben co] ,
out
[co Fehlerhaftes Taskende co
  deallocate(ptr,3) , bt:=bt\{ptr} ,
  message(tid of ptr,"Fehlerhaftes Taskende") ;
  co Der tcb ptr wird nicht gelöscht, da er noch von der
    Fehlerbehandlungsprozedur benötigt wird co
    error(ptr,"Fehlerhaftes Taskende")]
esac
od .
```

Die Prozedur check ist aufzurufen, wenn ein Betriebsmittel wieder verfügbar wird:

on $Z'[i]=+$ do check od , $i=1, \dots, n$.

Zu jedem kritischen Zeitpunkt der Vorhaltestrategie muß die Routine vstrat für den entsprechenden Teilmultiprozessor aufgerufen werden:

on weckersignal[i] do vstrat(i) od , $|PK_i| > 1$, $i=1, \dots, m$.

Der Spielraumwecker zeigt an, wann ablaufbereite Tasks ihren Spielraum verloren haben. Daraufhin werden die in Frage kommenden Tasks entweder auf schnellere Verarbeitungseinheiten verlagert oder fehlerhaft terminiert, wenn Lastab-

wurf nicht mehr möglich ist. Weiterhin wird das erste Element aus der Liste sl entfernt und das spielraumweckerregister neu gesetzt.

```
on spielraumweckersignal do
  log:=false ;
  for i from 1 by 1 to anz of first(sl) do
    ptr:=lt[i] of first(sl) ;
    if ek[1] of ptr>m then
      deallocate(ptr,3) , bt:=bt\{ptr} ;
      error(ptr,"Nicht mehr verarbeitbar")
    else
      ek[1] of ptr:=ek[1] of ptr+1 , log:=true ;
      t1 of ptr:= $\lambda$ [ek[1] of ptr]×t1 of ptr ; insert(ptr)
    fi
  od ;
  sl:=sl\{first(sl)} ;
  spielraumweckerregister:=kt of first(sl) ;
  if log then check fi
od .
```

Bei der nun folgenden Abschätzung der Komplexität obiger Algorithmen setzen wir voraus, daß die Möglichkeiten zur Parallelverarbeitung genutzt werden und daß günstige Implementationen gewählt werden, so daß insbesondere die Anzahl der zur Ausführung von \cup_A , delete und insert notwendigen Operationen nicht von der Mächtigkeit von bt bzw. sl abhängt. Dann ist die Komplexität von allocate, deallocate und der durch Supervisor-Call aufgerufenen Routinen gleich

$O(1)$. Die Prozedur zeitgerecht benötigt für $|PK_k|=1$ $O(\lg[k])$ und für $|PK_k|>1$ $O(\lg[k]+2)$ Operationen, $k=1,\dots,m$. Die Komplexität der Vorhaltestrategie wächst linear mit $|PK_k|$, $\lg[k]$ und $|PK_k|\times\lg[k]$, $k=1,\dots,m$. Wie zu Beginn dieses Paragraphen ausgeführt, kann man davon ausgehen, daß im Normalbetrieb keine Systemüberlastungen auftreten. Dann gilt für die Komplexität K der Routine check und ihrer Unterprogramme

$$K \leq |bt| \cdot (\text{const.} + O(\lg[i] + 2)) + O(n_1 + n_2) + O(|blm|) + O(n_1) + \\ + O(|blm| \cdot \max_{k=1,\dots,m} |PK_k|),$$

wobei i durch $\lg[i] = \max_{k=1,\dots,m} \{\lg[k]\}$ bestimmt sei. Die Mächtigkeit von blm ist durch die maximale Anzahl selbständig arbeitender Betriebsmittelgruppen beschränkt. Die nicht von $|bt|$ abhängigen Summanden obiger Formel zeigen den Einfluß einiger Konfigurationsparameter auf die Komplexität von check. Die Ausführung der Activate- und Continue-Routinen erfordert jeweils $K+O(1)$, die der Suspend- und Terminate-Programme jeweils nur $O(1)$ Operationen für jeden Eingabesatz. Schließlich hat der durch das spielraumwecker-signal aktivierte Abschnitt die Komplexität $K+O(|bt|)$.

Kapitel III Beispiele für den Einsatz spezialisierter Prozessoren

Nachdem wir bisher die im ersten Paragraphen eingeführten acht Typen allgemeiner Funktionseinheiten zum Aufbau von Prozeßrechnern beschrieben haben, wollen wir uns nun mit spezialisierten Prozessoren beschäftigen. Darunter sollen für bestimmte Anwendungen eigens entwickelte Module verstanden werden, die ihre Aufgaben weitgehend selbständig bearbeiten. Auf diese Weise wird wiederum zur Reduzierung des Datenübertragungsaufwandes und zur Entlastung der Verarbeitungs- und ablaufsteuernden Einheiten beigetragen. Wegen der Fülle möglicher Einsatzgebiete für spezialisierte Prozessoren können wir hier nur einige Beispiele behandeln. Im Rahmen physikalischer Experimente werden Prozeßrechner häufig zur Erfassung, Verarbeitung und Ausgabe stetiger Zeitfunktionen angewandt. Im Paragraphen 5 wollen wir uns mit geeigneten Methoden zur Aufnahme und rechnerinternen Darstellung solcher Funktionen beschäftigen und im zweiten Teil dieses Kapitels sollen dann entsprechende Module angegeben werden.

§ 5 Mathematische Grundlagen zur Erfassung und Darstellung
empirischer Funktionen mit Hilfe lokaler Integrale

In diesem Abschnitt soll als Vorbereitung und Grundlage für § 6 eine mathematische Theorie zur Erfassung stetiger Zeitfunktionen unter Rauscheinfluß mit Hilfe von Prozeßrechnern angegeben werden. Die Suche nach geeigneten Verfahren, die sich durch gute Rauschunterdrückungseigenschaften auszeichnen, führt auf die Approximation der empirisch gegebenen Funktionen mit Polynom-Splines, zu deren Konstruktion lokale Integrale verwendet werden. Wir werden einige Methoden zur Darstellung experimenteller Funktionen durch Approximationen aus Spline-Unterräumen anführen und ihre Eigenschaften in Form von Sätzen angeben, deren Beweise z.T. [11] bzw. dem Anhang entnommen werden können. Ein Vergleich zeigt schließlich die Überlegenheit dieser Verfahren gegenüber der bisher üblichen Anwendung von Punktfunktionalen.

5.1 Rauschunterdrückende stetige lineare Approximations-
operatoren

Gegeben sei ein physikalisches Experiment, das als Ausgangssignal während des Zeitintervalls $[0, T]$, $T > 0$, die elektrische Spannung $u(t)$ liefere. Letztere setze sich additiv aus der zu erfassenden, stetigen und reellwertigen Zeitfunktion $f: [0, T] \rightarrow \mathbb{R}$ und dem Rauschsignal $r(t)$ zusammen:
 $u(t) = f(t) + r(t)$, $t \in [0, T]$. Das Rauschen lasse sich in Sinus-

schwingungen der Form $s(t) = \rho \cdot \sin(\omega t + \phi)$ zerlegen. Die Darstellung der Funktion f muß sich demnach und auf Grund technischer Beschränkungen auf endlich viele Größen stützen, die das Verhalten von u beschreiben. Mit Hilfe von $n \in \mathbb{N}$ Daten lassen sich nur die Funktionen aus einer Teilmenge von $C[0, T]$ exakt angeben, die von höchstens n Parametern abhängen, während man sich sonst mit Approximationen bzgl. einer solchen Menge begnügen muß. Wird durch theoretische Überlegungen nicht die Verwendung einer bestimmten Funktionenmenge nahegelegt, so ist die Wahl eines n -dimensionalen linearen Unterraums W von $C[0, T]$, der gute Approximationen der übrigen stetigen Funktionen enthält, als darstellbare Funktionenmenge am zweckmäßigsten. Denn nur dann kann zur Erzeugung einer Rechnerdarstellung von u eine lineare Abbildung $L: C[0, T] \rightarrow W$ angegeben werden, die die additive Zusammensetzung von f und der Rauschanteile in den Bildraum W überträgt: $Lu = Lf + Lr$. Damit der Beitrag von Lr zur Funktion Lu aus der Amplitude des Rauschens r abgeschätzt werden kann, muß der lineare Operator L stetig sein. Sei $\{w_1, \dots, w_n\}$ eine Basis von W , dann läßt sich die Abbildung L durch

$$(Lv)(t) = \sum_{i=1}^n l_i(v) \cdot w_i(t), \quad t \in [0, T] \text{ und } v \in C[0, T],$$

darstellen, wobei die l_i , $i=1, \dots, n$, stetige lineare Funktionale auf $C[0, T]$ seien. Jedes dieser Funktionale l kann in der Form einer Integraltransformation

$$lv = \int_0^T v(t) \cdot \lambda(t) dt, \quad v \in C[0, T],$$

geschrieben werden. Die Größen λ sind hierbei Summen stück-

weise stetiger Funktionen und höchstens abzählbar vieler Dirac-Distributionen, die Punktfunktionale erzeugen. Da die Werte $l_i v, i=1, \dots, n$, aus dem Experiment bestimmt werden, ist es wünschenswert, daß sie mit einer bzw. gleichartigen Eingabeeinheiten erfaßt werden können. Die einzelnen Funktionale l_i unterscheiden sich dann nur noch durch die - insbesondere äquidistanten - Zeitpunkte $t_i \in [0, T], i=1, \dots, n$, der Datenübernahme und können mit den Erweiterungen

$$\hat{v}(t) := \begin{cases} v(t), & t \in [0, T] \\ 0 & , \text{sonst} \end{cases} \quad \text{für } v \in C[0, T]$$

und einem einzigen "Kern" ϕ in Form einer Faltung geschrieben werden:

$$l_i v = \int_{\mathbb{R}} \hat{v}(t) \cdot \phi(t - t_i) dt, \quad v \in C[0, T] \quad \text{und } i \in \{1, \dots, n\}.$$

Wir wollen nun untersuchen, welche Gestalt solche Faltungskerne haben, die den Rauschanteil r unter der Abbildung L weitgehend unterdrücken, so daß Lu die Funktion Lf hinreichend gut approximiert. Wegen der Linearität der Faltung genügt es, den Operator auf eine Sinusschwingung der Form $s(t) = \rho \cdot \sin(\omega t + \phi)$ anzuwenden:

$$\begin{aligned} (\phi * s)(t) &= \int_{\mathbb{R}} \phi(\tau - t) \cdot \rho \cdot \sin(\omega \tau + \phi) d\tau = \rho \cdot \int_{\mathbb{R}} \phi(x) \cdot \sin(\omega x + (\omega t + \phi)) dx \\ &= \rho \cdot \int_{\mathbb{R}} \phi(x) \cdot [\sin(\omega x) \cdot \cos(\omega t + \phi) + \cos(\omega x) \cdot \sin(\omega t + \phi)] dx, \\ &\quad t \in [0, T]. \end{aligned}$$

Im folgenden wollen wir uns auf gerade Faltungskerne beschränken, wodurch sich obiger Ausdruck wesentlich vereinfacht:

$$\begin{aligned} (\phi * s)(t) &= \rho \cdot \sin(\omega t + \phi) \cdot 2 \cdot \int_0^{\infty} \phi(x) \cdot \cos(\omega x) dx = s(t) \cdot \sqrt{2\pi} \cdot \tilde{f}(\phi)(\omega), \\ &\quad t \in [0, T]. \end{aligned}$$

Hierbei bezeichne \mathcal{F} die Fourier-Transformation. Aus der letzten Gleichung ersehen wir, wie geeignete Kerne zu konstruieren sind: wird für jede Frequenz $\omega \in [0, \infty)$ ein Dämpfungsverhältnis $d(\omega) = \frac{\phi * s}{s}$ vorgegeben, dann liefert die Anwendung der inversen Fourier-Transformation auf die Funktion $\frac{1}{\sqrt{2\pi}} \cdot d$ einen geraden Faltungskern ϕ . Durch die Wahl von n Zeitpunkten $t_i \in [0, T]$, $i=1, \dots, n$, wird schließlich der Operator L definiert. Um die Existenz der inversen Fourier-Transformation sicherzustellen, müssen die Funktionen d dem Raum $L^1(\mathbb{R}_+)$ angehören. Daraus folgt $d(\omega) \rightarrow 0$ für $\omega \rightarrow \infty$, d.h. hohe Störfrequenzen werden auf jeden Fall stark unterdrückt. Weitere Kriterien bei der Definition von d sind z.B.

- (i) die Ordnung, mit der d für $\omega \rightarrow \infty$ gegen 0 strebt; und
- (ii) ob gewisse Mengen von Störfrequenzen, die einen besonders großen Anteil des Rauschens r ausmachen wie die Netzfrequenz und ihre Harmonischen, herausgefiltert werden sollen.

Bezeichnen wir eine solche Grundfrequenz mit ω_0 , die sich als $\omega_0 = m \cdot \hat{\omega}$ mit $m \in \mathbb{N} \setminus \{0\}$ schreiben läßt, dann ist

$$d(\omega) = \left(\frac{\pi\omega}{\hat{\omega}}\right)^{-1} \cdot \sin\left(\frac{\pi\omega}{\hat{\omega}}\right), \quad \omega \in \mathbb{R}_+,$$

ein einfaches Beispiel einer Funktion aus $L^1(\mathbb{R}_+)$, die für $\omega \rightarrow \infty$ von der Ordnung $\frac{1}{\omega}$ gegen 0 strebt und für alle Argumente $v \cdot \hat{\omega}$, $v \in \mathbb{N} \setminus \{0\}$, d.h. insbesondere für alle Vielfachen von ω_0 , verschwindet. Wegen

$$\mathcal{F}^{-1}\left(\frac{1}{\sqrt{2\pi}} \cdot d\right)(t) = \begin{cases} \frac{\hat{\omega}}{2\pi}, & t \in \left(-\frac{\pi}{\hat{\omega}}, \frac{\pi}{\hat{\omega}}\right) \\ 0, & \text{sonst} \end{cases} =: \frac{\hat{\omega}}{2\pi} \cdot b_0\left(\frac{\hat{\omega}}{2\pi} \cdot t\right), \quad t \in \mathbb{R},$$

reduziert sich die Faltung mit dem erhaltenen Kern auf eine Integration, d.h. für $i=1, \dots, n$ gilt

$$l_i v = \frac{\hat{\omega}}{2\pi} \cdot \int_{t_i - \frac{\pi}{\hat{\omega}}}^{t_i + \frac{\pi}{\hat{\omega}}} \hat{v}(t) dt, \quad v \in C[0, T].$$

Aus der Darstellung von d ersehen wir, daß die Dämpfungsverhältnisse mit $\hat{\omega}$ und deshalb mit wachsender Länge $\frac{2\pi}{\hat{\omega}}$ des Integrationsintervalls abnehmen. Demnach sollte die Integrationszeit so groß wie möglich gewählt werden, was insbesondere zur Unterdrückung kurzer Störimpulse, sogenannter Spikes, beiträgt. Durch Potenzierung von d läßt sich die Dämpfung hoher Frequenzen verstärken. Auf Grund des Faltungstheorems gilt für die inverse Fourier-Transformierte von

$$\begin{aligned} \frac{1}{\sqrt{2\pi}} \cdot d^m, m \in \mathbb{N} \setminus \{0\}, \\ \mathcal{F}^{-1} \left(\frac{1}{\sqrt{2\pi}} \cdot d^m \right) (t) &= \hat{\omega}^m \cdot (2\pi)^{-\frac{m+1}{2}} \cdot \mathcal{F}^{-1} \left[\left(\mathcal{F} \left(b_0 \left(\frac{\hat{\omega}}{2\pi} \cdot \right) \right) \right)^m \right] (t) \\ &= \hat{\omega}^m \cdot (2\pi)^{-\frac{m+1}{2}} \cdot b_0 \left(\frac{\hat{\omega}}{2\pi} \cdot \right) * \dots * b_0 \left(\frac{\hat{\omega}}{2\pi} \cdot \right) (t) \\ &= \hat{\omega} \cdot (2\pi)^{\frac{m}{2} - \frac{3}{2}} \cdot b_{m-1} \left(\frac{\hat{\omega}}{2\pi} \cdot t \right), \quad t \in \mathbb{R}, \end{aligned}$$

wobei die Funktion b_{m-1} der mit $(x)_+^{m-1} := \begin{cases} x^{m-1}, & x \geq 0 \\ 0, & x < 0 \end{cases}$ definierte "B-Spline"

$$b_{m-1}(x) := \frac{1}{(m-1)!} \cdot \sum_{i=0}^m (-1)^i \cdot \binom{m}{i} \cdot \left(x + \frac{m}{2} - i\right)_+^{m-1}, \quad x \in \mathbb{R},$$

vom Grade $m-1$ ist. Einige Eigenschaften dieser Funktionen sind in [11] zusammengestellt. An dieser Stelle wollen wir nur erwähnen, daß $\text{supp } b_{m-1} = [-\frac{m}{2}, \frac{m}{2}]$ gilt, so daß zur Auswertung der Funktionele l_i , $i=1, \dots, n$, jeweils nur die Bildung lokaler Integrale erforderlich ist und daß die Anzahl der in einem entsprechenden Eingabemodul notwendigen Integratoren nicht gleich n ist, sondern nur von m abhängt. Bei

gleichem Aufwand kann die Dämpfung hoher Frequenzen noch wesentlich verstärkt werden, wenn man als Faltungskern ϕ eine Funktion mit kompaktem Träger aus dem Raum

$$S := \{ \psi \in C^\infty(\mathbb{R}) \mid \lim_{|x| \rightarrow \infty} |x|^\mu \cdot \psi^{(\nu)}(x) = 0 \text{ für alle } \mu, \nu \in \mathbb{N} \}$$

der "schnell abnehmenden Funktionen" wählt. Da die Fourier-Transformation S auf S abbildet, nimmt die resultierende Dämpfungsfunktion $\zeta(\phi)$ für $\omega \rightarrow \infty$ exponentiell ab. Als Beispiel für einen solchen Kern, der darüberhinaus auch alle Harmonischen der Grundfrequenz ω herausfiltert, definieren wir mit

$$\psi(x) = \begin{cases} \exp\left(-\frac{1}{x^2-1}\right) & , x \in (-1, 1) \\ 0 & , \text{sonst} \end{cases} \in S$$

und geeigneten Konstanten $a, A \in \mathbb{R}$ die Abbildung

$$\phi := A \cdot \psi\left(\frac{\cdot}{a}\right) * b_0\left(\frac{\omega}{2\pi} \cdot\right) \in S.$$

5.2 Spline-Approximationsverfahren auf der Grundlage

lokaler Integrale

Nachdem wir im letzten Abschnitt gezeigt haben, daß sich lokale Integrale wegen ihrer Rauschunterdrückungseigenschaften zur Erfassung empirischer Funktionen sehr gut eignen, wollen wir nun einige Methoden der numerischen Mathematik zur Weiterverarbeitung der anfallenden Daten vorstellen. Denn während sich die meisten Approximations- und Interpolationsverfahren mit Ausnahme der L^2 -Approximation auf diskrete Funktions- und Ableitungswerte stützen, ist in der Literatur nur wenig über entsprechende Verfahren

auf der Grundlage anderer Funktionale zu finden. Selbst die bei L^2 -Approximationen notwendigen Integrationen werden zu-
meist numerisch, also mit Hilfe einzelner Funktionswerte,
ausgeführt. Zur angenäherten Darstellung stetiger, im Expe-
riment beobachteter Zeitfunktionen wählen wir Räume von
Polynom-Splinefunktionen aus, da sich diese durch eine Reihe
im folgenden aufgeführter Eigenschaften auszeichnen, die die
Konstruktion von Approximationsfunktionen und weitere Aus-
wertungen erleichtern, und da Basiselemente solcher Unter-
räume, die "B-Splines", sich wegen ihrer guten Rauschunter-
drückung bereits als geeignete Faltungskerne erwiesen haben.
Es sind verschiedene Basissysteme für Splineräume bekannt,
wovon die B-Splines für unsere Zwecke auf Grund ihrer kom-
pakten Träger am günstigsten sind. Mithin hängt die Anzahl
der zur Berechnung von Funktions- und Ableitungswerten
notwendigen Operationen nur vom Grade der Splines, jedoch
nicht von der Dimension der Räume ab, die deshalb durchaus
sehr groß sein kann. Splinefunktionen sind aus Polynom-
stücken zusammengesetzt, so daß für die genannten Auswer-
tungen nur Additionen und Multiplikationen benötigt werden
und auch Kurvendiskussionen einfach durchzuführen sind. Der
Einfluß fehlerhafter Koeffizienten der B-Splines bleibt
wegen der Kompaktheit ihrer Träger lokal. Bei der Realis-
ierung bestimmter linearer Operatoren $L: C[0, T] \rightarrow W$ hängt bei
vorgegebenen Funktionalen l_i , $i=1, \dots, n$, von letzteren die
Wahl der Basen $\{w_1, \dots, w_n\}$ von W in der oben angegebenen

Gleichung $(Lv)(t) = \sum_{i=1}^n l_i(v) \cdot w_i(t)$, $t \in [0, T]$ und $v \in C[0, T]$ ab.

Sind die Basiselemente w_i , $i=1, \dots, n$, keine B-Splines, dann führt die Konstruktion der gewünschten Darstellungen auf lineare Gleichungssysteme. Deren Lösung bereitet aber auch für große n keine numerischen Schwierigkeiten, da die entsprechenden Koeffizientenmatrizen bei den hier angewandten Funktionalen l_i , $i=1, \dots, n$, wegen des Verschwindens der B-Splines außerhalb beschränkter Intervalle Bandform haben. Die Forderungen nach gleichmäßiger Rauschunterdrückung und selbständiger Erfassung der Werte der Funktionalen l_i , $i=1, \dots, n$, durch spezialisierte Prozessoren legen nahe, die zugehörigen Zeitpunkte $t_i \in [0, T]$, $i=1, \dots, n$, äquidistant zu wählen. Dementsprechend definieren wir die Partition $\pi_n := \{i \cdot \frac{T}{n} \mid i=0, \dots, n\}$ des Intervalls $[0, T]$. Die bzgl. π_n konstruierten B-Splines lassen sich durch Translationen ineinander überführen. Deshalb werden die erwähnten Bandmatrizen weitgehend symmetrisch mit im wesentlichen konstanten Diagonalen und die Elemente des Polynom-Splinesraums vom Grade $m \in \mathbb{N}$ bzgl. π_n lassen sich schreiben als

$$s(t) = \sum_{i=-m}^{n-1} a_i \cdot b_m\left(\frac{n}{T} \cdot t - i - \frac{m+1}{2}\right), \quad t \in [0, T], \quad a_i \in \mathbb{R}, \quad i=-m, \dots, n-1.$$

Die m zusätzlich in dieser Darstellung auftretenden Basisfunktionen sind zur Erzielung einer auf $[0, T]$ gleichmäßigen Approximationsgüte notwendig. Die zugehörigen Koeffizienten müssen gegebenenfalls aus Randbedingungen berechnet werden.

Zur Approximation stetiger Funktionen $f \in C[0, T]$ durch Elemente des Raumes $\Pi_m\{\pi_n\}$ der Polynom-Splines m -ten Grades, $m \in \mathbb{N}$, bzgl. der Partition π_n , $n \in \mathbb{N}$, geben wir nun drei stetige

lineare Operatoren an.

Im ersten werden die Koeffizienten in obiger Darstellung durch Interpolation der gemessenen lokalen Integrale

$$\int_{i \cdot \frac{T}{n}}^{(i+1) \cdot \frac{T}{n}} f(t) dt, \quad i=0, \dots, n-1, \text{ bestimmt.}$$

Satz 5.1: Seien $m > 1$, $n > 1$, $T > 0$ und $\pi_n := \{i \cdot \frac{T}{n} \mid i=0, \dots, n\}$ eine äquidistante Partition des Intervalls $[0, T]$.

(i) Für alle $f \in C^{m-1}[0, T]$ gibt es genau ein

$$s \in \Pi_{2m}(\pi_n) \cap C^{2m-1}[0, T] \text{ mit}$$

$$\int_{i \cdot \frac{T}{n}}^{(i+1) \cdot \frac{T}{n}} f(t) dt = \int_{i \cdot \frac{T}{n}}^{(i+1) \cdot \frac{T}{n}} s(t) dt, \quad i=0, \dots, n-1, \text{ und}$$

$$f^{(j)}(\tau) = s^{(j)}(\tau), \quad \tau \in \{0, T\} \text{ und } j=0, \dots, m-1.$$

(ii) Mit den Abkürzungen $h := \frac{T}{n}$, $c_i := \int_i^{i+1} b_{2m}(t - \frac{2m+1}{2}) dt$, $i=0, \dots, 2m$, und $d_i^{(j)} := b_{2m}^{(j)}(i - \frac{2m+1}{2})$, $i=1, \dots, 2m$ und $j=0, \dots, m-1$, lassen sich die Koeffizienten in der Darstellung

$$s(t) = \sum_{i=-2m}^{n-1} a_i \cdot b_{2m}\left(\frac{t}{h} - i - \frac{2m+1}{2}\right)$$

aus dem folgenden linearen Gleichungssystem berechnen:

$$\begin{aligned} \sum_{i=-2m}^{-1} a_i \cdot d_{-i}^{(j)} &= h^j \cdot f^{(j)}(0), \quad j=m-1, \dots, 0, \\ \sum_{i=j-2m}^j a_i \cdot c_{j-i} &= \frac{1}{h} \cdot \int_{j \cdot h}^{(j+1) \cdot h} f(t) dt, \quad j=0, \dots, n-1, \\ \sum_{i=n-2m}^{n-1} a_i \cdot d_{n-i}^{(j)} &= h^j \cdot f^{(j)}(T), \quad j=0, \dots, m-1. \end{aligned}$$

(iii) Sei $f \in C^{2m+1}[0, T]$, dann gilt mit nur von m , j und T abhängigen Konstanten $K(m, j, T)$ die Fehlerabschätzung

$$\|D^j(f-s)\|_{\infty} \leq K(m, j, T) \cdot \left(\frac{T}{n}\right)^{2m+1-j} \cdot \|D^{2m+1}f\|_{\infty}, \quad j=0, \dots, 2m.$$

Für $m=1$ gilt insbesondere $K(1, 0, T) = \frac{9+\sqrt{3}}{216}$, $K(1, 1, T) = \frac{1}{3}$

und $K(1,2,T)=1$.

(iv) Weiterhin gilt folgende Minimaleigenschaft:

$$\int_0^T (D^m s)^2(t) dt \leq \int_0^T (D^m g)^2(t) dt$$

für alle $g \in C^m[0,T]$ mit $\int_{i \cdot h}^{(i+1) \cdot h} (f-g)(t) dt = 0, i=0, \dots, n-1$.

Aufschluß über den Einfluß der einzelnen Interpolationsdaten liefert die Betrachtung des entsprechenden unendlichen Interpolationsproblems.

Satz 5.2: Seien $m \geq 1, h > 0$ und $\pi_h := \{(i+\frac{1}{2}) \cdot h \mid i \in \mathbb{Z}\}$ eine äquidistante Partition von \mathbb{R} .

(i) Für alle $f \in C(\mathbb{R}) \cap B(\mathbb{R})$ gibt es genau ein

$$s \in \Pi_{2m}(\pi_h) \cap C^{2m-1}(\mathbb{R}) \text{ mit}$$

$$\int_{(i-\frac{1}{2}) \cdot h}^{(i+\frac{1}{2}) \cdot h} f(t) dt = \int_{(i-\frac{1}{2}) \cdot h}^{(i+\frac{1}{2}) \cdot h} s(t) dt, i \in \mathbb{Z}, \text{ das sich als}$$

$$s(t) = \sum_{i \in \mathbb{Z}} \frac{1}{h} \int_{(i-\frac{1}{2}) \cdot h}^{(i+\frac{1}{2}) \cdot h} f(\tau) d\tau \cdot k\left(\frac{t}{h} - i\right), t \in \mathbb{R},$$

schreiben läßt. Das charakteristische Polynom

$$p_m(x) := \sum_{i=0}^{2m} c_i \cdot x^i \text{ hat } m \text{ einfache Nullstellen}$$

$\{\alpha_i^{(m)} \mid i=1, \dots, m\} \subset (-1, 0)$. Damit kann der Kardinalspline k wie folgt dargestellt werden:

$$k(t) = \sum_{i \in \mathbb{Z}} \sum_{j=1}^m \frac{(\alpha_j^{(m)})^{m-1+|i|}}{p_m'(\alpha_j^{(m)})} \cdot b_{2m}(t-i), t \in \mathbb{R}.$$

(ii) Der Interpolationsoperator hat für $m=1$ bzgl. der Čebyšev-Norm die Norm $\sqrt{3}$.

Für $n \rightarrow \infty$ streben die Elemente der zu den in Satz 5.1(i) angegebenen Funktionalen dualen Basis von $\Pi_{2m}(\pi_h) \cap C^{2m-1}[0,T]$

gegen entlang der t-Achse verschobene Kardinalsplines. Aus deren Darstellung ist ersichtlich, daß der Einfluß der Interpolationsdaten und damit eventuell vorhandener Fehler auf $s(t)$, $t \in \mathbb{R}$, mit zunehmendem Abstand von den entsprechenden Integrationsintervallen exponentiell abnimmt.

Bei dem im folgenden Satz definierten positiven Approximationsoperator wirken sich fehlerhafte Daten

$$\int_{(i-\frac{1}{2}) \cdot h}^{(i+\frac{1}{2}) \cdot h} f(t) dt, \quad i \in \mathbb{Z}, \text{ nur lokal aus.}$$

Satz 5.3: Seien $m \in \mathbb{N}$, $h > 0$ und $\pi_h := \{(i+\frac{1}{2}) \cdot h \mid i \in \mathbb{Z}\}$ eine äquidistante Partition von \mathbb{R} sowie $I \subset \mathbb{R}$ ein kompaktes Intervall.

Dann gilt für den Operator

$$L_{m,h}: \begin{cases} C(\mathbb{R}) \rightarrow \Pi_m\{\pi_h\} \cap C^{m-1}(\mathbb{R}) \\ f \mapsto \sum_{i \in \mathbb{Z}} \frac{1}{h} \cdot \int_{(i-\frac{1}{2}) \cdot h}^{(i+\frac{1}{2}) \cdot h} f(t) dt \cdot b_m\left(\frac{\cdot - i \cdot h}{h}\right) \end{cases} :$$

- (i) $L_{m,h}$ ist linear und positiv.
- (ii) Für die Monome π_i , $i=0,1,2$, gilt $L_{m,h}\pi_i = \pi_i$, $i=0,1$, und $L_{m,h}\pi_2 = \pi_2 + \frac{m+2}{12} \cdot h^2 \cdot \pi_0$ falls $m \geq 2$.
- (iii) $L_{m,h}f$ konvergiert gegen f für $h \rightarrow 0$.
- (iv) Der Operator $L_{m,h}: C(\mathbb{R}) \rightarrow C(\mathbb{R})$ ist für $m \geq 2$ stetig und hat die Norm 1.
- (v) Der Operator $L_{m,h}$ ist variationsvermindernd.
- (vi) $\text{Var}_I L_{m,h}f \leq \text{Var}_I f$, $f \in C(I) \cap BV(I)$ und $m \geq 1$.
- (vii) $\|L_{m,h}f - f\|_{\infty, I} \leq \frac{m+14}{12} \cdot \omega(f, h)$ für $m \geq 2$.
- (viii) Für $m \geq 1$, $j \in \{0, \dots, m-1\}$ und $f \in C^{j+2}$ gilt die scharfe Fehlerabschätzung

$$\| D^j(L_{m,h} f - f) \|_{\infty, I} \leq K(m, j) \cdot h^2 \cdot \| D^{j+2} f \|_{\infty, I}$$

mit $K(m, j) = \frac{m+2}{24}$ für $j \in \{0, \dots, m-2\}$ und $K(m, m-1) = \frac{m+3}{24}$.

$$(ix) \quad \| D^m(L_{m,h} f - f) \|_{\infty, I} \leq \left(\frac{1}{2} + \frac{2}{(m+2)I} \right) \cdot \sum_{k=0}^{\lfloor \frac{m+1}{2} \rfloor} (-1)^k \cdot \binom{m+1}{k} \cdot \left(\frac{m+1}{2} - k \right)^{m+2} \times$$

$$\times h \cdot \| D^{m+1} f \|_{\infty, I}, \quad m \in \mathbb{N} \text{ und } f \in C^{m+1}(I).$$

Bemerkung: Bei Anwendung des Operators $L_{m,h}$ auf Funktionen aus $C(I)$ und insbesondere $C[0, T]$ sind diese auf eine Umge-
bung von I geeignet fortzusetzen, um die Koeffizienten
aller B-Splines, deren Träger mit I einen nichtleeren Durch-
schnitt haben, zu bestimmen.

Die beiden bisher behandelten Verfahren dienen zur Konstruk-
tion von Approximationsfunktionen mit Hilfe der Daten

$\frac{1}{2\pi} \cdot (b_0(\frac{1}{2\pi}) * u)(t_i)$, $i=1, \dots, n$. Sind die Werte
 $\frac{1}{2\pi} \cdot (b_j(\frac{1}{2\pi}) * u)(t_i)$, $i=1, \dots, n$ und $j > 1$, gegeben, dann
können wir bzgl. jedes Polynom-Splineräume $\Pi_m\{\pi_n\} \cap C^{m-1}[0, T]$,
 $m \in \mathbb{N}$, das entsprechende verallgemeinerte Interpolations-
problem lösen. Wir wollen nun den Fall $j=m$ betrachten, wo
dieses mit der L^2 -Approximationsaufgabe zusammenfällt.

Satz 5.4: Seien $m > 1$, $n > 1$, $T > 0$ und $\pi_n := \{i \cdot \frac{T}{n} \mid i=0, \dots, n\}$ eine
äquidistante Partition des Intervalls $[0, T]$.

(i) Für alle $f \in C[0, T]$ gibt es genau eine beste L^2 -Approx-
mierende $s \in \Pi_m\{\pi_n\} \cap C^{m-1}[0, T]$.

(ii) Sei $h := \frac{T}{n}$. Die Koeffizienten in der Darstellung

$$s(t) = \sum_{i=-m}^{n-1} a_i \cdot b_m\left(\frac{t}{h} - i - \frac{m+1}{2}\right), \quad t \in \mathbb{R},$$

sind die Lösungen des linearen Gleichungssystems

$$\sum_{i=-m}^{n-1} a_i \cdot \int_0^T b_m\left(\frac{t}{h}-i-\frac{m+1}{2}\right) \cdot b_m\left(\frac{t}{h}-j-\frac{m+1}{2}\right) dt =$$

$$\int_0^T f(t) \cdot b_m\left(\frac{t}{h}-j-\frac{m+1}{2}\right) dt, \quad j=-m, \dots, n-1,$$

dessen bandförmige Koeffizientenmatrix symmetrisch und positiv definit ist.

- (iii) Sei $f \in C^{m+1}[0, T]$, dann gilt mit nur von m, j und T abhängigen Konstanten $K(m, j, T)$ die Fehlerabschätzung
- $$\|D^j(f-s)\|_{\infty} \leq K(m, j, T) \cdot \left(\frac{T}{h}\right)^{m+1-j} \cdot \|D^{m+1}f\|_{\infty}, \quad j=0, \dots, m.$$

Auch bei diesem Verfahren nimmt der Einfluß der gemessenen Daten und eventuell darin enthaltener Fehler auf $s(t)$, $t \in \mathbb{R}$, mit zunehmendem Abstand von den Integrationsintervallen exponentiell ab. Dies folgt aus der Betrachtung des unendlichen L^2 -Approximationsproblems bzgl. $\Pi_m\{\pi_h\} \cap C^{m-1}(\mathbb{R})$, $h > 0$, dessen Kardinalspline \hat{k} mit den Bezeichnungen aus Satz 5.2(i) die Darstellung

$$\hat{k}(t) = \sum_{i \in \mathbb{Z}} \sum_{j=1}^m \frac{(\alpha_j^{(m)})^{m-1+|i|}}{p_m'(\alpha_j^{(m)})} \cdot b_m\left(\frac{t}{h}-i\right), \quad t \in \mathbb{R},$$

hat.

Wir beschließen diesen Abschnitt mit einigen Anmerkungen zur Lösung der linearen Gleichungssysteme aus den Sätzen 5.1(ii) und 5.4(ii). Wie oben bereits erwähnt, sind die entsprechenden Koeffizientenmatrizen abgesehen von den oberen linken und den unteren rechten Ecken bandförmig und symmetrisch mit konstanten Diagonalen, da die B-Splines durch Translationen entlang der t -Achse ineinander überführt

werden können, kompakte Träger haben und bzgl. der Mittelpunkte dieser Träger gerade Funktionen sind. Als Normalmatrix ist die Koeffizientenmatrix aus Satz 5.4(ii) positiv definit und läßt sich demnach nach Cholesky symmetrisch in das Produkt dreieckiger Bandmatrizen zerlegen. Nachdem die Matrix aus Satz 5.1(ii) durch elementare Operationen in Bandform überführt worden ist, kann auch hier eine Dreieckszerlegung durchgeführt werden. Wegen der weitgehenden Konstanz der Diagonalen der Koeffizientenmatrizen konvergieren die Elemente in jeder Diagonalen der resultierenden Dreiecksmatrizen mit wachsendem Zeilenindex i . Sei $\alpha_1^{(m)}$ die betragsmäßig größte Nullstelle des charakteristischen Polynoms p_m , $m > 1$, dann ist die Konvergenz linear mit dem Faktor $(\alpha_1^{(m)})^2$ [2]. Dies kommt der Lösung der Gleichungssysteme mit Digitalrechnern sehr entgegen. Denn da die Konvergenz insbesondere für die praktisch bedeutsamen Fälle, wo nicht $m \gg 1$ gilt, recht schnell ist, sind schon für relativ kleine $i \ll n$ die Rechnerapproximationen der einzelnen Grenzwerte erreicht, mit denen die Gleichungsaufösungen zum größten Teil durchgeführt werden können. Daneben werden nur noch die Elemente der Dreiecksmatrizen für sehr kleine und sehr große Zeilenindizes, $i \approx 1$ und $i \approx n$, benötigt. Es läßt sich zeigen, daß bei exakter Rechnung der Fehler in der Bestimmung der Lösungsvektoren von der Ordnung des maximalen Darstellungsfehlers der Dreiecksmatrizenelemente ist. Da aber darüberhinaus Rundungsfehler auftreten, ist eine a-posteriori-Fehlerabschätzung nach dem Banach'schen Fixpunktsatz und eine Verbesserung der Lösungsvektoren durch die

anschließende Ausführung eines oder mehrerer Schritte eines geeigneten Iterationsverfahrens möglich.

5.3 Vergleich mit herkömmlichen Verfahren zur Funktions- erfassung unter Rauscheinfluß

Zur Erfassung empirischer Funktionen werden in der Prozeßdatenverarbeitung bisher fast ausschließlich Augenblickswerte des anliegenden Signals $u(t)$ gemessen. Durch die Berechnung (gewichteter) Mittelwerte aus zu äquidistanten Zeitpunkten gemessenen Daten soll der Rauschanteil von u reduziert werden. Um diese Vorgehensweise mit integrierenden Verfahren zu vergleichen, wenden wir einen solchen Operator auf die Sinusschwingung $s(t) = \rho \cdot \sin(\omega t + \phi)$ an. Seien δ die Dirac-Distribution bzgl. des Ursprungs, $n \in \mathbb{N}$, a_0, \dots, a_n reelle Koeffizienten und Δ eine Zeitdifferenz, dann läßt sich der entsprechende Mittelungsoperator als Faltung mit dem Kern

$$\psi(t) := a_0 \cdot \delta(t) + \sum_{i=1}^n a_i \cdot (\delta(t-i \cdot \Delta) + \delta(t+i \cdot \Delta))$$

schreiben. Wegen der Geradheit von ψ und der Kosinusfunktion gilt

$$\begin{aligned} (\psi * s)(t) &= \rho \cdot \sin(\omega t + \phi) \cdot \int_{\mathbb{R}} \{ a_0 \cdot \delta(x) + \sum_{i=1}^n a_i \cdot (\delta(x-i \cdot \Delta) + \delta(x+i \cdot \Delta)) \} \times \\ &\quad \times \cos(\omega x) dx \\ &= s(t) \cdot \{ a_0 + 2 \cdot \sum_{i=1}^n a_i \cdot \cos(i \Delta \omega) \} =: s(t) \cdot p(\Delta \omega), \quad t \in \mathbb{R}, \end{aligned}$$

wobei p ein gerades trigonometrisches Polynom bezeichne. Auf Grund der 2π -Periodizität von p sind die Möglichkeiten der Rauschfilterung durch Berechnung gewichteter Mittelwerte erheblich eingeschränkt. Zwar können durch die Vorgabe von

maximal n Nullstellen für p im Intervall $[0, \pi]$ alle entsprechenden Störfrequenzen und ihre Vielfachen völlig unterdrückt werden, jedoch ist die Wahl nichtperiodischer Filtercharakteristiken wie z.B. die eines Tief-, Hoch- oder Bandpasses unmöglich. Mithin wird man durch Integration des Signals u bzgl. geeigneter Gewichtsfunktionen ϕ weniger von Rauschen verfälschte Daten gewinnen, die die Funktion f beschreiben. Dies werden wir weiter unten noch für den Fall $\phi=b_0$ verdeutlichen. Sollen unter Rauscheinfluß die Elemente eines m -dimensionalen Unterraums W durch Meßwerte bestimmt werden können, dann sind entweder m Integrale zu erfassen oder $m(2n+1)$ Augenblickswerte zu messen und weiterzuverarbeiten. Daraus ersehen wir, daß integrierende Verfahren neben besserer Rauschunterdrückung den Vorteil einer bedeutenden Datenreduktion bieten. Die Integration des anliegenden Signals u muß (weitgehend) analog erfolgen, da die meisten numerischen Integrationsverfahren die Form gewichteter Mittelungsoperatoren haben. Eingabeeinheiten, die diese Forderungen erfüllen, sollen im folgenden Paragraphen beschrieben werden.

Zum quantitativen Vergleich integrierender und auf gewichteten Mittelwerten beruhender Meßdatenerfassungsverfahren gehen wir von mit Rauschen überlagerten Polynomen 1. Grades als Testfunktionen aus. In dem in Satz 5.3 eingeführten Verfahren und der dazu analogen Abbildung

$$P_{m,h}: \begin{cases} C(\mathbb{R}) \rightarrow \Pi_m\{\pi_h\} \cap C^{m-1}(\mathbb{R}) \\ f \rightarrow \sum_{i \in \mathbb{Z}} f(i \cdot h) \cdot b_m\left(\frac{\cdot - i}{h}\right) \end{cases}, \quad m \geq 2, h > 0,$$

haben wir zwei positive, lineare Operatoren der Norm 1 zur Verfügung, die bei der Vorgabe exakter Daten Geraden reproduzieren. Wie weit die aus den Meßwerten berechneten Funktionen von den Testpolynomen abweichen, hängt nun nur noch vom Rauschen r und den Eigenschaften der verwendeten Datenerfassungsverfahren ab. Nach Voraussetzung können wir das Rauschsignal $r(t)$ in der Form $\int_{-\omega_g}^{\omega_g} s(\omega, t) d\omega$ mit $s(\omega, t) = \rho(\omega) \cdot \sin(\omega t + \phi(\omega))$ und der Grenzfrequenz $\omega_g > 0$ schreiben, wobei für jedes reale physikalische System $\omega_g < \infty$ gilt. Für beide betrachtete Verfahren schätzen wir jetzt den Rauschanteil in den Meßwerten ab, indem wir die Konvolution von r mit dem jeweiligen Faltungskern zur Zeit $t = \tau$ bilden, die Reihenfolge der Integrationen vertauschen und dann die Schwarz'sche Ungleichung anwenden.

1. Lokale Integration:

$$|(\Phi * r)(\tau)| = |(\Phi * \int_{-\omega_g}^{\omega_g} s(\omega, \cdot) d\omega)(\tau)| = |\int_{-\omega_g}^{\omega_g} (\Phi * s(\omega, \cdot))(\tau) d\omega| \leq$$

$$\int_{-\omega_g}^{\omega_g} |s(\omega, \tau) \cdot \frac{\sin(\frac{\pi \omega}{\Delta})}{(\frac{\pi \omega}{\Delta})}| d\omega \leq$$

$$\{\int_{-\omega_g}^{\omega_g} \rho^2(\omega) \cdot \sin^2(\omega \tau + \phi(\omega)) d\omega\}^{1/2} \cdot \{\int_{-\omega_g}^{\omega_g} (\frac{\sin(\frac{\pi \omega}{\Delta})}{\frac{\pi \omega}{\Delta}})^2 d\omega\}^{1/2} \leq$$

$$\{\int_{-\omega_g}^{\omega_g} \rho^2(\omega) d\omega\}^{1/2} \cdot \{\int_{\mathbb{R}} (\frac{\sin(\frac{\pi \omega}{\Delta})}{\frac{\pi \omega}{\Delta}})^2 d\omega\}^{1/2} = \{\int_{-\omega_g}^{\omega_g} \rho^2(\omega) d\omega\}^{1/2} \cdot \sqrt{\Delta}$$

Wegen $\Delta = \frac{2\pi}{l}$ nimmt $(\Phi * r)(\tau)$ mit wachsender Länge l des Integrationsintervalls ab.

2. Gewichtete Mittelwertbildung:

$$|(\psi * r)(\tau)| = |(\psi * \int_{-\omega_g}^{\omega_g} s(\omega, \cdot) d\omega)(\tau)| = |\int_{-\omega_g}^{\omega_g} (\psi * s(\omega, \cdot))(\tau) d\omega| \leq$$

$$\int_{-\omega_g}^{\omega_g} |s(\omega, \tau) \cdot p(\Delta\omega)| d\omega \leq$$

$$\left\{ \int_{-\omega_g}^{\omega_g} p^2(\omega) \cdot \sin^2(\omega\tau + \phi(\omega)) d\omega \right\}^{1/2} \cdot \left\{ \int_{-\omega_g}^{\omega_g} p^2(\Delta\omega) d\omega \right\}^{1/2} \leq$$

$$\left\{ \int_{-\omega_g}^{\omega_g} p^2(\omega) d\omega \right\}^{1/2} \cdot \left\{ 2a_0^2 + 4 \cdot \sum_{i=1}^n a_i^2 \right\}^{1/2} \cdot \sqrt{\omega_g}$$

wenn wir o.B.d.A. annehmen, daß $\frac{\Delta\omega}{\pi} g \in \mathbb{N}$ ist. Die für die

Meßdatenerfassung unerläßliche Voraussetzung $\psi * c = c$ für

konstante Funktionen c ist äquivalent mit $a_0 + 2 \cdot \sum_{i=1}^n a_i = 1$.

Eine kurze Rechnung zeigt, daß unter dieser Voraussetzung

der Ausdruck $\left\{ 2a_0^2 + 4 \cdot \sum_{i=1}^n a_i^2 \right\}^{1/2}$ für $a_i = \frac{1}{2n+1}$, $i=0, \dots, n$,

minimal wird. Im günstigsten Fall gilt also

$$|(\psi * r)(\tau)| \leq \left\{ \int_{-\omega_g}^{\omega_g} p^2(\omega) d\omega \right\}^{1/2} \cdot \left\{ \frac{2}{2n+1} \right\}^{1/2} \cdot \sqrt{\omega_g}.$$

Diese Abschätzung zeigt, daß wegen des Auftretens des

Terms $\sqrt{\omega_g}$ die Rauschunterdrückung bei diesem Verfahren

nicht vom physikalischen Experiment unabhängig ist. Eine

Verbesserung der Rauschreduktion ist nur durch die Ver-

größerung der Meßwertanzahl $2n+1$ möglich.

Für den Komplexitätsvergleich beider Methoden legen wir ein durch die Grenzfrequenz ω_g charakterisiertes physikalisches System und das Mittelwertverfahren bzgl. gleicher Gewichte zugrunde, da dieses die günstigste Fehlerabschätzung liefert und nur eine Multiplikation für jeden zu berechnenden B-Spline-Koeffizienten erfordert. Weiterhin nehmen wir an, daß die Bestimmung lokaler Integrale und Augenblicksmessungen in etwa gleich aufwendig sind, denn letztere werden

häufig durch Integrationen über sehr kurze Zeiträume realisiert. Schließlich geben wir eine gemeinsame obere Schranke S für den Rauscheinfluß $|\phi * r|$ und $|\psi * r|$ vor, mit der wir wegen der Eigenschaften der beiden Spline-Operatoren die Normen der Bildfunktionen von r ebenfalls abschätzen können:

$$\|L_{m,h}r\|_{\infty} \leq S \quad \text{und} \quad \|P_{m,h}r\|_{\infty} \leq S.$$

Aus S läßt sich die Länge l des Integrationsintervalls mit der entsprechenden Rauschunterdrückung berechnen. Durch Gleichsetzen der beiden oben hergeleiteten Ausdrücke für $|\phi * r|$ bzw. $|\psi * r|$ erhalten wir eine Formel für die Anzahl der zu mittelnden Meßwerte: $2n+1 = \frac{1}{\pi} \cdot \omega_g$. Mithin erfordert die Bestimmung eines B-Spline-Koeffizienten beim Integrationsverfahren eine Messung und eine Übertragung des Meßwertes zur verarbeitenden Einheit sowie eine Multiplikation. Dem stehen beim günstigsten Mittelungsoperator $\frac{1}{\pi} \cdot \omega_g$ Messungen und Übertragungen, $\frac{1}{\pi} \cdot \omega_g - 1$ Additionen und eine Multiplikation gegenüber. Mehr Zeit als die Rechenoperationen benötigt jedoch i.a. die Bearbeitung der Treiber- und Unterbrechungsfunktionen. Auch im Hinblick auf diese Verwaltungsaufgaben ist die Komplexität der Mittelwertbildung um den Faktor $\frac{1}{\pi} \cdot \omega_g$ höher als die der direkten Integration.

§ 6 Module zur integrierenden Erfassung und Ausgabe

kontinuierlicher Zeitfunktionen

Auf Grund der Ergebnisse des im letzten Paragraphen durchgeführten Vergleichs der Erfassung und Darstellung empirischer Zeitfunktionen mit Hilfe diskreter Funktionswerte bzw. lokaler Integrale sollen nun Eingabemodule beschrieben werden, die die notwendigen Integrationen fortlaufend selbstständig ausführen und die erhaltenen Funktionsparameter dann zur weiteren Verarbeitung an andere Einheiten übertragen. Wie wir oben gesehen haben, müssen die eigentlichen Integrationen analog erfolgen. Zur Überführung der so erhaltenen Spannungen in digitale Werte werden wir Analog-Digital-Wandlungsverfahren auf der Basis von Zwischengrößen anwenden, nämlich Modifikationen der Spannungsfrequenz-Wandlungs- und der Zwei-Rampen-Methode, die wesentlich genauer als die anderen bekannten Wandlungsmethoden sind und zu einem großen Teil digitale Komponenten verwenden. Der Nachteil dieser Verfahren, nämlich der relativ große Zeitbedarf für eine Umsetzung, wirkt sich hier nicht aus, da durch die intrinsische Störunterdrückung der Integration, die auf der Auswertung der ganzen verfügbaren Information beruht, wesentlich weniger Daten und damit weniger Wandlungen als bei der Verwendung von Funktionswerten erforderlich sind. Wir werden einen Hilfssatz beweisen, nachdem die bei der L^2 -Approximation erforderlichen mit B-Splines gewichteten Integrationen der experimentellen

Funktionen auf Linearkombinationen lokal gebildeter iterierter Integrale letzterer zurückgeführt werden können, so daß auf analoge Multiplikationen verzichtet werden kann. Bisher war es aus Kostengründen üblich, mehrere analoge Kanäle über einen Multiplexer an einen zentralen Analog-Digital-Wandler anzuschließen, der damit zu einem Flaschenhals wurde. Dagegen werden die Vorteile der integrierenden Datenerfassung durch den parallelen Einsatz mehrerer Integratoren erkaufte, deren Anzahl der der Kanäle proportional ist. Da jedoch die häufige Verwendung gleichartiger Hardwarekomponenten die Voraussetzung zu ihrer Standardisierung in Form von Halbleiterbausteinen ist, die billig produziert werden können und deren Preise laufend sinken, wird sich auch die kontinuierliche Erfassung lokaler Integrale wirtschaftlich realisieren lassen. Wir beschließen diesen Paragraphen mit der Beschreibung der Funktionsweise eines auf die Ausgabe quadratischer Splines spezialisierten Moduls.

6.1 Integration mit Polynom-Splines als Gewichtsfunktionen

Im letzten Paragraphen haben wir das Rauschunterdrückungsverhalten linearer Funktionale der Form

$$L_x f := \int_{\mathbb{R}} f(t) \cdot \phi(t-x) dt \text{ untersucht, wobei die geraden Kerne } \phi$$

kompakte Träger haben. Da die Integration analog durchgeführt werden muß, müßte auch die Funktion ϕ analog erzeugt und mit f multipliziert werden. Die beiden letztgenannten Operationen sind jedoch relativ ungenau, so daß die direkte

Auswertung von $L_x f$ i.a. nicht angebracht ist. Wie die folgende Umformung zeigt, läßt sich aber die Berechnung von $L_x f$ für den Fall, daß ϕ eine Polynom-Splinefunktion ist, auf eine gewichtete Summe iterierter Integrale von f zurückführen.

Dazu betrachten wir die Partition $\pi: a=x_0 < x_1 < \dots < x_n < x_{n+1}=b$, $n \in \mathbb{N}$, des Intervalls $[a,b]$, die Funktion $f \in C[a,b]$ und eine Polynom-Splinefunktion m -ten Grades s , $m \in \mathbb{N}$, bzgl. π . Für alle $i \in \{0, \dots, n\}$ existiert dann genau ein Polynom m -ten Grades p_i , derart daß $s(x) = p_i(x)$ für $x \in [x_i, x_{i+1}]$ und $i=0, \dots, n$ gilt. Wir können p_i als Taylorpolynom mit dem Entwicklungspunkt x_{i+1} schreiben:

$$p_i(x) = \sum_{j=0}^m \frac{1}{j!} \cdot p_i^{(j)}(x_{i+1}) \cdot (x - x_{i+1})^j, \quad i=0, \dots, n.$$

Bezeichnen wir mit $(J_y^{j+1} f)(t) = \frac{1}{j!} \cdot \int_y^t (t-x)^j \cdot f(x) dx$, $j \in \mathbb{N}$,

das vom Punkt $y \in \mathbb{R}$ aus gebildete $(j+1)$ -te iterierte Integral der Funktion f , dann läßt sich nun

$\int_a^b f(x) \cdot s(x) dx$ wie folgt umformen:

$$\begin{aligned} \int_a^b f(x) \cdot s(x) dx &= \sum_{i=0}^n \int_{x_i}^{x_{i+1}} f(x) \cdot p_i(x) dx \\ &= \sum_{i=0}^n \sum_{j=0}^m \frac{1}{j!} \cdot p_i^{(j)}(x_{i+1}) \cdot (-1)^j \cdot \int_{x_i}^{x_{i+1}} (x_{i+1} - x)^j \cdot f(x) dx \\ &= \sum_{i=0}^n \sum_{j=0}^m (-1)^j \cdot p_i^{(j)}(x_{i+1}) \cdot (J_{x_i}^{j+1} f)(x_{i+1}). \end{aligned}$$

Die iterierten Integrale von f können leicht durch mehrfache analoge Integration mit verschwindenden Anfangswerten gebildet werden. Die übrigen in obiger Formel vorkommenden

Operationen, Multiplikationen und Additionen, lassen sich nach geeigneten Analog-Digital-Wandlungen digital durchführen. Wegen ihrer guten Rauschunterdrückungseigenschaften haben wir im letzten Paragraphen B-Splines als Faltungskerne betrachtet. Auf Grund der oben hergeleiteten Berechnungsformel zeichnen sie sich gegenüber nicht stückweise polynomialen Kernen dadurch aus, daß die entsprechenden Faltungsintegrale einfach und genau bestimmt werden können.

6.2 Module zur fortlaufenden Erfassung lokaler Integrale

Zur Umwandlung der analog gebildeten Integrale in digital dargestellte Werte werden wir Modifikationen zweier Verfahren anwenden, die sehr genau sind und einen großen dynamischen Bereich besitzen, und zwar des Spannungs-Frequenz-Umsetzungs- und des Zwei-Rampen-Verfahrens [24]. Die Modifizierungen dienen dazu, den im Falle des Spannungs-Frequenz-Umsetzers durch die Rückstellzeit verursachten Linearitätsfehler zu beseitigen und andererseits kontinuierliche Integrationen mit Zwei-Rampen-Umsetzern zu ermöglichen.

Im folgenden werden wir zwei Funktionseinheiten beschreiben, die jeweils eine Anzahl gemäß der beiden genannten Verfahren konstruierter Integratoren steuern und die resultierenden Daten weiterverarbeiten. Zur Vereinfachung synchronisieren wir Start und Stop der einzelnen Kanäle mit einem gemeinsamen Takt zur Markierung der Integrationsintervallgrenzen. Die Arbeit dieser Einheiten läßt sich mit drei Befehlen

steuern: Versetzen der Module in einen definierten Grundzustand und Start sowie Stop bestimmter Kanäle. Bei der Entgegennahme dieser Befehle werden die Parameter überprüft und unzulässige Eingaben zurückgewiesen. Die Module nehmen die Umwandlung der von den Integratoren eingelesenen Werte in weiterverarbeitbare physikalische Größen und eine Pufferung der Daten in Fifo-Speichern vor. Von dort werden die Ausgabedaten blockweise zu Datenspeichermodulen unter Ausnutzung ihrer automatischen Feldverwaltung übertragen. Ist der einem Kanal in einem DSM zugeordnete Puffer vor der Abschaltung dieses Kanals gefüllt, dann führt die Integrationseinheit letztere Operation durch und sendet der die Daten bearbeitenden Task eine entsprechende Nachricht.

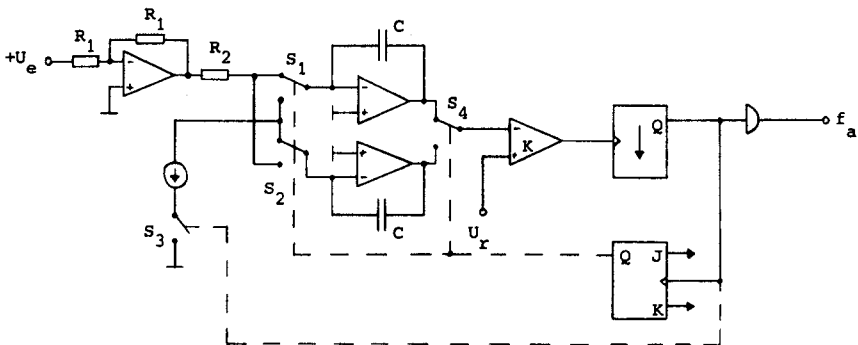
6.2.1 Integrationseinheit nach dem Spannungs-Frequenz-

Umsetzungsverfahren

Die Funktionseinheit, die wir jetzt vorstellen wollen, kann nur einfache lokale Integrale der Eingangsspannung U_e bilden, da ihre Arbeitsweise auf dem Spannungs-Frequenz-Wandlungsprinzip beruht. Dafür ist jedoch eine räumliche Trennung zwischen dem eigentlichen Umsetzer und dem nachgeschalteten Zähler möglich, die nur eine einzige Verbindungsleitung erfordert, auf der digitale Signale übertragen werden.

Das Modul setzt sich folgendermaßen zusammen: an eine zentrale Steuereinheit ist für jeden Kanal eine Zähl- und Speicherkomponente angeschlossen, die während der Integra-

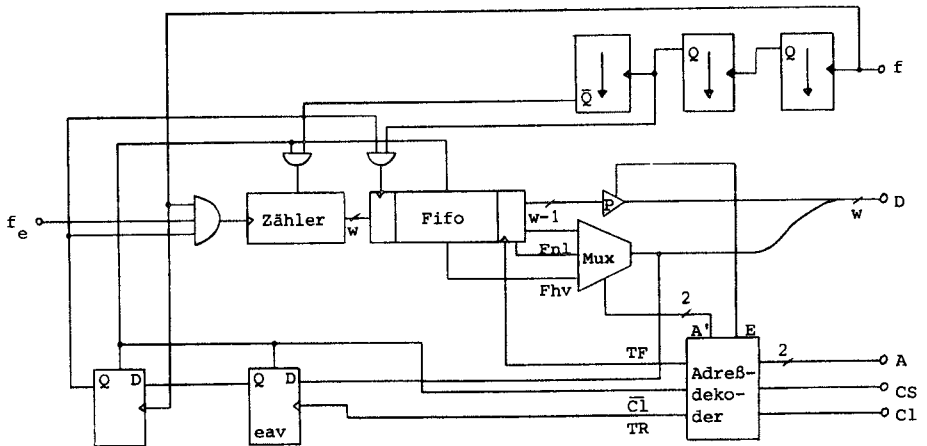
tionsintervalle die Impulse zählt, welche der ihr zugeordnete Spannungs-Frequenz-Umsetzer mit zur angelegten Eingangsspannung proportionalen Frequenz aussendet. Die akkumulierten Meßwerte werden am Ende der Integrationsintervalle in Fifos eingegeben, die sich in den Zähl- und Speicherkomponenten befinden und aus denen sie die Steuereinheit zur Offsetsubtraktion und zum Weitertransfer entnimmt, wenn die Fifos bis zu einem gewissen Grade gefüllt sind. Weiterhin verwaltet die letztgenannte Einheit die Zustände der einzelnen Kanäle. Im folgenden wollen wir nun die verschiedenen Komponenten des Moduls detailliert beschreiben.



Figur 8. Modifizierter Spannungs-Frequenz-Umsetzer

Der Spannungs-Frequenz-Wandler, dessen Funktionsdiagramm in Figur 8 dargestellt ist, invertiert zuerst die positive Eingangsspannung U_e , die dann über den Widerstand R_2 abwechselnd einen der beiden Kondensatoren C auflädt, d.h. integriert wird. Hat das gerade gebildete Integral einen

bestimmten Wert erreicht, dann liegt an dem soeben mit U_e verbundenen Kondensator eine Spannung von der Größe der Referenzspannung U_r an, weshalb der Komparator K schaltet, was einen Ausgangsimpuls des Monoflops zur Folge hat. Dieser stellt das Ausgangssignal dar und invertiert den Zustand des J-K-Flipflops, wodurch die Schalter S_1 , S_2 und S_4 umgeschaltet und der andere Integrator mit U_e verbunden werden. Während letzterer schon zu arbeiten begonnen hat, entlädt der Monoflop-Impuls noch mit Hilfe des Schalters S_3 den zuvor verwendeten Kondensator. Auf diese Weise wird der bei bisherigen Spannungs-Frequenz-Wandlern durch die Rückstellzeit des Integrationskondensators hervorgerufene Linearitätsfehler eliminiert. Die Ausgangsfrequenz f_a wird mit dem Eingang f_e des in Figur 9 skizzierten Zähl- und Speichermoduls verbunden.



Figur 9. Zähl- und Speichermodul

Tabelle 3. Funktion des im Zähl- und Speichermodul vorhandenen Adreßdekoders und Multiplexers

CS	C1	A	A'	E	TF	TR	$\overline{C1}$	Mux.-Ausgang	P-Ausgang
0	X	XX	00	0	0	0	1	-	-
1	1	XX	00	0	0	0	0	-	-
1	0	00	00	0	0	1	1	-	-
1	0	01	01	0	0	0	1	Fhv	-
1	0	10	10	0	0	0	1	Fnl	-
1	0	11	11	1	1	0	1	(Fifo) ₀	(Fifo) ₁ ... (Fifo) _{w-1}

An die Steuereinheit eines Funktionseingabemoduls seien $n \geq 1$ Zähl- und Speichereinheiten $zsm[i]$, $i \in \{1, \dots, n\}$, angeschlossen. Die durch Zählen der Eingangsimpulse bestimmten Integralwerte werden in den nachgeschalteten Fifos zwischengespeichert, die die Signale $fhv[i]$, falls $fifo[i]$ bis zu einem gewissen Grade gefüllt ist, und $fnl[i]$, wenn das Ausgangsregister $fifoaus[i]$ belegt ist, erzeugen. Über den $w \geq 1$ Bits breiten Bus D lassen sich gemäß Tabelle 3 diese Zustandssignale abfragen und die Fifos auslesen. Bei letzterer Operation wird durch Erzeugung des Signals TF das Register $fifoaus[i]$ gelöscht, womit die übrigen Daten nachrücken. Bei geeigneter Adressierung über den Bus A und die Wählleitung CS lassen sich mit C1 der Zähler, der Fifo und die internen Register von $zsm[i]$ löschen bzw. durch Erzeugung des Taktes TR das Flipflop $eav[i]$ über D laden. Die Ausgänge A' und E des Adreßdekoders steuern den Multiplexer und die Treiber P. Der negativ aktive Takt f markiert die

Integrationsintervalle, trennt die Eingangsfrequenz f_e kurzzeitig vom Zähler und triggert die Monoflop-Kette. Diese lädt den Inhalt des Zählers, nachdem letzterer zur Ruhe gekommen ist, in den Fifo und löscht anschließend den Zähler. Mit der positiven Flanke von f wird der Zustand von $eav[i]$ in das nachgeschaltete D-Flipflop übernommen, welches ebenfalls die Eingangsfrequenz maskiert. In dieser Weise wird der Zustand des Kanals i , der jederzeit durch Schreiben in das Register $eav[i]$ geändert werden kann, mit f synchronisiert.

Bevor wir die Funktionsweise der Steuereinheit durch die Angabe einer Initialisierungsroutine, eines durch den Takt f aufgerufenen Kanalstatusverwaltungsprogramms sowie einer Parametereingabe- und einer immer aktiven Datentransferprozedur beschreiben werden, müssen wir zuerst die Bedeutung der von diesen Programmen bearbeiteten Daten darlegen.

So werden mehrere eindimensionale Felder der Länge n benötigt: in "ein" wird der Zustand der Kanäle gespeichert und für eingeschaltete Kanäle werden in an f die Anfangszeitpunkte der Integrationen, in adr die logischen Adressen der jeweils in einem DSM angelegten Datenpuffer und in len deren Längen abgelegt. Nach normaler Beendigung der Eingaben bzgl. des Kanals i setzt die Steuereinheit das an den Task-Scheduler angeschlossene Ereignis mit der Nummer $er[i,1]$ und im Falle des Abbruchs der Integrationen auf Grund eines erschöpften Datenpuffers das Ereignis $er[i,2]$.

Die aktuelle Uhrzeit sei als `time` verfügbar und werde im Abstand Δ vom Takt f inkrementiert. Mit der Prozedur `get` werden solange Befehlsparameter eingelesen, bis die Zustandsvariable `eof` den Wert `true` annimmt. Dabei wird überprüft, ob die übergebenen Adressen und Ereignisnummern zu den zulässigen Mengen `adm` bzw. `erm` solcher Daten gehören und ob die eingelesenen Pufferlängen eine obere Grenze `lmax` nicht überschreiten. Vor der Übertragung zu den DSM's subtrahiert die Steuereinheit die Konstante `offset` von den in den Zählern gebildeten Werten und wandelt sie dann mittels der Prozedur `float` in Gleitkommadarstellung um. Die übrigen in den folgenden Routinen vorkommenden Namen bezeichnen Hilfsvariable.

Auf Grund eines `reset`-Signals werden das Eingabemodul in einen definierten Grundzustand versetzt, der Takt gestartet sowie das Einlesen von Befehlsparametern und die Ausgabe von Integralwerten ermöglicht:

on `reset` do

for `i` from 1 by 1 to `n` do clear `zsm[i]` , `ein[i]` := 0 od ,
 `eof` := `true` , start `f` ;

`startstop` ; `ausgabe`

od .

Die folgende Prozedur übernimmt die Parameter der an das Modul übergebenen Befehle, überprüft sie und leitet gegebenenfalls Kanalein- bzw. ausschaltungen ein. Ihre Komplexität ist der Anzahl der anstehenden Befehle proportional.

proc startstop=:

while ~eof do

get(u,i) ;

if i>1^i<n then

if u then

if ein[i]=6 then ein[i]:=5 , eav[i]:=false fi

else

get(ad,1,j,k) ;

if adeadm^1>2^1<lmax^j^erm^k^erm^ein[i]=0 then

ein[i]:=6 , eav[i]:=true , adr[i]:=ad , len[i]:=1+2 ,

er[i,1]:=j , er[i,2]:=k , anf[i]:=time+Δ

fi

fi

fi

od .

Die zentrale Routine der Steuereinheit ist die Prozedur ausgabe, die nach dem reset-Signal gestartet wird und dann aktiv bleibt. Sie überprüft fortlaufend die Füllungsgrade der einzelnen Fifos. Sind letztere mindestens zur Hälfte gefüllt oder sollen Kanäle ausgeschaltet werden, dann werden die anstehenden Meßwerte aus den Fifos ausgelesen, verarbeitet und zu den DSM's übertragen. Die Prozedur führt auch die bei der Beendigung von Funktionseingaben erforderlichen Operationen durch. Abgesehen von der immerwährenden Kontrolle der Fifoinhalte ist ihre Komplexität den Anzahlen der einzulesenden Daten und der zu erfassenden Funktionen proportional.

```
proc ausgabe:=  
  a:for i from 1 by 1 to n do  
    if fhv[i]vein[i]=1vein[i]=3 then  
      while fnl[i] do  
        cont adr[i]:=float(fifoaus[i]-offset)  
      od ;  
    if ein[i]=1vein[i]=3 then  
      cont adr[i]:=anf[i] ,  
      cont er[i,if ein[i]=3 then 1 else 2 fi]:=true ;  
      clear zsm[i] , ein[i]:=0  
    fi  
  fi  
od ; goto a] .
```

Mit der positiven Flanke von f, d.h. nachdem die letzten Meßwerte in die Fifos übernommen und die Zähler gelöscht worden sind, wird wieder das Einlesen von Befehlsparametern durch Aufruf von startstop ermöglicht, für eingeschaltete Kanäle geprüft, ob sie wegen einer bevorstehenden Erschöpfung der zugeordneten Puffer abgeschaltet werden müssen, und werden schließlich Statustransfers durchgeführt. Die Komplexität der nun folgenden, mit jedem Taktimpuls aktivierten Routine ist proportional zur Anzahl der angeschlossenen Eingabekanäle n.

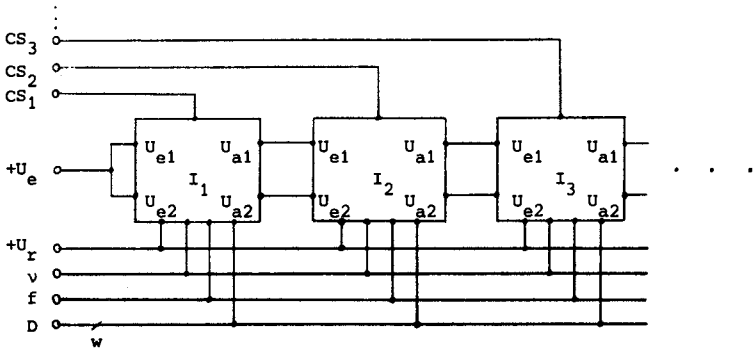
```
on f=+ do  
  startstop ;  
  for i from 1 by 1 to n do  
    if ein[i]=6 then  
      len[i]:=len[i]-1 ;
```

```
if len[i]=2 then ein[i]:=2 , eav[i]:=false fi  
      else  
      if ein[i]=2vein[i]=4vein[i]=5 then ein[i]:=ein[i]-1 fi  
    fi  
  od  
od .
```

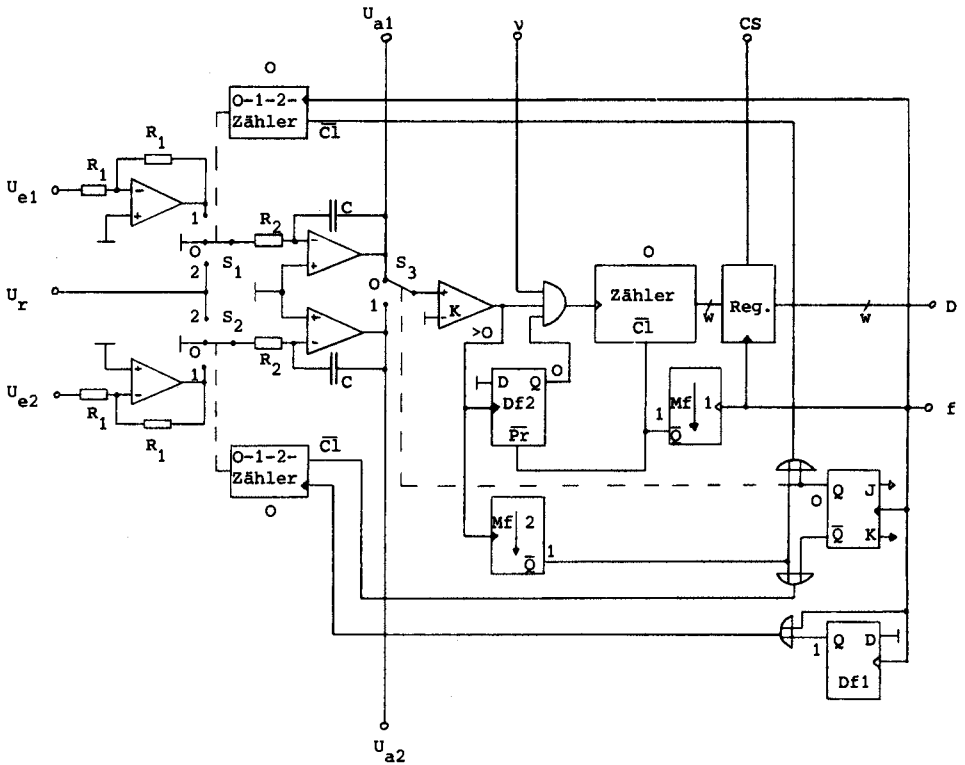
6.2.2 Integrationseinheit nach dem Zwei-Rampen-Verfahren

In diesem Abschnitt werden wir ein Eingabemodul behandeln, das der fortlaufenden Erfassung von mit B-Splines m -ten Grades, $m \in \mathbb{N}$, als Gewichtsfunktionen gebildeten Integralen einer extern anliegenden zeitlich veränderlichen Spannung U_e dient. Dies läßt sich auf Grund des oben hergeleiteten Hilfssatzes mit jeweils $m+1$ gleichartigen Integratoren pro Kanal erreichen. Als Integrierelemente setzen wir dabei Komponenten ein, die nach dem Zwei-Rampen-Verfahren arbeiten, jedoch derart modifiziert sind, daß eine kontinuierliche Arbeitsweise ermöglicht wird.

Der Aufbau des Moduls läßt sich wie folgt umreißen: an eine zentrale Steuereinheit sind für jeden Kanal $m+1$ Integrierelemente angeschlossen, die gemäß Figur 10 zusammengeschaltet seien und die während der Integrationsintervalle analog die ersten $m+1$ iterierten Integrale der Eingangsspannungen bilden. Gleichzeitig werden die in den jeweils vorhergehenden Intervallen angefallenen Analogwerte digitalisiert. Diese Daten werden am Ende der Integrationsintervalle von



Figur 10. Zusammenschaltung von Integrierelementen zur Bildung iterierter Integrale



Figur 11. Modifizierter Zwei-Rampen-Umsetzer

der Steuereinheit eingelesen und gemäß des Hilfssatzes zusammen mit früheren Werten zu den gewünschten Faltungsintegralen weiterverarbeitet. Letztere werden schließlich in Fifos zwischengespeichert, von wo aus sie die Steuereinheit blockweise an DSM's weitertransferiert. Ansonsten entsprechen die Funktionen dieser Einheit den im letzten Abschnitt genannten.

Zur Beschreibung des in Figur 11 dargestellten Integrierelementes setzen wir voraus, daß Eingangs- und Referenzspannung a priori angelegt seien und daß sich die Schalter S_1 und S_2 beim Anlegen der Betriebsspannung(en) in der Stellung 0 befinden, damit die Integrationskondensatoren C am Anfang entladen seien. Zu diesem Zeitpunkt mögen die internen Flipflops und Monoflops die in Figur 11 angegebenen Zustände haben. Der erste Impuls der Taktfrequenz f leitet die Arbeit der Integratoren ein, indem der Schalter S_1 in die Stellung 1 gesetzt wird, während S_2 in Position 0 verbleibt. Die invertierte Eingangsspannung lädt nun während des folgenden Intervalls den Kondensator C auf, dessen Spannung U_a an den Eingang des nachgeschalteten Elementes zur Bildung des nächst höheren iterierten Integrals gelegt wird. Zur gleichen Zeit führt der Ausgang des Komparators K eine logische 0, da sich der Schalter S_3 in der Stellung 1 befindet. Der zweite Taktimpuls schaltet das J-K-Flipflop und damit S_3 wieder um und inkrementiert beide 0-1-2-Zähler, weil der Ausgang von Df1 mittlerweile eine 0 führt. Nun wird das Eingangssignal auf dem anderen Kondensator integriert,

während der vorher benutzte durch die konstante Referenzspannung U_r entladen wird. Da f durch Triggern von $Mf1$ das Flipflop $Df2$ gesetzt hat, läßt sich die dem Integralwert proportionale Entladezeit durch Zählen der Digitalisierungsfrequenz $v \gg f$ messen. Ist der Kondensator entladen, dann fällt der Komparatorausgang auf 0 zurück, wodurch $Df2$ gelöscht und $Mf2$ getriggert werden. Der erzeugte Impuls versetzt, da der Q-Ausgang des J-K-Flipflops eine 0 führt, den Schalter S_1 durch Löschen des zugehörigen Zählers in den neutralen Zustand 0 zurück. Mit der negativen Flanke des dritten Impulses von f wird der Zählerinhalt, d.h. das Integral bzgl. des ersten Intervalls, in das Register übernommen, von wo er von der Steuereinheit durch Aktivierung der Wählleitung CS über den Bus D ausgelesen werden kann. Weiterhin schaltet diese negative Flanke S_1 , S_2 und S_3 um, wodurch sich der oben beschriebene Vorgang nach jedem Taktimpuls mit vertauschten Rollen wiederholt. Die positiven Flanken von f erzeugen mit Hilfe von $Mf1$ Impulse, die den Zähler löschen und $Df2$ setzen. Da die Schalter S_1 , S_2 und S_3 der zu einem Kanal gehörenden Integratoren von f parallel geschaltet werden, werden in jedem Intervall zwischen den Taktimpulsen auf einer Kette von Kondensatoren die iterierten Integrale von U_e gebildet, während die entsprechenden Werte des vorhergehenden Intervalls in der gleichen Zeit digitalisiert werden.

Wir wollen die Funktionen der Steuereinheit wieder durch eine Reihe von Programmen beschreiben. Werden im folgenden

die gleichen Bezeichnungen wie im Abschnitt 6.2.1 gebraucht, so sollen sich auch die Bedeutungen entsprechen.

Die Steuereinheit liest nach jedem Taktimpuls beginnend mit dem dritten die Register der Integratoren für alle eingeschalteten Kanäle aus, die wir als Feld $\text{reg}[1:n,0:m]$ zusammenfassen wollen. Zur Weiterverarbeitung dieser Daten gemäß unseres Hilfssatzes werden die Konstanten $(-1)^j \cdot b_m^{(j)} \left(-\frac{m-1}{2} + i - 0 \right)$ als $b[i,j]$, $i, j \in \{0, \dots, m\}$, bereitgehalten und $w[0:m]$ sowie die mittels des Indexfeldes $\text{ind}[1:n]$ verwalteten Ringpuffer $r[1:n,0:m]$ als Zwischenspeicher benötigt. Die fertig berechneten Faltungsintegrale werden schließlich über die Eingangsregister $\text{fifoein}[1:n]$ in die Fifospeicher $\text{fifo}[1:n]$ eingegeben, die sich in der Steuereinheit befinden. Die logische Variable enable dient zur Maskierung des Taktes f .

Die im folgenden angegebenen Prozeduren startstop und ausgabe sind den oben aufgeführten Routinen gleichen Namens sehr ähnlich und führen die gleichen Aufgaben durch. Das durch ein reset -Signal gestartete Initialisierungsprogramm versetzt nicht nur den ganzen spezialisierten Prozessor in einen definierten Grundzustand, sondern sorgt auch dafür, daß die Einlese- und Verarbeitungsroutine erstmals mit dem dritten Impuls von f aufgerufen wird:

on reset do

eof:=true , enable:=false ,

for i from 1 by 1 to n do

clear fifo[i] , ein[i]:=0 , ind[i]:=m ,

for j from 0 by 1 to m do r[i,j]:=0 od

```

od ;
start v,f , i:=0 ;
α:if f=↑ then i:=i+1 ; if i<2 then goto α fi fi ;
    enable:=true ; startstop ; ausgabe
od .
proc startstop=:
[while ~eof do
    get(u,i) ;
    if i>1^i<n then
        if u then if ein[i]=4 then ein[i]:=3 fi
            else get(ad,l,j,k) ;
                if ad&adm^l>m+1^l<lmax^j&erm^k&erm^ein[i]=0 then
                    ein[i]:=4 , adr[i]:=ad , len[i]:=1 ,
                    er[i,1]:=j , er[i,2]:=k , anf[i]:=time-Δ
                fi
            fi
        fi
    od ] .
proc ausgabe=:
[β:for i from 1 by 1 to n do
    if fhv[i]^vein[i]=1^vein[i]=2 then
        while fnl[i] do cont adr[i]:=fifoaus[i] od ;
        if ein[i]<3 then
            cont er[i,if ein[i]=2 then 1 else 2 fi]:=true ;
            clear fifo[i] , ein[i]:=0
        fi
    fi
od ; goto β ] .

```

Vom dritten Taktimpuls an liest die Steuereinheit nach jeder positiven Flanke von f die Integratorregister aus, berechnet aus diesen Daten die gewünschten Faltungsintegrale und speichert letztere in den Fifos zwischen. Weiterhin werden gegebenenfalls Statustransfers durchgeführt und Kanäle abgeschaltet sowie jeweils durch Aufruf von startstop das Einlesen neuer Befehlsparameter ermöglicht. Die Komplexität der nun folgenden Routine ist von der Ordnung $O(n \cdot (m+1)^2)$.

```
on ~f!enable=+ do
  for i from 1 by 1 to n do
    if ein[i]>2 then
      for j from 0 by 1 to m do w[j]:=float(reg[i,j]) od ;
      j:=mod(ind[i]+1,m+1) ;
      for k from m by -1 to 0 do
        for l from 0 by 1 to m do r[i,j]:=r[i,j]+b[k,l]*w[l] od ;
        j:=mod(j+1,m+1)
      od ;
      ind[i]:=j , fifoein[i]:=r[i,j] , len[i]:=len[i]-1 ; r[i,j]:=0 ;
      if ein[i]=3*len[i]=m+1 then
        for k from 1 by 1 to m do
          j:=mod(j+1,m+1) ; fifoein[i]:=r[i,j] ; r[i,j]:=0
        od ;
        fifoein[i]:=anf[i] , ind[i]:=m ;
        ein[i]:=if ein[i]=3 then 2 else 1 fi
      fi
    fi
  od ; startstop
od .
```

6.3 Modul zur Ausgabe quadratischer Splines

Abschließend soll nun noch ein spezialisierter Prozessor behandelt werden, der selbständig die Ausgabe stetiger Zeitfunktionen durchführt. Wir gehen davon aus, daß die auszugebenden Funktionen nach geeigneten Verfahren durch Polynom-Splines bzgl. äquidistanter Knoten hinreichend genau approximiert seien. Da das Wesentliche der Steuerungs- und Rechenverfahren bereits im Falle quadratischer Splines deutlich wird, werden wir uns hier auf solche beschränken. Weiterhin nehmen wir zur Vereinfachung an, daß alle auszugebenden Splines bzgl. von Teilmengen der Partition $\{i \cdot h \mid i \in \mathbb{N}, h > 0\}$ von \mathbb{R}_+ definiert seien. Diese Funktionen s seien jeweils durch ein in einem DSM bereitgehaltenes Feld a von B-Spline-Koeffizienten dargestellt. Die Aufgabe des Prozessors besteht nun darin, diese Daten zu übernehmen und aus ihnen Funktionswerte zu berechnen, die im zeitlichen Abstand $\Delta t := \frac{h}{m}$, $m > 1$, über Digital-Analog-Wandler auszugeben sind.

Um die Anzahl der dazu erforderlichen Rechenoperationen auf ein Minimum zu reduzieren, führen wir die lineare Variablentransformation $t := \frac{h}{m} \cdot \tau$, $\tau \in \mathbb{R}_+$, durch, die die Eigenschaft von s ein quadratischer Spline zu sein unverändert läßt und betrachten s im τ -Intervall $[i \cdot m, (i+1) \cdot m]$, $i \in \mathbb{N}$:

$$s(\tau) = a_i \cdot \frac{1}{2} \cdot \left(1 - \frac{\tau - i \cdot m}{m}\right)^2 + a_{i+1} \cdot \left(\frac{1}{2} + \frac{\tau - i \cdot m}{m}\right) \cdot \left(\frac{\tau - i \cdot m}{m}\right)^2 + a_{i+2} \cdot \frac{1}{2} \cdot \left(\frac{\tau - i \cdot m}{m}\right)^2$$

$$= \frac{1}{2} \cdot (a_i + a_{i+1}) + \frac{1}{m} \cdot (a_{i+1} - a_i) \cdot (\tau - i \cdot m) + \frac{1}{2m^2} \cdot (a_i - 2a_{i+1} + a_{i+2}) \times$$

$$\times (\tau - i \cdot m)^2,$$

$$s'(\tau) = \frac{1}{m} \cdot (a_{i+1} - a_i) + \frac{2}{2m^2} \cdot (a_i - 2a_{i+1} + a_{i+2}) \cdot (\tau - i \cdot m).$$

Wegen

$$s'(\tau+1) = s'(\tau) + 2 \cdot \frac{1}{2m^2} \cdot (a_i - 2a_{i+1} + a_{i+2})$$

und

$$\begin{aligned} s(\tau+1) &= \frac{1}{2} \cdot (a_i + a_{i+1}) + \frac{1}{m} \cdot (a_{i+1} - a_i) \cdot (\tau - i \cdot m) + \frac{1}{2m^2} \cdot (a_i - 2a_{i+1} + a_{i+2}) \times \\ &\quad \times (\tau - i \cdot m)^2 + \frac{1}{m} \cdot (a_{i+1} - a_i) + \frac{2}{2m^2} \cdot (a_i - 2a_{i+1} + a_{i+2}) \cdot (\tau - i \cdot m) + \\ &\quad + \frac{1}{2m^2} \cdot (a_i - 2a_{i+1} + a_{i+2}) \\ &= s(\tau) + s'(\tau) + \frac{1}{2m^2} \cdot (a_i - 2a_{i+1} + a_{i+2}) \end{aligned}$$

lassen sich bei Kenntnis von $s(i \cdot m)$ und $s'(i \cdot m)$ und nach Bestimmung der Konstanten $\frac{1}{2m^2} \cdot (a_i - 2a_{i+1} + a_{i+2})$ die weiteren Werte $s(i \cdot m + j)$, $j=1, \dots, m$, nur mit Hilfe von Additionen rekursiv berechnen. Nach jeweils m Ausgaben ist auf Grund der stetigen Differenzierbarkeit quadratischer Splinefunktionen nur eine neue Konstante erforderlich.

Die im folgenden dargestellten Algorithmen, welche zur exakten Angabe der Funktionsweise des Moduls dienen, sprechen wieder eine Reihe von Variablen mit bereits aus Abschnitt 6.2.1 bekannten Namen an, die wir in den dort eingeführten Bedeutungen verwenden wollen.

Wie oben werden zum Start eines Kanals von der Einheit Parameter eingelesen, geprüft und zwischengespeichert. Daraufhin werden die entsprechenden Elemente der beiden Felder $v[1:n]$ und $w[1:n]$ initialisiert, die die aktuellen Funktions- und Ableitungswerte enthalten. In den mit dem

Hilfsfeld `ind[1:n]` verwalteten Ringpuffern `r[1:n,0:2]` werden die jeweils gerade benötigten B-Spline-Koeffizienten gespeichert. Um Verzögerungen bei der blockweisen Übertragung letzterer aus den DSM's auszugleichen, ist wieder für jeden Kanal ein Fifo vorgesehen. Von diesen werden die Signale `fhl[1:n]`, sobald sie zu einem gewissen Grade entleert sind, sowie `fnl[1:n]` und `fnn[1:n]` erzeugt, falls sie noch gültige Daten enthalten bzw. noch weitere Werte aufnehmen können. In die Fifos werden die Konstanten $\frac{1}{2m} \cdot (a_i - 2a_{i+1} + a_{i+2})$, $i > 0$, eingegeben, und zwar die erste von der Startroutine und die übrigen von der Prozedur `eingabe`, die nach dem Auslesen zur rekursiven Funktionswertberechnung im Feld `u[1:n]` abgelegt werden. Bei Erscheinen der Impulse des positiv aktiven Taktes `v` werden die im Feld `v` anstehenden Daten durch Multiplikation mit der Konstanten `z` skaliert, mittels der Prozedur `fix` in Festkommadarstellung umgewandelt und schließlich in die Register `dac[1:n]` der Digital-Analog-Wandler geschrieben. Mit den negativen Flanken von `v` wird die Berechnung der beim jeweils nächsten Takt benötigten Funktionswerte begonnen. Vorher werden nach jedem `m`-ten Impuls den Fifos neue Daten entnommen bzw. im Falle der Erschöpfung der Vorräte die entsprechenden Kanäle abgeschaltet und diese dem Task-Scheduler angezeigt.

Auf Grund eines `reset`-Signals stellt das folgende Programmstück einen definierten Anfangszustand her:

```
on reset do  
  t:=m-1 , eof:=true ,  
  for i from 1 by 1 to n do  
    dac[i]:=0 , ein[i]:=0 , clear fifo[i] od ;  
  start v ; eingabe  
od .
```

Die beiden folgenden Routinen werden durch den Takt v aufgerufen. Sie nehmen die Analogdatenausgabe bzw. die rekursive Funktionswertberechnung und nach jeweils m Takten das Nachladen von Spline-Parametern aus den Fifos sowie eventuell das Ausschalten von Kanälen vor. Die Komplexität der Programme ist der Anzahl n der Ausgabekanäle proportional.

```
on v=+ do  
  t:=mod(t+1,m) ; s:=sign(t) ;  
  for i from 1 by 1 to n do  
    if ein[i]>s then dac[i]:=fix(z*v[i]) fi  
  od  
od  
und  
on v=+ do  
  if t=0 then  
    for i from 1 by 1 to n do  
      if ein[i]>0 then  
        if ein[i]=1 then ein[i]:=2 fi ;  
        if fnl[i] then u[i]:=fifoaus[i]  
          else dac[i]:=0 , ein[i]:=0 , cont er[i]:=true  
        fi  
      fi  
    fi
```

```

od
fi ;
for i from 1 by 1 to n do
  if ein[i]=2 then
    w[i]:=w[i]+u[i] ; v[i]:=v[i]+w[i] ; w[i]:=w[i]+u[i]
  fi
od
od .

```

Das Einlesen und Vorverarbeiten von Parametern und B-Spline-Koeffizienten führen schließlich die beiden letzten Programme durch. Die Komplexität dieser Routinen ist der Anzahl auszugebender Funktionen bzw. der Anzahl aller zu übertragender Spline-Parameter proportional, wenn man von der fortlaufenden Überprüfung der Fifoinhalte absieht.

```

on eof=+ do
  while ¬eof do
    get(i,ad,l,j) ;
    if i>1^i<n^ad<adm^l>3^l<lm^aj^erm^ein[i]=0 then
      adr[i]:=ad , len[i]:=l-3 , er[i]:=j ;
      r[i,0]:=cont adr[i] ; r[i,1]:=cont adr[i] ; r[i,2]:=cont adr[i] ;
      v[i]:=0.5×(r[i,0]+r[i,1]) , w[i]:=(r[i,1]-r[i,0])/m ,
      fifoein[i]:=(r[i,0]-2×r[i,1]+r[i,2])/(2×m×m) ,
      ind[i]:=2 , ein[i]:=1
    fi
  od
od
sowie

```

proc eingabe=:

```

a: for i from 1 by 1 to n do
    if ein[i] > 0 and fl[i] then
        k := ind[i] ;
        for j from len[i] by -1 while fnv[i] ^ j > 0 do
            k := mod(k+1, 3) ; r[i, k] := cont adr[i] ;
            fifoein[i] := (r[i, mod(k+1, 3)] - 2 * r[i, mod(k+2, 3)] +
                           r[i, k]) / (2 * m * m)
        od ;
        ind[i] := k , len[i] := j
    fi
od ; goto a .

```

Um die Komplexität der Funktionsausgabe mit Hilfe des hier vorgestellten spezialisierten Prozessors und eines konventionellen Prozeßrechners zu vergleichen, legen wir natürlich das gleiche, hier angewandte mathematische Verfahren zugrunde. Deshalb fallen bei beiden Strukturen die gleiche Anzahl von Rechenoperationen und in etwa gleich viele Instruktionen für die Kanalverwaltung an. Der wesentliche Komplexitätsunterschied besteht in der Zahl der Übertragungen. Soll nämlich eine Funktion über ein Zeitintervall der Länge $i \cdot h$, $i \in \mathbb{N}$, ausgegeben werden, so müssen $i+2$ B-Spline-Koeffizienten von einem DSM zur Ausgabeeinheit transferiert werden. Dagegen sind zum Setzen eines Digital-Analog-Wandlers im zeitlichen Abstand $\Delta t = \frac{h}{m}$ bei der konventionellen Struktur $i \cdot m$ Übertragungen aus einem Register der CPU erforderlich. Die Routine, welche diese Aufgabe durchführt, wird in der

Regel durch mit der gleichen Frequenz eintreffende Unterbrechungen aktiviert. Deshalb kommen in diesem Falle noch 2-fach in der Übertragungskomplexität weitaus umfangreichere Context-Switching-Operationen hinzu.

Anhang zu § 5

Beweis von Satz 5.1:

(i,iv) Die Existenz und Eindeutigkeit sowie die Minimal-eigenschaft wurden in [10, Sätze 7.6 und 7.7] gezeigt.

(ii) Unter Beachtung von $\text{supp } b_{2m} = [-m - \frac{1}{2}, m + \frac{1}{2}]$ folgt das angegebene Gleichungssystem unmittelbar aus den Interpolationsbedingungen.

(iii) In [10, Satz 7.6] wurde gezeigt, daß s die erste Ableitung der eindeutig bestimmten Splinefunktion

$S \in \Pi_{2m+1} \{ \pi_n \} \cap C^{2m} [0, T]$ ist, die die Stammfunktion $F(x) := \int_0^x f(t) dt$, $x \in [0, T]$, von f bzgl. der Partition π_n interpoliert und Hermite-Randbedingungen erfüllt. Mithin liefert die Anwendung geeigneter Fehlerabschätzungen aus [23] bzw. [12] für den Fall $m=1$ die Behauptung.

Beweis von Satz 5.2:

(i) Wegen $c_i = c_{2m-i}$, $i=0, \dots, m$, gilt für das charakteristische Polynom die Beziehung $p_m(x) = x^{2m} \cdot p_m(\frac{1}{x})$. In [22] wird gezeigt, daß p_m m einfache Nullstellen $\{ \alpha_i^{(m)} \mid i=1, \dots, m \}$ im offenen Intervall $(-1, 0)$ besitzt. Auf Grund obiger Relation ist dann $\{ (\alpha_i^{(m)})^{-1} \mid i=1, \dots, m \} \subset (-\infty, -1)$ die Menge der m weiteren Nullstellen von p_m . Damit sind alle Voraussetzungen des Satzes 1 aus [22] erfüllt, der aussagt, daß die Differenzengleichung

$$\sum_{i=0}^{2m} c_i \cdot u_{j+i} = r_j, \quad j \in \mathbb{Z},$$

eine eindeutige beschränkte Lösung besitzt, falls $\{ r_j \}_{j \in \mathbb{Z}}$ eine beschränkte Folge ist. Aus der expliziten Formel

$$u_j = \sum_{l \in \mathbb{Z}} \sum_{i=1}^m \frac{(\alpha_i^{(m)})^{m-1+|l|}}{p_m'(\alpha_i^{(m)})} \cdot r_{j-m+1}, \quad j \in \mathbb{Z},$$

für diese Lösung läßt sich durch Einsetzen der rechten Seite $\{\delta_{0j}\}_{j \in \mathbb{Z}}$ sofort die angegebene Darstellung des Kardinal-splines gewinnen. Da nach Voraussetzung die Funktion f und demnach auch die Folge ihrer lokalen Integrale bzgl. π_h gleichmäßig beschränkt ist, hat auch das gestellte Interpolationsproblem gemäß des zitierten Satzes genau eine Lösung, die sich auf Grund der Eigenschaften des Kardinal-splines in der angegebenen Form schreiben läßt.

(ii) Sei $f \in C(\mathbb{R}) \cap B(\mathbb{R})$ mit $\|f\|_{\infty} = 1$. Für die Koeffizienten $a_j, j \in \mathbb{Z}$, der Kardinalsplines in der Darstellung $s(t) = \sum_{j \in \mathbb{Z}} a_j \cdot k(\frac{t}{h} - j)$ des entsprechenden interpolierenden Splines aus $\Pi_2(\pi_h) \cap C^1(\mathbb{R})$ gilt mithin $|a_j| \leq 1, j \in \mathbb{Z}$. Zur Bestimmung der Operatornorm werden wir im folgenden das Maximum aller jener Splines berechnen, deren Koeffizienten die genannten Bedingungen erfüllen. Mit der Lösung $\alpha_1^{(1)} =: \alpha = \sqrt{3} - 2 \in (-1, 0)$ des charakteristischen Polynoms $p_1(x) = \frac{1}{6}x + \frac{4}{6}x + \frac{1}{6}x^2$ gilt $(p_1'(\alpha_1^{(1)}))^{-1} = \sqrt{3}$. Damit können wir schreiben

$$\begin{aligned} s(t) &= \sum_{j \in \mathbb{Z}} a_j \cdot k(\frac{t}{h} - j) = \sum_{j \in \mathbb{Z}} a_j \cdot \sum_{l \in \mathbb{Z}} \sqrt{3} \cdot \alpha^{|l|} \cdot b_2(\frac{t}{h} - 1 - j) \\ &= \sqrt{3} \cdot \sum_{j \in \mathbb{Z}} a_j \cdot \sum_{l \in \mathbb{Z}} \alpha^{|l-j|} \cdot b_2(\frac{t}{h} - 1) \\ &= \sqrt{3} \cdot \sum_{l \in \mathbb{Z}} \left(\sum_{j \in \mathbb{Z}} a_j \cdot \alpha^{|l-j|} \right) \cdot b_2(\frac{t}{h} - 1), \quad t \in \mathbb{R}, \end{aligned}$$

wobei $l := i + j$ sei. Wir betrachten nun o.B.d.A. das Intervall $[-\frac{h}{2}, \frac{h}{2}]$. Dort hat s die Darstellung

$$s(t) = \sqrt{3} \cdot \sum_{l=-1,0,1} \left(\sum_{j \in \mathbb{Z}} a_j \cdot \alpha^{|l-j|} \right) \cdot b_2(\frac{t}{h} - 1)$$

$$= \sqrt{3} \cdot \left\{ \left(\sum_{j \in \mathbb{Z}} a_j \cdot \alpha^{|j+1|} \right) \cdot \frac{1}{2} \cdot \left(\frac{t-1}{h} \right)^2 + \left(\sum_{j \in \mathbb{Z}} a_j \cdot \alpha^{|j|} \right) \cdot \left(\frac{3}{4} \cdot \left(\frac{t}{h} \right)^2 + \left(\sum_{j \in \mathbb{Z}} a_j \cdot \alpha^{|j-1|} \right) \cdot \frac{1}{2} \cdot \left(\frac{t+1}{h} \right)^2 \right) \right\}.$$

Vor weiteren Umformungen führen wir die Variablentransformation $x := \frac{t}{h}$ durch. Im Intervall $[-\frac{1}{2}, \frac{1}{2}]$ gilt dann

$$\begin{aligned} s(x) &= \sqrt{3} \cdot \sum_{j \in \mathbb{Z}} a_j \cdot \left\{ \alpha^{|j+1|} \cdot \frac{1}{2} \cdot \left(x - \frac{1}{2} \right)^2 + \alpha^{|j|} \cdot \left(\frac{3}{4} - x^2 \right) + \alpha^{|j-1|} \cdot \frac{1}{2} \cdot \left(x + \frac{1}{2} \right)^2 \right\} \\ &= \sqrt{3} \cdot a_0 \cdot \left\{ \alpha \cdot \left(x^2 + \frac{1}{4} \right) + \left(\frac{3}{4} - x^2 \right) \right\} + \\ &\quad \sqrt{3} \cdot \sum_{j=1}^{\infty} a_j \cdot \alpha^j \cdot \left\{ \alpha \cdot \frac{1}{2} \cdot \left(x - \frac{1}{2} \right)^2 + \left(\frac{3}{4} - x^2 \right) + \frac{1}{\alpha} \cdot \frac{1}{2} \cdot \left(x + \frac{1}{2} \right)^2 \right\} + \\ &\quad \sqrt{3} \cdot \sum_{j=1}^{\infty} a_{-j} \cdot \alpha^j \cdot \left\{ \frac{1}{\alpha} \cdot \frac{1}{2} \cdot \left(x - \frac{1}{2} \right)^2 + \left(\frac{3}{4} - x^2 \right) + \alpha \cdot \frac{1}{2} \cdot \left(x + \frac{1}{2} \right)^2 \right\}. \end{aligned}$$

Die in dieser Formel auftretenden Grenzwerte sind wegen

$|a_j| \leq 1$, $j \in \mathbb{Z}$, beschränkt:

$$|A_i| := \left| \sum_{j=1}^{\infty} a_{i+j} \cdot \alpha^j \right| \leq \sum_{j=1}^{\infty} |\text{sign}(\alpha^j)| \cdot \alpha^j = \sum_{j=1}^{\infty} (2 - \sqrt{3})^j = \frac{1}{2}(\sqrt{3} - 1),$$

$i \in \{+1, -1\}$. Bei festgehaltenen Koeffizienten a_0 , A_1 und A_{-1}

nimmt das quadratische Polynom $s|_{[-\frac{1}{2}, \frac{1}{2}]}$ sein Extremum in

den Randpunkten $\pm \frac{1}{2}$ des Intervalls oder in der Nullstelle seiner ersten Ableitung an, falls diese in $[-\frac{1}{2}, \frac{1}{2}]$ existiert.

Andererseits stellt die Maximierung von $|s(x)|$ für jeden

Wert der Variablen $x \in [-\frac{1}{2}, \frac{1}{2}]$ ein lineares Optimierungsproblem bzgl. a_0 , A_1 und A_{-1} sowie der Nebenbedingungen

$|a_0| \leq 1$ und $|A_i| \leq \frac{1}{2}(\sqrt{3} - 1)$ für $i \in \{+1, -1\}$ dar. Das Optimum

wird bekanntlich in den Eckpunkten der zulässigen Lösungsmenge angenommen, so daß wir im folgenden nur die 8 Fälle

$(a_0, A_1, A_{-1}) = (\pm 1, \pm \frac{1}{2}(\sqrt{3} - 1), \pm \frac{1}{2}(\sqrt{3} - 1))$ zu betrachten brauchen.

Aus

$$s'(x) = \sqrt{3} \cdot \{ a_0 \cdot [2\alpha x - 2x] + A_1 \cdot [\alpha \cdot (x - \frac{1}{2}) - 2x + \frac{1}{\alpha} \cdot (x + \frac{1}{2})] +$$

$$A_{-1} \cdot \left[\frac{1}{\alpha} \cdot \left(x - \frac{1}{2} \right) - 2x + \alpha \cdot \left(x + \frac{1}{2} \right) \right], \quad x \in [-1, 1],$$

ergibt sich die Nullstelle von s' zu

$$x_N = \frac{(A_1 - A_{-1}) \cdot \left(\frac{\alpha}{2} - \frac{1}{2\alpha} \right)}{a_0 \cdot (2\alpha - 2) + (A_1 + A_{-1}) \cdot \left(\alpha - 2 + \frac{1}{\alpha} \right)}.$$

In den Fällen $A_1 = A_{-1}$ gilt $x_N = 0$, da der Nenner für alle möglichen Kombinationen von $a_0, \epsilon \in \{-1, 1\}$ nicht verschwindet:

$$a_0 \cdot (2\alpha - 2) + 2A_1 \cdot \left(\alpha - 2 + \frac{1}{\alpha} \right) = a_0 \cdot (2\sqrt{3} - 4 - 2) + 2\epsilon \cdot \frac{1}{2} \cdot (\sqrt{3} - 1) \cdot (\sqrt{3} - 2 - 2 - \sqrt{3} - 2) \\ = (2a_0 - 6\epsilon) \cdot \sqrt{3} + 6 \cdot (\epsilon - a_0). \text{ Ansonsten, d.h. wenn } A_1 = -A_{-1} \text{ ist,}$$

erhalten wir

$$x_N = \frac{A_1}{a_0} \cdot \frac{\alpha - \frac{1}{\alpha}}{2\alpha - 2} = \frac{1}{a_0} \cdot \frac{\sqrt{3} - 1}{2} \cdot \frac{\sqrt{3} - 2 + \sqrt{3} + 2}{2\sqrt{3} - 4 - 2} = \frac{1}{a_0} \cdot \frac{3 - \sqrt{3}}{2\sqrt{3} - 6} = \frac{1}{a_0} \cdot \frac{(3 - \sqrt{3})(2\sqrt{3} + 6)}{12 - 36} = \\ \frac{1}{a_0} \cdot \frac{18 - 6}{-24} = -\frac{1}{2a_0} = \pm \frac{1}{2}.$$

Da wir den Spline s an den Stellen $\pm \frac{1}{2}$ und 0 auswerten müssen, berechnen wir die Funktionswerte in Abhängigkeit von a_0 ,

A_1 und A_{-1} :

$$s(0) = \sqrt{3} \cdot \left\{ \frac{\sqrt{3} + 1}{4} \cdot a_0 + \frac{1}{4} \cdot (A_1 + A_{-1}) \right\} \text{ sowie}$$

$$s\left(\pm \frac{1}{2}\right) = \sqrt{3} \cdot \left\{ \frac{\sqrt{3} - 1}{2} \cdot a_0 - \frac{\sqrt{3} + 1}{2} \cdot A_1 + \frac{\sqrt{3} - 1}{2} \cdot A_{-1} \right\} \text{ für } i \in \{+1, -1\}.$$

Die Betrachtung der 8 Eckpunkte der Lösungsmenge führen wir in der folgenden Fallunterscheidung durch.

1. Fall: $A_1 = A_{-1} = i \cdot \frac{1}{2}(\sqrt{3} - 1)$ mit $a_0, i \in \{+1, -1\}$.

Wir haben s für die Abszissen 0 und $\pm \frac{1}{2}$ auszuwerten:

$$|s(0)| = \left| \sqrt{3} \cdot \left\{ \frac{\sqrt{3} + 1}{4} \cdot a_0 + \frac{\sqrt{3} - 1}{4} \cdot i \right\} \right| \leq \frac{3}{2} \text{ und} \\ |s(\pm \frac{1}{2})| = \left| \sqrt{3} \cdot \left\{ \frac{\sqrt{3} - 1}{2} \cdot a_0 + \left(\frac{\sqrt{3} - 1}{2} - \frac{\sqrt{3} + 1}{2} \right) \cdot \frac{\sqrt{3} - 1}{2} \cdot i \right\} \right| = \left| \frac{3 - \sqrt{3}}{2} \cdot (a_0 - i) \right| \leq 3 - \sqrt{3}.$$

2. Fall: $A_1 = -A_{-1} = i \cdot \frac{1}{2}(\sqrt{3} - 1)$ mit $a_0, i \in \{+1, -1\}$.

Hier brauchen wir s nur in den Randpunkten des Intervalls auszuwerten:

$$|s(\pm \frac{1}{2})| = \left| \sqrt{3} \cdot \left\{ \frac{\sqrt{3} - 1}{2} \cdot a_0 - \left(\frac{\sqrt{3} + 1}{2} + \frac{\sqrt{3} - 1}{2} \right) \cdot \frac{\sqrt{3} - 1}{2} \cdot i \right\} \right| =$$

$$\left| \frac{3-\sqrt{3}}{2} \cdot (a_0 - \sqrt{3} \cdot 1) \right| \leq \sqrt{3}.$$

Mit dem letzten Wert $\sqrt{3}$ für $\|s\|_\infty$ haben wir eine obere Schranke für die Norm des Interpolationsoperators gefunden, die von dem mit den Koeffizienten $a_0=1$, $a_j=(-1)^{j+1}$ und $a_{-j}=(-1)^j$, $j \in \mathbb{N} \setminus \{0\}$, konstruierten Spline s angenommen wird. Da es Folgen stetiger Funktionen mit der Norm 1 und der Eigenschaft, daß die Folgen der entsprechenden Kardinal-splinekoeffizienten gegen die oben angegebenen Werte konvergieren, gibt, ist $\sqrt{3}$ auch die kleinste obere Schranke.

Die Beweise von Satz 5.3 können [11] entnommen werden.

Beweis von Satz 5.4:

(i) Die Existenz und Eindeutigkeit der besten L^2 -Approximierenden folgt sofort aus der Vollständigkeit des von den $n+m$ linear unabhängigen B-Splines

$\{b_m(\frac{\cdot}{h}-j-\frac{m+1}{2}) \mid [0, T] \mid j=-m, \dots, n-1\}$ aufgespannten Unterraumes des Hilbertraumes $L^2[0, T]$.

(ii) Das angegebene lineare Gleichungssystem ist äquivalent mit den notwendigen Bedingungen

$\frac{\partial}{\partial a_i} \|f-s\|_2^2 = 0$, $i=-m, \dots, n-1$, zur Bestimmung des Splines s m -ten Grades bzgl. π_n , der die Fehlernorm $\|f-s\|_2$ minimiert.

(iii) Mit (\dots) wollen wir das innere Produkt von $L^2[0, T]$, mit S den Unterraum $\text{span}\{b_m(\frac{\cdot}{h}-j-\frac{m+1}{2}) \mid [0, T] \mid j=-m, \dots, n-1\}$ und mit $P: S \rightarrow L^2[0, T]$ den entsprechenden Orthogonalprojektionsoperator bezeichnen. Wir werden zeigen, daß $s=Ps$ die $(m+1)$ -te Ableitung der Splineinterpolierenden vom Grade

$2m+1$ bzgl. π_n und Hermite-Randbedingungen des $(m+1)$ -ten iterierten Integrals von f ist.

Sei $Q := \Pi_{2m+1} \{ \pi_n \} \cap C^{2m}[0, T]$ und R der Splineinterpolationsoperator bzgl. Q , π_n und Hermite-Randbedingungen. Wir definieren die Abbildung

$$A: \begin{cases} C[0, T] \rightarrow C^{m+1}[0, T] \\ f(t) \mapsto \frac{1}{m!} \cdot \int_0^t (t-x)^m \cdot f(x) dx \end{cases},$$

für die $D^j A(f)(0) = 0$, $j=0, \dots, m$, $D^{m+1} A(f) = f$ und

$A(S) = Q' := \{ q \in Q \mid q^{(j)}(0) = 0, j=0, \dots, m \}$ gilt. Da P ein Orthogonalprojektor ist, haben wir $(f - Pf, u) = 0$ für alle $u \in S$, oder anders geschrieben

$$(D^{m+1} A(f) - D^{m+1} A(Pf), D^{m+1} A(u)) = 0 \text{ bzw. } (D^{m+1} A(f) - D^{m+1} A(Pf), D^{m+1} q) = 0$$

für alle $q \in Q'$. Davon subtrahieren wir die aus der für die Splineinterpolation gültigen Orthogonalitätsrelation

Im $D^{m+1} R \perp \text{Im } D^{m+1} (Id - R)$ folgende Gleichung

$$(D^{m+1} A(f) - D^{m+1} RA(f), D^{m+1} q) = 0 \text{ für alle } q \in Q:$$

$$(D^{m+1} A(Pf) - D^{m+1} RA(f), D^{m+1} q') = 0 \text{ für alle } q \in Q'.$$

Da auch $A(Pf) - RA(f) \in Q'$ ist, gilt mit Anwendung der "einseitigen" Rayleigh-Ritz-Ungleichung:

$$0 = \| D^{m+1} [A(Pf) - RA(f)] \|_2^2 \geq \left(\frac{\pi}{2T} \right)^{2m+2} \cdot \| A(Pf) - RA(f) \|_2^2 > 0,$$

d.h. $A(Pf) = RA(f)$ und mithin $Pf = D^{m+1} RA(f)$, woraus nach [23]

$$\begin{aligned} \| D^j (f - Pf) \|_\infty &= \| D^j (D^{m+1} A(f) - D^{m+1} RA(f)) \|_\infty \\ &= \| D^{m+1+j} (A(f) - RA(f)) \|_\infty \\ &\leq K(m, j, T) \cdot \left(\frac{T}{n} \right)^{2m+2-m-1-j} \cdot \| D^{2m+2} A(f) \|_\infty \\ &= K(m, j, T) \cdot \left(\frac{T}{n} \right)^{m+1-j} \cdot \| D^{m+1} f \|_\infty \end{aligned}$$

für $j=0, \dots, m$ folgt.

Literaturverzeichnis

- [1] Aho,A.V., Hopcroft,J.E., Ullman,J.D.: The Design and Analysis of Computer Algorithms. Reading, Mass.: Addison-Wesley 1974
- [2] Bauer,F.L.: Beiträge zur Entwicklung numerischer Verfahren für programmgesteuerte Rechenanlagen. II. Direkte Faktorisierung eines Polynoms. Sitz. Ber. Bayer. Akad. Wiss. 1956, 163-203.
- [3] Baumann,R. et al.: Funktionelle Beschreibung von Prozeßrechner-Betriebssystemen. VDI-Richtlinie VDI/VDE 3554. Düsseldorf: VDI-Verlag 1976
- [4] Becker,W. et al.: Einsatzmöglichkeiten externer Funktionseinheiten in prozeßrechnergestützten Automatisierungssystemen. KFK-PDV 23, Karlsruhe, 1974
- [5] Brunner,P.J., Bösmann,H., Tarabout,A., Werum,W.: Universelles PEARL-Betriebssystem. KFK-PDV 55, Karlsruhe, 1976
- [6] Coffman,E.G., Denning,P.J.: Operating Systems Theory. Ch. 3.7. Englewood Cliffs, N.J.: Prentice-Hall 1973
- [7] DIN 44300
- [8] Ecker,K.: Organisation von parallelen Prozessen. Mannheim: Bibliographisches Institut 1977
- [9] Eichenauer,B.: Dynamische Prioritätsvergabe an Tasks in Prozeßrechensystemen. Dissertation. Universität Stuttgart 1975
- [10] Halang,W.A.: Über interpolierende C^∞ -Spline-Funktionen. Dissertation. Ruhr-Universität Bochum 1976

- [11] Halang, W.A.: Approximation durch positive lineare Spline-Operatoren konstruiert mit Hilfe lokaler Integrale. Erscheint im Journal of Approximation Theory.
- [12] Hall, C.A.: On Error Bounds for Spline Interpolation. J. Approx. Theory 1, 209-218 (1968)
- [13] Havender, J.W.: Avoiding Deadlock in Multitasking Systems. IBM Sys. J. 7, 74-84 (1968)
- [14] Henn, R.: Deterministische Modelle für die Prozessorzuteilung in einer harten Realzeit-Umgebung. Dissertation. TU München 1975
- [15] Henn, R.: Zeitgerechte Prozessorzuteilung in einer harten Realzeit-Umgebung. GI - 6. Jahrestagung. Informatik-Fachberichte 5. Berlin, Heidelberg: Springer-Verlag 1976
- [16] Henn, R.: Ein antwortzeitgesteuertes Unterbrechungswerk - Auswirkungen auf Betriebssystem und Programmstruktur. GMR-GI-GfK. Fachtagung Prozeßrechner 1977. Informatik-Fachberichte 7. Berlin, Heidelberg: Springer-Verlag 1977
- [17] Henn, R.: Antwortzeitgesteuerte Prozessorzuteilung unter strengen Zeitbedingungen. Computing 19, 209-220 (1978)
- [18] Holler, E., Drobnik, O., Knöpker, R.: Entwurf und Modellierung von Mehrrechnersystemen für Prozeßlenkungsaufgaben. KFK-PDV 57, Karlsruhe, 1975
- [19] Johnson, H.H., Maddison, M.: Deadline Scheduling for a Real-Time Multiprocessor. Eurocomp. Conf.

Proceedings, 139-153, 1974

- [20] Labetoulle, J.: Ordonnancement des Processus Temps Réel sur une ressource préemptive. Thèse de 3ème cycle, Université Paris VI.
- [21] Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. JACM 20, 46-61 (1973)
- [22] Subbotin, Ju.N.: On the Relations between Finite Differences and the Corresponding Derivatives. Proc. Steklov Inst. Math. 78, 23-42 (1965)
- [23] Swartz, B.K., Varga, R.S.: Error Bounds for Spline and L-Spline Interpolation. J. Approx. Theory 6, 6-49 (1972)
- [24] Tränkler, H.-R.: Die Technik des digitalen Messens. München: R. Oldenbourg Verlag 1976

Offeildruck
GmbH & Co. KG

86 Bamberg Hohe-Kreuz-Str. 9b

M. SCHADEL

• Tel. 0951/54414-58481 •