Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Verteilte Systeme
Arbeitsgruppe Echtzeitsysteme und Kommunikation

Diplomarbeit

Thema:

Scheduling- und Kommunikationskonzept für ein verteiltes, DSP-basiertes Steuerungssystem eines Rasterkraftmikroskops

März 2004

vorgelegt von: Christoph Walter
German-Titow-Str. 7
06449 Aschersleben

chwalter@cs.uni-magdeburg.de

Kurzfassung

Mit Hilfe der Rastersondenmikroskopie ist es möglich, einen kleinen Ausschnitt der Oberfläche von zu untersuchenden Proben mit hoher lateraler Auflösung abzutasten. Es können im Idealfall Abbildungen erzeugt werden, in denen einzelne Atome zu unterscheiden sind. Im Rahmen einer Weiterentwicklung eines bestehenden Rasterkraftmikroskops soll ein Konzept für Teilaspekte der Steuerungssoftware dieses Gerätes erarbeitet werden. Die Steuerungssoftware setzt hardwareseitig auf zwei unabhängige DSP-Boards auf, die beide über eine Kommunikationsverbindung mit einem PC verbunden sind. Auf diesem erfolgt die Parametrisierung von Scan-Vorgängen sowie die Darstellung und Auswertung der Resultate mit Hilfe eines speziellen Anwendungsprogramms. Die Entwicklung eines neuen Task- und Kommunikationskonzepts für die Steuerungssoftware wurde primär notwendig, da die bestehende Lösung bei einer Erweiterung um zusätzliche Aufgaben die Anforderungen an deren zeitlicher Ausführung sowie des Austauschs von Daten zunehmend nicht mehr erfüllen konnte.

Ausgehend von dieser Grundlage werden die Aufgaben der Steuerungssoftware unter dem besonderen Aspekt ihrer zeitlichen Anforderungen untersucht. Es wird darauf aufbauend zum einen ein dynamisches Scheduling-Konzept für die identifizierten Teilaufgaben vorgestellt. Zum anderen wird ein Konzept für den Austausch von Daten zwischen den DSP-Boards sowie dem PC erarbeitet. Hierbei werden insbesondere die Gegebenheiten der Hardwarearchitektur berücksichtigt. Des Weiteren werden Ansätze zur softwaretechnischen Umsetzung dieser Konzepte betrachtet sowie einige ausgewählte Aspekte implementiert.

Abstract

By using Atomic Force Microscopy it is possible to scan a small part of the surface of some specimen with very high lateral resolution. Ideally it is even possible to create pictures where it is possible to differentiate between single atoms. A conception of some aspects of the control-software of an existing Atomic Force Microscope will presented within this paper. The development takes place in the scope of general development work on such a microscope. The control-software is based on two independent DSP-boards that are connected by some communication facility to a PC. The PC runs an application program that is used by the operator of the microscope to parameterize and carry out the scanning process. The development of a new concept for task-scheduling and communication was primarily necessary since the existing solution, when further extended, has proven to be unable to handle time constraints of the execution of its tasks as well as of the exchange of data.

Starting from that point, the tasks of the control-software are examined regarding their scheduling requirements. Based on that, a dynamic scheduling approach is presented. Additionally a concept for the exchange of data between the DSP-boards and the PC is shown. Furthermore, an approach for s suitable implementation is discussed and the implementation of some selected aspects is presented.

Inhaltsverzeichnis

Kurzfassung ... **III**

Abstract ... **IV**

Inhaltsverzeichnis ... **V**

Abkürzungsverzeichnis ... **VIII**

Abbildungsverzeichnis ... **IX**

Tabellenverzeichnis ... **X**

1 Einführung ... **1**
- 1.1 Einleitung und Motivation ... 1
- 1.2 Kontext und verwandte Arbeiten ... 2
- 1.3 Ergebnisse ... 3
- 1.4 Gliederung ... 3

2 Grundlagen und verwandte Arbeiten ... **5**
- 2.1 Aufbau und Eigenschaften von Echtzeitsystemen ... 5
- 2.2 Task und Taskplanung ... 6
  - 2.2.1 Der Task-Begriff ... 6
  - 2.2.2 Bestimmung von Ausführungszeiten ... 8
  - 2.2.3 Scheduling ... 10
  - 2.2.4 Fehlerbehandlung ... 11
    - 2.2.4.1 Verringerung der Qualität von Berechnungen ... 12
    - 2.2.4.2 TaskPair-Scheduling und TAFT ... 12
  - 2.2.5 Prozesssynchronisation ... 13
- 2.3 Echtzeitbetriebssysteme ... 14
  - 2.3.1 Aufgaben und Nutzen ... 14
  - 2.3.2 Einbindung von Anwendungssoftware ... 15
  - 2.3.3 Beispiele für Scheduling-Verfahren in RTOSs ... 17
  - 2.3.4 Das Betriebsystem MicroC/OS-II ... 17
    - 2.3.4.1 Allgemeines ... 17
    - 2.3.4.2 Funktionalität ... 18
    - 2.3.4.3 Erweiterungen ... 19
- 2.4 Verteilte Systeme ... 20
  - 2.4.1 Eigenschaften und Anwendungen ... 20
  - 2.4.2 Verteilte Betriebssysteme ... 21
  - 2.4.3 Kommunikation in verteilten Systemen ... 21

3 Gerätetechnische Grundlagen ... **23**
- 3.1 Die Rastersondenmikroskopie ... 23
  - 3.1.1 Funktionsprinzip und Verfahrensweisen ... 23
  - 3.1.2 Die Rasterkraftmikroskopie ... 25
  - 3.1.3 Anwendungsgebiete ... 28
- 3.2 Digitale Signalprozessoren ... 28
  - 3.2.1 Die Harvard-Architektur ... 30

3.2.2 Anwendungsgebiete ... 31
3.2.3 Softwareentwicklung für DSP-Anwendungen ... 32
3.3 Aufbau des vorliegenden Systems ... 34
3.3.1 Überblick ... 34
3.3.2 Der ADSP-21061 ... 35
3.3.3 XY-System ... 37
3.3.4 Z-System ... 38

4 Ist-Zustand der bestehenden Softwarelösung ... 41
4.1 DSP-Software ... 41
4.2 Anwendungsprogramm ... 42
4.3 Die Kommunikationslösung ... 43
4.4 Nachteile der existierenden Lösung ... 45
4.5 Zielsetzungen bei der Konzeption ... 46

5 Analyse der Aufgaben der Steuerungssoftware ... 47
5.1 Anwendungsszenarien des Gesamtsystems ... 47
5.1.1 Abbildung von Proben ... 47
5.1.2 Weitere Untersuchungen an Proben ... 48
5.1.3 Werkzeug zur Micromanipulation ... 48
5.2 Arbeitsschritte bei der Bilderfassung ... 49
5.3 Identifikation von Funktionsgruppen der Steuerungssoftware ... 50
5.3.1 Oszilloskop ... 51
5.3.2 Tip Approach ... 52
5.3.3 Force Calibration ... 53
5.3.4 Digitale Abstandsregelung ... 54
5.3.5 Abtasten einer Zeile ... 56
5.3.6 Ausführen der Scanbewegung ... 56
5.3.7 Weitere Aufgaben ... 57
5.4 Zusammenfassung der zeitlichen Anforderungen der Teilaufgaben ... 57
5.5 Geplante zukünftige Aufgaben ... 58

6 Konzeption der Steuerungssoftware ... 60
6.1 Prinzipielle Lösungsansätze ... 60
6.1.1 Auswahl geeigneter Systemsoftware ... 61
6.1.2 Einsatz von MicroC/OS-II im Rahmen dieser Arbeit ... 62
6.2 Das Konzept des Supertasks ... 63
6.3 Aussetzen einzelner Instanzen des Supertasks ... 64
6.3.1 Motivation ... 64
6.3.2 Ansätze ... 65
6.3.3 Vergleich mit dem Ansatz des TaskPair-Schedulings ... 66
6.3.4 Auswirkungen auf das Systemverhalten ... 67
6.3.4.1 Arbeitsfrequenzen der Z-Aktoren ... 68
6.3.4.2 Aktoren im XY-System ... 70
6.3.4.3 Anpassung des Regelungsalgorithmus ... 71
6.4 Realisierung des Aussetzens von Instanzen ... 72
6.4.1 Anforderungen an den Algorithmus ... 72
6.4.2 Der Algorithmus ... 73

6.5 Kommunikation 76
6.5.1 Zu übermittelnde Daten 76
6.5.2 Ansatz zur Erweiterung der bestehenden Lösung 78
6.5.2.1 Berücksichtigung von Zeitanforderungen der Nutzdaten 78
6.5.2.2 Synchrone Ereignisse 79
6.5.2.3 Integration des Kommunikationskonzepts in die Software 81

7 Implementierung 84
7.1 Anpassungen der Systemsoftware an die Hardware 84
7.2 Umsetzung der Steuerungssoftware mit µC/OS-II Tasks 85
7.3 Erweiterung der Systemsoftware für den Supertask 86
7.4 Laufzeitstatistiken 88

8 Ergebnisse 89
8.1 Einsatz einer Simulation 89
8.2 Anwendung im realen System 92

9 Zusammenfassung und Ausblick 94

Referenzen / Literatur 96

Anhang 101

A Systemfunktionen von MicroC/OS-II 101

B Enhanced Parallel Port Hardware 104

C ISO/OSI-Referenzmodell 105

Selbstständigkeitserklärung 106

Abkürzungsverzeichnis

ADC	Analog-Digital-Converter
AFM	Atomic Force Microscope
API	Application Programming Interface
BCET	Best Case Execution Time
DAC	Digital-Analog-Converter
DMA	Direct Memory Access
DSP	Digitaler Signalprozessor
ECET	Expected Case Execution Time
EDF	Earliest Deadline First
EPP	Enhanced Parallel Port
EPROM	Electrical Programmable Read Only Memory
ET	Execution Time
FIFO	First In First Out
FPGA	Field Programmable Gate Array
QoS	Quality of Service
RM	Rate Monotonic
RT	Real-Time
RTS	Real-Time System
SNOM	Scanning Near field Optical Microscope
SRAM	Static Random Access Memory
STM	Scanning Tunneling Microscope
STOM	Scanning Tunneling Optical Microscope
TAFT	Time Aware Fault Tolerant
WCET	Worst Case Execution Time

Abbildungsverzeichnis

Abb. 2.1: Grundmodell eines Echtzeitsystems 6
Abb. 2.2: Prozesse in einem RTS 7
Abb. 2.3: TaskPair-Scheduling 13
Abb. 2.4: Prinzip von RT-Linux 16
Abb. 3.1: Prinzip eines Rastersondenmikroskops 24
Abb. 3.2: Cantilever mit Spitze 26
Abb. 3.3: Kraft-Distanz-Kurve bei Annäherung der Sonde 26
Abb. 3.4: Detektion der Kraftwirkung Probe - Messspitze am AFM 27
Abb. 3.5: Harvard- und Von-Neumann-Architektur 30
Abb. 3.6: Werkzeuge bei der Softwareentwicklung für DSP-Systeme 33
Abb. 3.7: Überblick über den Systemaufbau 34
Abb. 3.8: SHARC ADSP-21061 Block Diagramm 36
Abb. 3.9: XY-Systems mit angeschlossenen Komponenten 37
Abb. 3.10: Z-Systems mit angeschlossenen Komponenten 39
Abb. 4.1: Foreground/Background-System der bestehenden Lösung 41
Abb. 4.2: Hardwarekomponenten der Kommunikationsschnittstelle 43
Abb. 4.3: Schema der Schichten des bestehenden Kommunikationsmodells 44
Abb. 5.1: Mit dem AFM aufgenommenes, topografisches Profil 48
Abb. 5.2: Übersicht der Arbeitsschritte bei der Bilderfassung 49
Abb. 5.3: Zeitlicher Ablauf der Oszilloskop-Funktion 52
Abb. 5.4: Zeitlicher Ablauf des Tip Approach 53
Abb. 5.5: Aufnehmen der Kraft-Abstands Kurve 54
Abb. 5.6: Prinzip der digitalen Regelung 55
Abb. 5.7: Ablauf der XY-Scanbewegung 57
Abb. 6.1: Konzept des Supertasks 63
Abb. 6.2: Aussetzen des Supertasks 66
Abb. 6.3: Bestimmung des maximalen Ausschlags des Cantilevers 69
Abb. 6.4: Pseudocode des Algorithmus zum Aussetzen von Task-Instanzen 74
Abb. 6.5: Übertragung von Daten mit unterschiedlichen Prioritäten 79
Abb. 6.6: Ablauf der Signalisierung eines verteilten Ereignisses 81
Abb. 6.7: Kommunikationskonzept als Schichtendarstellung 82
Abb. 7.1: Erzeugungsreihenfolge der Anwendungstasks 85
Abb. 7.2: Integration des Supertasks in µC/OS-II 87
Abb. 8.1: Schema des Aussetzens des Supertasks bei konstanten Ausführungszeiten der Instanzen 89
Abb. 8.2: Schema des Aussetzens des Supertasks bei normalverteilten Ausführungszeiten der Instanzen 90
Abb. 8.3: Ausgelassene Instanzen des Supertasks pro Intervall 91
Abb. 8.4: Zusammenhängend ausgelassene Instanzen des Supertasks 92
Abb. 8.5: Vergleich der Bildqualität mit und ohne Aussetzen des Regelungstasks 93

Tabellenverzeichnis

Tab. 2.1: Zeitliche Parameter von Tasks 8
Tab. 3.1: Verschiedene Verfahren der Rastersondenmikroskopie 25
Tab. 3.2: Anwendungsgebiete von DSPs 31
Tab. 5.1: Erlaubte Kombinationen von Funktionsgruppen 51
Tab. A.1: Task Management 101
Tab. A.2: Time Management 101
Tab. A.3: Semaphoren 102
Tab. A.4: Mutexes 102
Tab. A.5: Event Flags 102
Tab. A.6: Mailbox Management 103
Tab. A.7: Message Queues 103
Tab. A.8: Memory Management 104
Tab. B.1: Signalbelegung am EPP 104
Tab. B.2: EPP Controller - Register 105
Tab. C.1: Schichten des ISO/OSI-Referenzmodells 105

1 Einführung

1.1 Einleitung und Motivation

Die Rasterkraftmikroskopie sowie verwandte Verfahren stellen eine Methode für die Untersuchung und Manipulation kleinster Strukturen dar. Mit der zunehmenden Verbreitung von Nanotechnologien gewinnt auch dieses Verfahren in vielfältigen Variationen an Bedeutung.

Die vorliegende Arbeit beschäftigt sich mit der Konzeption von Teilaspekten der Steuerungssoftware eines solchen Rasterkraftmikroskops. Sie entstand in enger Zusammenarbeit mit dem Fraunhofer Institut für Fabrikbetrieb und -automatisierung in Magdeburg. Im Rahmen der Weiterentwicklung eines bereits bestehenden Geräts erwies sich eine Neukonzeption der Steuerungssoftware als notwendig. Die bestehende Lösung setzt sich aus der Mikroskophardware mit angeschlossenem Steuerrechner sowie einem PC, auf dem ein entsprechendes Anwendungsprogramm läuft, zusammen. Das Anwendungsprogramm dient der Parametrisierung von Scan-Vorgängen sowie der Darstellung und Auswertung der Resultate. Der PC ist über eine Kommunikationsverbindung mit dem Steuerrechner verbunden, der sich aus zwei unabhängigen, auf digitalen Signalprozessoren basierenden Teilsystemen zusammensetzt.

Während die Hardwarekomponenten modular aufgebaut sind und sich gut modifizieren bzw. erweitern lassen, erwies sich das Konzept der bestehenden Software der beiden Teilsysteme des Steuerrechners als unzureichend. Die für die Steuerung der Hardware notwendige Echtzeitfähigkeit des hier verwendeten einfachen Foreground/Background-Systems stößt bei Erweiterung der Software an seine Grenzen. Ein besonderer Aspekt dabei ist die Einhaltung der maximalen Ausführungszeit einer hochfrequenten Regelschleife. Die bisher praktizierte, rechnerische Bestimmung der Ausführungszeiten dieser Schleife zur Vermeidung von Überlastsituationen hat sich als unpraktikabel und fehleranfällig erwiesen. Durch eine zukünftige Weiterentwicklung des hier zum Einsatz kommenden Regelalgorithmus ist mit einer rechnerisch noch schwieriger zu bestimmenden maximalen Ausführungszeit, sowie einer hiervon stärker abweichenden mittleren Ausführungszeit zu rechnen. Das Versagen der Software aufgrund von sich ergebenden temporären oder andauernden Überlastsituationen kann zu Beschädigungen des Geräts führen. Aus dieser Situation heraus soll ein geeignetes Konzept für das Scheduling von Abläufen innerhalb der Steuerungssoftware erarbeitet werden.

Ein weiterer Aspekt dieser Arbeit liegt in der Auswahl eines geeigneten Kommunikationskonzepts. Es sind hierbei zwei Gesichtspunkte zu berücksichtigen. Dabei handelt es sich zum einen um Unzulänglichkeiten der bestehenden Lösung, die bei der Benutzung der existierenden Systems aufgetreten sind, und die die unzureichenden Möglichkeiten zur zeitlichen Synchronisierung von Startzeitpunkten von Aufgaben beider Teilsysteme des Steuerrechners betreffen. Zum anderen sollen Möglichkeiten untersucht werden, wie bei einer zukünftigen Erweiterung des Geräts für Mikromanipulationen die hierbei anfallenden, zeitkritischen Daten mit dem PC ausgetauscht werden können.

1.2 Kontext und verwandte Arbeiten

Auf dem Gebiet des Designs von Echtzeitsystemen werden kontinuierlich Fortschritte erzielt. Ein Schwerpunkt liegt dabei auf der Nutzbarmachung von modernen, dynamischen Schedulingverfahren auf eine breite Palette von realen Anwendungen. Unter dem Aspekt einer möglichst effizienten Ausnutzung von Ressourcen ergibt sich dabei die Frage nach der Behandlung von temporären Überlastsituationen. Nachdem sich die Idee des dynamischen Eintauschens von Qualität von durchgeführten Berechnungen gegen die Einhaltung von damit verbundenen zeitlichen Anforderungen durchgesetzt hatte [Chung, Liu, Lin 1990], wurden verschiedene Konzepte zur Umsetzung dieser Idee unter der Berücksichtigung mehr oder weniger allgemeiner Aufgabenbereiche veröffentlicht. Das Grundprinzip ist hier die Aufspaltung eines Tasks in einen notwendigen und einen optionalen Teil, der das Ergebnis, wenn dafür Zeit bleibt, verfeinert. Ein anderer Ansatz ist das TaskPair-Scheduling, auf das noch detaillierter im zweiten Kapitel eingegangen wird.

Ein anderer Ansatz basiert auf der Erkenntnis, dass in vielen Anwendungsfällen ein gewisses Verletzen von zeitlichen Anforderungen akzeptabel ist. Im Kontext von solchen Weakly-Hard Real-Time Tasks stellen [Wang et al. 2002] ein Modell auf, welches Zeitüberschreitungen mit einer bestimmten Häufigkeit erlaubt. Bei der Betrachtung von Soft Real-Time Tasks kombinieren [Stankovic et al. 2000] Methoden der Regelungstechnik mit einem dynamischen Scheduling-Algorithmus mit Akzeptanztest. [Nawab et al. 1997] haben festgestellt, dass das Auslassen der Verarbeitung einzelner Abtastwerte bei der digitalen Signalverarbeitung akzeptabel sein kann und sich eignet, um die Auslastung von Ressourcen zu steuern.

Die vorliegende Arbeit steht insofern im Kontext dieser vorangegangenen Arbeiten, als das hier die dort gewonnenen Erkenntnisse um Möglichkeiten zur Verringerung der

Qualität einer realisierten Funktionalität angewandt werden auf den Anwendungsfall der Abstandsregelung eines Rasterkraftmikroskops.

1.3 Ergebnisse

Um die angestrebte, leichtere Erweiterbarkeit bzw. Modifizierbarkeit der Software des Steuerrechners zu gewährleisten, wurden die von der Steuerungssoftware zu erfüllenden Aufgaben entsprechend ihrer zeitlichen Anforderungen untersucht. Darauf aufbauend konnte die bestehende unflexible Lösung in mehrere Tasks aufgeteilt werden. Um diese geeignet umsetzten zu können wurde aufgrund hohen Aufwands für eine Eigenentwicklung ein bereits verfügbares Echtzeitbetriebssystem ausgewählt. Dieses wurde an die Hardware angepasst und erweitert. Ein auftretender und mit hoher Frequenz auszuführender Regelungs-Task wurde hinsichtlich seiner zeitlichen Anforderungen genauer untersucht. Wegen seiner sehr kurzen Periodenlänge ergab sich hier das Problem eines hohen Overheads für das Dispatching dieses Tasks. Es wurde aus diesem Grund für ihn das Konzept eines besonderen Tasks, welcher höchste Priorität hat und zu welchem mit besonders wenig Overhead umgeschaltet werden kann, vorgestellt und angewandt. Das Konzept dieses Super-Tasks wurde in die Systemsoftware integriert. Um Überlastsituationen, die durch unbekannte Ausführungszeiten dieses Tasks entstehen können noch während der Entwicklungszeit zu erkennen und gleichzeitig ein Versagen des Gesamtsystems zu vermeiden, wurde eine Methode erarbeitet, die die Qualität der durch diesen Task ausgeführten Aufgabe dynamisch herabsetzt. Dies wird durch das Aussetzen von einzelnen Instanzen des Tasks erreicht. Es wurde für den Anwendungsfall nachgewiesen, dass ein Aussetzen von Instanzen in einem weiten Rahmen möglich ist, ohne die Forderung nach der Vermeidung von Beschädigungen der Hardware zu vernachlässigen. Ausgehend von dieser Erkenntnis wurde ein Algorithmus erarbeitet, der drohende Überlastsituationen erkennt und darauf aufbauend entscheidet, welche Instanzen ausgesetzt werden. Der vorgestellte Algorithmus arbeitet dabei mit einer kurzen, konstanten Laufzeit und eignet sich somit hervorragend für den vorliegenden Einsatzfall, bei dem nur wenige Ressourcen zur Verfügung stehen. Durch eine Implementierung des Verfahrens konnte dessen Anwendbarkeit auf den Anwendungsfall praktisch überprüft werden.

1.4 Gliederung

Diese Arbeit ist in mehrere Kapitel gegliedert. Auf diese Einführung folgt im zweiten Kapitel eine Übersicht über den Themenbereich der Echtzeitsysteme, wobei auf zentrale

Begriffe dieses Bereichs eingegangen wird. Es wird dabei auch auf für diese Arbeit wichtige Konzepte verwandter Arbeiten eingegangen.

Das dritte Kapitel schildert gerätetechnische Grundlagen. Dabei wird das Verfahren der Rasterkraftmikroskopie sowie wichtige Grundlagen für seine Umsetzung vorgestellt. Es wird auf Digitale Signalprozessoren eingegangen und deren Besonderheiten und Anwendungsgebiete geschildert. Es wird außerdem der konkrete Systemaufbau geschildert und auf alle wichtigen Eigenschaften der Hardware eingegangen.

Im vierten Kapitel wird der Ist-Zustand der verwendeten Softwarelösung beschrieben. Es wird auf seine Vor- und Nachteile eingegangen. Weiterhin werden die Punkte, die eine Neukonzeption der Lösung notwendig gemacht haben dargestellt und davon ausgehend Zielsetzungen für diese Konzeption abgeleitet.

Das fünfte Kapitel dient der Vorbereitung des Task- und Kommunikationskonzepts. Es werden die Einsatzszenarien des Geräts vorgestellt. Darauf aufbauend werden Aufgaben der Steuerungssoftware identifiziert und entsprechend ihres Auftretens in Funktionsgruppen unterteilt. Weiterhin werden sie hinsichtlich ihrer zeitlichen Anforderungen untersucht.

Im sechsten Kapitel werden ausgehend von der existierenden Softwarelösung Ziele der Neukonzeption formuliert. Nachdem Lösungsansätze diskutiert wurden, wird auf den geplanten Einsatz des Echtzeitbetriebssystems MicroC/OS-II in einer erweiterten Variante eingegangen. Es wird das Supertask-Konzept vorgestellt und um ein dynamisches Scheduling-Verfahren mit Akzeptanztest erweitert. Es wird außerdem ein Konzept für die Kommunikation vorgestellt.

Im siebenten Kapitel werden für eine Implementierung notwendige Fragen diskutiert. Es wird sich darauf konzentriert die Einsatzfähigkeit des erweiterten Supertask-Konzepts mit Hilfe einer geeigneten Implementierung an dem vorliegenden Rasterkraftmikroskop zu überprüfen. Dafür wurde eine Implementierung ausgewählter wichtiger Teile der Steuerungssoftware vorgenommen.

Eine Darstellung der Ergebnisse der Arbeit im achten, sowie eine allgemeine Zusammenfassung im neunten Kapitel schließen die Arbeit ab.

2 Grundlagen und verwandte Arbeiten

In diesem Kapitel soll auf relevante Grundlagen von Echtzeitsystemen und Scheduling-Konzepten eingegangen werden. Wichtige Konzepte aus verwandten Arbeiten, welche bereits in der Einführung angesprochen worden sind, werden hier detaillierter dargestellt.

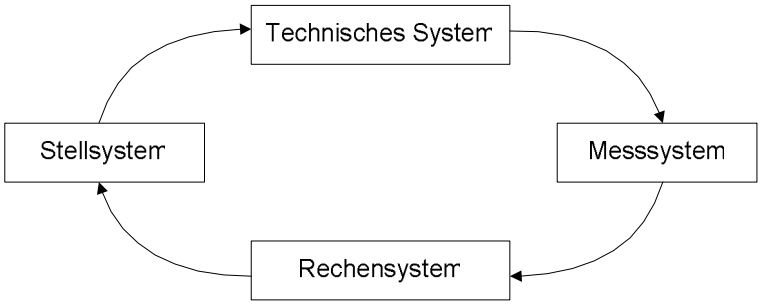
Die folgenden Betrachtungen sind notwendig, um spezifische Sachverhalte darzustellen, die speziell für die Problemlösung im Rahmen dieser Arbeit von Bedeutung sind. Es wird dabei auf computergestützte Echtzeitsysteme eingegangen. Dabei wird zum einen der Begriff "Echtzeitsystem" erläutert, indem anhand eines einfachen Modells auf Charakteristika und Anwendungsgebiete eines solchen Systems eingegangen wird. Zum anderen erfolgt eine nähere Diskussion der in Echtzeitsystemen anzutreffenden Systemsoftware und von verwendeten Schedulingverfahren. Es steht dabei nicht im Vordergrund allgemeine Grundlagen zu erörtern, sondern es wird auf relevante Arbeiten in dem Bereich eingegangen. Das Hauptaugenmerk liegt hierbei weniger auf spezifischen Implementierungen als vielmehr auf den zugrunde liegenden Konzepten.

2.1 Aufbau und Eigenschaften von Echtzeitsystemen

Das wesentliche Merkmal, durch das sich ein Echtzeitsystem auszeichnet, ist das Vorhandensein von Zeitbedingungen. Das heißt es verhält sich nur dann korrekt, wenn sich das System einerseits funktional richtig verhält, andererseits diese Funktionen aber auch innerhalb eines vorgegebenen Zeitrahmens ausgeführt werden.

In DIN 443000 wird über dieses Merkmal mithilfe des Begriffs des "Echtzeitbetriebs" folgendermaßen definiert: "Echtzeitbetrieb ist ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen."

Die Vorgabe von Zeitbedingungen ergibt sich aus dem typischen Anwendungsszenario von Echtzeitsystemen: Ein solches System ist oft ein Gerät, dass auf die physische Welt reagieren muss, indem es in sie eingreift [Ward, Mellor 1993, S. 14]. Das System muss sich dabei entsprechend an den zeitlichen Anforderungen der physischen Welt orientieren. Aus dem Einhalten von Zeitbedingungen leitet sich die Vorhersagbarkeit des Systemverhaltens ab. Der schematische Aufbau eines typischen Echtzeitsystems ist in Abbildung 2.1 dargestellt.



vgl.: [Zöbel, Albrecht 1995, S. 4]

Abb. 2.1: Grundmodell eines Echtzeitsystems

Man kann hier deutlich die Trennung zwischen dem Technischen System, welches Teil der physischen Welt ist und dem Rechensystem, das auf das Technische System reagiert, erkennen. Die Kopplung zwischen beiden erfolgt mittels Mess- und Stellsystem. Im weiteren Verlauf der Arbeit wird mit dem Begriff "Echtzeitsystem" oder englisch "Real-Time System" (RTS) v. a. auf das Rechensystem Bezug genommen.

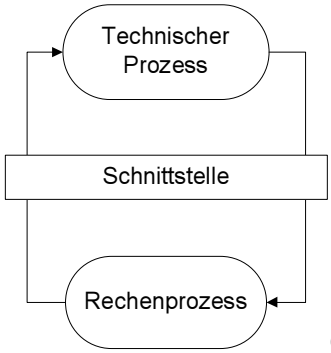
Durch ihre Eigenschaft, zeitliche Bedingungen mit zu berücksichtigen, sind Echtzeitsysteme in vielen Anwendungsgebieten zu finden. Z. B. für Steuer- oder Überwachungsaufgaben, in Telefonnetzen oder aber auch für Multimediageräte sind Echtzeitsysteme notwendig oder zumindest sinnvoll.

2.2 Task und Taskplanung

2.2.1 Der Task-Begriff

Der Task, ist ein zentraler Begriff bei der Betrachtung von Echtzeitsystemen. Dabei ist der Begriff Task recht allgemein gefasst und beschreibt allgemein eine zu erledigende Aufgabe. Unter einem Task soll im Folgenden ein Rechenprozess verstanden werden. Dieser besteht aus einer begrenzten Anzahl von Anweisung und kann auf dem entsprechenden Rechensystem innerhalb eines bestimmten Zeitrahmens ausgeführt werden. In einem RTS erfüllt ein solcher Task eine bestimmte Aufgabe und kann in vielen Fällen in Verbindung mit einem Technischen Prozess gesehen werden[1]. In diesem Zusammenhang der Technischen Prozesses über eine Schnittstelle wie z. B. ein Mess- und Stellsystem mit dem Rechenprozess in Verbindung. Dieser kann über diese Schnittstelle wiederum Einfluss auf den Technischen Prozess nehmen.

[1] siehe Abbildung 2.2



Quelle: [Zöbel, Albrecht 1995, S. 7]

Abb. 2.2: Prozesse in einem RTS

Tasks können gut nach ihrem wichtigsten Merkmal, ihrem zeitlichen Verhalten, charakterisiert werden. Eine wichtige Eigenschaft ist zum einen ihr Startzeitpunkt. Sie können in regelmäßigen Abständen starten (periodisch) oder in unregelmäßigen Abständen (aperiodisch), z. B. ausgelöst durch ein externes Ereignis. Eine Unterscheidung ist ebenfalls möglich nach Verdrängbarkeit. Tasks können während ihre Ausführung unterbrochen werden (preemptiv) damit andere Tasks ausgeführt werden können oder ohne Unterbrechung bis zu ihrer Beendigung laufen (non-preemptiv).

Einer konkreten Task-Instanz T_i sind neben ihrem Startzeitpunkt weitere Zeitbedingungen wie die Bereitzeit (engl. release time), der frühste Zeitpunkt an dem der Task ausgeführt werden kann, sowie die Frist[2] (deadline), bis zu der er abgeschlossen sein muss zuzuordnen. Die Menge aller Task-Instanzen T_i heißt T. Spätestens nach der Ausführung von T_i sind weiterhin seine konkrete Startzeit (starting time), Endzeit (completion time) sowie die Ausführungszeit (execution time) bekannt. Die Ausführungszeit ist selbst wenn keine Unterbrechungen stattfinden nicht konstant, sondern hängt z. B. von den vorliegenden Eingabedaten ab. Sie kann ebenfalls, je nach Systemarchitektur, auch vom Zustand des Gesamtsystems wie z. B. dem Cacheinhalt oder anstehenden DMA-Transfers abhängen. Hieraus ergibt sich die Notwendigkeit zur weiteren Charakterisierung eines Tasks mit Hilfe seiner durchschnittlichen, minimalen und maximalen Ausführungszeit. In Tabelle 2.1 sind die verschiedenen Zeiten, mit denen ein Task üblicherweise charakterisiert wird aufgeführt [Zöbel 1995, S. 22 ff], [Li 1999, S 6 ff].

[2] steht hier für einen festen Zeitpunkt, nicht einen Zeitraum

Tab. 2.1: Zeitliche Parameter von Tasks

Parameter	**Beschreibung**
r_i – Release Time	frühster Zeitpunkt an dem mit der Ausführung von Task T_i begonnen werden kann
d_i – Deadline	Zeitpunkt an dem T_i spätestens beendet sein muss
s_i – Starting Time	Zeitpunkt des Starts der Ausführung von T_i
c_i – Completion Time	Zeitpunkt des Beendens der Ausführung von T_i
Δe_i – Execution Time	Dauer der Ausführung von T_i
Δp_i – Period	Zeit nach der sich die release time eines periodischen Tasks wiederholt
ET_{MIN_i} – Best Case Execution Time (BCET)	minimal nötige Zeit zur Ausführung eines bestimmten Tasks T_i
ET_{AVG_i} – Average Execution Time	durchschnittlich erwartete Ausführungszeit dieses Tasks
ET_{MAX_i} – Worst Case Execution Time (WCET)	maximal erwartete Zeit für die Ausführung dieses Tasks

2.2.2 Bestimmung von Ausführungszeiten

Um einen Plan zur Ausführung der zu verarbeitenden Tasks so zu gestallten, dass die Zeitbedingungen der Tasks berücksichtigt werden können, sind Informationen über die Ausführungszeiten der Tasks wichtig. Der WCET kommt dabei besondere Bedeutung zu, da sie, um die Einhaltung einer Deadline auch in den ungünstigsten Fällen und unter der Bedingung der vollständigen Ausführung eines Tasks zu gewährleisten, als Planungsgrundlage herangezogen werden muss.

Für die Bestimmung der Ausführungszeit eines Tasks gibt es verschiedene Ansätze. Eine Methode hierfür ist die Analyse des Programmcodes noch vor der Ausführung. Aufgrund eines (vereinfachten) Modells des Rechensystems kann die Ausführungszeit jeder einzelnen Instruktion in ihrem jeweiligen Kontext ermittelt werden. Die Summe dieser Zeiten plus der Zeiten für alle, während der Ausführung möglicherweise auftretenden Unterbrechungen ergibt die Ausführungszeit des Tasks. Dieses Verfahren stößt jedoch recht schnell an seine Grenzen, insbesondere wenn komplexe Programme analysiert werden müssen. Der Grund dafür ist, dass bereits in einfachen Befehlssequenzen verschiedene Programmpfade möglich werden, sobald auch nur einfache, bedingte Sprünge enthalten sind. Oft kommt es auch zu Abhängigkeiten von den Eingabedaten, z. B. innerhalb von Schleifen. Die Verarbeitung von komplexen

Datenstrukturen kann schnell zu einer sehr großen Anzahl von verschiedenen Ausführungspfaden und damit auch von möglichen Ausführungszeiten führen. Die Simulation aller möglichen Pfade, um die WCET zu bestimmen ist bei komplexen Programmen nicht mehr praktikabel. Bei der Berücksichtigung von Hardwarefunktionen wie z. B. Cache, die die mittlere Ausführungszeit verkürzen helfen sollen, ergibt sich eine noch größere Vielfalt an möglichen Ausführungszeiten. Eine Lösung für dieses Problem ist die Einführung von Restriktionen hinsichtlich der Komplexität des Programms schon bei der Programmierung. Ein anderer Ansatz verfolgt die Auswahl von Programmpfaden, für die es wahrscheinlich ist, dass sie zur WCET bzw. BCET führen [Li, Malik 1999, S. 29 ff]. Der Nachteil hierbei ist, dass diese Zeiten dann nur noch näherungsweise bestimmt werden können.

Ein anderer Ansatz zur Bestimmung der Ausführungszeit eines Tasks ist die Überwachung seines zeitlichen Verhaltens während der Laufzeit [Gergeleit et al. 1999]. Bei diesem Ansatz steht dem Scheduler eine Monitoring-Komponente zu Seite. Diese führt Statistiken, welche die Ausführungszeiten der Tasks charakterisieren. Aus der Verteilung der Ausführungszeiten kann diejenige Zeit ermittelt werden, bis zu der eine Tasks-Instanz mit einer bestimmten Wahrscheinlichkeit ihre Ausführung beenden haben wird. Diese so genannte Expected Case Execution Time (ECET) kann nun zur Planung herangezogen werden. Der Nachteil, dass die WCET damit nicht ermittelt werden kann wird durch die Verbindung mit TaskPair-Scheduling[3] aufgehoben.

Die Realisierung der Überwachung des zeitlichen Verhaltens von Tasks zur Laufzeit kann z. B. durch eine in regelmäßigen Zeitintervallen erfolgende Aufzeichnung des Systemzustands (Sampling) geschehen [Gergeleit 2001, S. 16 ff]. Alternativ ist außerdem ein ereignisbasierter Ansatz denkbar. Hierzu muss der Programmablauf mit Hilfe so genannter Sensoren so erweitert werden, dass an bestimmten Stellen Informationen erfasst und der Monitoring-Komponente zur Verfügung gestellt werden. Ein solches Erweitern wird als Instrumentation bezeichnet. Konkret wird an diesen Stellen zusätzlicher Code eingefügt, der dann Ereignisse auslöst, die, mit den Zeitpunkten ihres Auftretens versehen, ein Bild von den im System stattfindenden Abläufen liefern. Stellen, an denen Ereignisse z. B. ausgelöst werden können, sind Start- und Endpunkte von Tasks.

Die Instrumentation kann auf verschiedenen Softwareebenen stattfinden [Gergeleit 2001, S. 18 ff]. Im Wesentlichen handelt es sich hierbei um die Ebene der Systemsoftware und die der Anwendungssoftware. Beide Lösungen haben jeweils eigene Vorteile. Falls Ereignisse nur innerhalb der Systemsoftware ausgelöst werden,

[3] siehe Abschnitt 2.2.4.2

muss die Anwendungssoftware nicht erweitert werden, was während der Entwicklung ein Vorteil sein kann. Die Instrumentation auf der Ebene der Anwendungssoftware bietet dafür das Potential für eine detailliertere Überwachung. Um dabei den Nachteil des manuellen Einfügens von Überwachungsroutinen zu relativieren ist beispielsweise der Einsatz einer Spracherweiterung in Verbindung mit einem speziellen Präprozessor praktikabel [Gergeleit et al. 1999].

2.2.3 Scheduling

Beim Scheduling handelt es sich um die zeitliche Planung von Ressourcenzuweisungen an Tasks. Die zentrale Ressource ist hierbei der Hauptprozessor. Wie bei herkömmlichen Multitasking Systemen, so werden auch bei einem RTS die auszuführenden Tasks von einem oder mehreren Prozessoren abgearbeitet. Da für den Gegenstand dieser Arbeit nur die Betrachtung von Einprozessorsystemen eine Rolle spielt, beschränken sich die folgenden Ausführungen auch auf dieses Szenario.

Besonders die für Tasks innerhalb eines RTS zu berücksichtigenden Zeitbedingungen erfordern bei der Verteilung der Prozessorzeit einen entsprechend gestalteten Plan. Der Sinn des Plans ist es, Werte für Starting und Completion Time jedes Tasks aus der Menge der auszuführenden Tasks sowie ggf. Unterbrechungsintervalle zu liefern. Als Grundlage der Planung kommen Release Time bzw. Periode der einzelnen Tasks sowie deren jeweilige Ausführungszeit[4] und Deadline in Frage. Die Erstellung eines solchen Plans ist der Gegenstand des RT-Schedulings (Real-Time-Scheduling).

Es sind bei der Echtzeitplanung eine Reihe von verschieden Vorgehensweisen möglich. Hierbei sind statische und dynamische Scheduling-Verfahren zu unterscheiden. Bei statischen Verfahren kann die Planung offline, also vor der Programmausführung erfolgen. Es müssen hierfür in der Regel alle für die Planung wichtigen Parameter der einzelnen Tasks, insbesondere Bereitzeit oder Periode sowie WCET bekannt sein. Auf dieser Basis kann bereits im Vorfeld entschieden werden, ob sich überhaupt ein Plan ableiten lässt, der alle Zeitbedingungen erfüllt, und wenn ja, wie dieser aussieht. In dem Fall, dass zur Planung notwendige Daten, wie z. B. die Release Time, erst zur Laufzeit zur Verfügung stehen, kann die Planung auch erst zur Laufzeit durchgeführt werden. Es ist dann allerdings zur Entwicklungszeit nicht mehr garantierbar, dass später auch immer alle Task-Instanzen ausgeführt werden können, ohne ihre Zeitbedingungen zu verletzen. Eine dynamische Planung bietet dafür den Vorteil, während der Laufzeit Informationen über die Ausführungszeiten der Tasks gewinnen zu können und diese

[4] hier normalerweise die WCETs

anstatt der offline ermittelten WCET für die Planung zu nutzen. Falls für eine Anwendung das strikte Einhalten von Zeitbedingungen von Tasks erforderlich ist spricht man von harten Zeitbedingungen (hard real-time). Falls dies nicht erforderlich ist und von dem System zu erwarten ist, dass Zeitbedingungen entsprechend einer statistischen Verteilung verletzt werden, spricht man auch von weichen Zeitbedingungen (soft real-time). Eine Vermischung von beiden Typen findet man in der form von firm oder weakly-hard real-time.

Im Folgenden sollen nun exemplarisch zwei klassische Scheduling-Verfahren genannt und kurz vorgestellt werden. Das wäre zum einen *Planen nach monotonen Raten* (Rate Monotonic – RM) [Liu, Layland 1973]. Dieses Verfahren ist auf periodische Tasks beschränkt. Es wird jedem Task in Abhängigkeit von seiner Periodenlänge eine Priorität fest zugewiesen. Der Task mit der kleinsten Periodenlänge erhält die höchste Priorität. Sobald ein Task lauffähig wird und eine höhere Priorität hat als der aktuell laufende Task, wird dieser verdrängt und der neue gestartet. Durch Anwendung dieses Schemas ergibt sich ein impliziter Plan. Das Verfahren ist optimal für alle Probleme mit statischen Prioritäten, d. h. falls eine Menge von Tasks planbar ist, ist sie auch mit diesem Verfahren planbar.

Ein weiters Verfahren ist *Planen nach Fristen* (Earliest Deadline First – EDF) [Liu, Layland 1973]. Im Gegensatz zu RM ist dieses nicht auf periodische Tasks beschränkt. Die Prioritäten werden hier nicht statisch zugewiesen sondern jeder Task bzw. jede Instanz eines Tasks erhält abhängig von der aktuellen Deadline seine Priorität dynamisch zugeteilt. Dabei bekommt der Task mit der am nächsten liegenden Deadline die höchste Priorität und verdrängt alle anderen Tasks. Das Verfahren kann die Einhaltung von Deadlines für eine Menge von Tasks garantieren sofern diese überhaupt planbar sind.

2.2.4 Fehlerbehandlung

Während der Laufzeit können vor allem dynamische Scheduler auf Probleme stoßen, die dazu führen, dass Zeitbedingungen verletzt werden können. Ein Beispiel für eine solche Situation ist das tatsächliche Auftreten einer Ausführungszeit eines Tasks, die die für seine Einplanung benutzte Zeit übersteigt. Ein anderes Beispiel ist eine Überlastsituation bei Verwendung eines dynamischen Schedulers, in der einfach nicht alle Prozesse einplanbar sind. In Überlastsituationen kann in dem Fall, dass der Scheduler einen Akzeptanztest vor der Einplanung eines Tasks durchführt, ein Task abgewiesen werden. Auf seine Ausführung muss dann verzichtet werden. Wenn ein

solcher Test nicht stattfindet, kann es prinzipiell zu Zeitüberschreitungen kommen. Das kann, z. B. bei einem EDF-Scheduler, dazu führen, dass auch die Zeitbedingungen der folgenden Instanzen nicht mehr eingehalten werden können. Für das Vermeiden solchen Verhaltens wurden in der Vergangenheit bereits verschiedene Ansätze entwickelt.

2.2.4.1 Verringerung der Qualität von Berechnungen

Bei der Entwicklung eines Echtzeitsystems können in bestimmten Situationen Kompromisse, was die Einhaltung von Zeitbedingungen oder die gebotene Funktionalität angeht, gemacht werden. Das Eintauschen von Qualität in der Berechnung von Ergebnissen gegen eine zeitlich kürzere Ausführungszeit dieser Berechung kann eingesetzt werden, um Zeitüberschreitungen von Task-Instanzen zu minimieren oder zu verhindern. Man spricht hier auch oft von Quality of Service (QoS). [Chung, Liu, Lin 1990] schlagen die Aufteilung eines Tasks in notwendigen und optionalen Teil vor. Nach Beendigung des notwendigen Teils kann ein Task jederzeit abgebrochen werden, da der notwendige Teil bereits ein für die Anwendung akzeptables Ergebnis geliefert hat, welches durch den optionalen Teil nur noch weiter verfeinert wird.

Eine weitere Möglichkeit besteht im Auslassen von Task-Instanzen mittels eines Akzeptanztests. Dies kann ebenfalls nur angewendet werden wenn dies für die Anwendung akzeptabel ist. Das Auslassen von Instanzen muss also in dem Anwendungsfall möglich sein und darf nicht zu einem Versagen des Systems führen. Bei vielen Signalverarbeitungsaufgaben ist dies der Fall [Nawab et al. 1997]. Ein Beispiel hierfür ist das Auslassen einzelner Datenelemente (Samples, Frames) bei der Verarbeitung von Multimediadaten. Eine Verringerung der Qualität der durch das System bearbeiteten Aufgabe tritt auch bei dieser Methode auf und muss in Kauf genommen werden können.

2.2.4.2 TaskPair-Scheduling und TAFT

Ein weiterer Ansatz zur Behandlung von zeitweisen Überlastsituationen ist das TaskPair-Scheduling [Streich 1994]. Auch hier besteht ein Task aus zwei Teilen. Ein langwieriger Task, in dem Fall der so genannte Main-Task, kann vorzeitig abgebrochen werden wenn durch ihn eine Zeitüberschreitung befürchtet wird. Ein anschließend auszuführender, kurzer Except-Task übernimmt in eingeschränkter Form seine Aufgabe.

Die WCET des Except-Tasks muss bekannt sein. Durch das Verfahren wird die Einhaltung einer Deadline durch das Task-Paar garantiert.

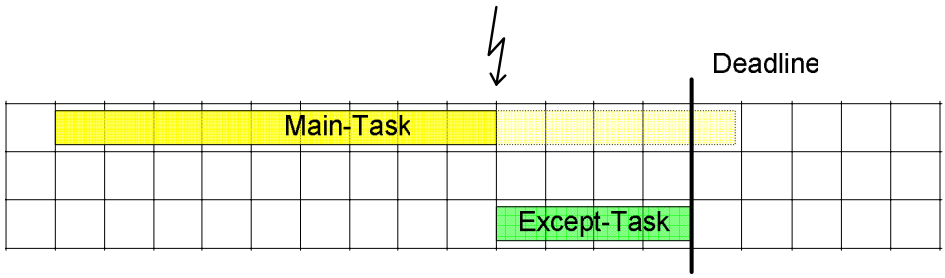


Abb. 2.3: TaskPair-Scheduling

Dieses System ist beispielsweise zusammen mit iterativen Tasks einsetzbar [Nett et al. 1998] welche Berechnungen so durchführen, dass die Ergebnisse bei vorzeitigem Abbruch zwar nicht optimal aber brauchbar sind und so vom Except-Task verwertet werden können. Auch hier verringert sich die Qualität des angebotenen Services.

Anwendung fand das Konzept des TaskPair-Schedulings in Form des TAFT-Schedulings (Time Aware Fault Tolerant) [Becker, Gergeleit 2001]. Dieser Ansatz verwendet eine, über eine Monitoring-Komponente bestimmte, geschätzte Ausführungszeit zur Planung und benutzt das TaskPair-Konzept zur Vermeidung von, durch falsche Schätzungen hervorgerufenen, Zeitüberschreitungen. Es ist eine RT-Linux basierte Implementierung des TAFT-Schedulers verfügbar.

2.2.5 Prozesssynchronisation

Synchronisation von Abläufen ist wie auch bei nicht echtzeitfähigen Multitaskingsystemen im Sinne der Aufgabenstellung oftmals notwendig, um einen Task zu verzögern oder einen verzögerten Task fortzusetzen [Zöbel, Albrecht 1995, S. 141 ff]. Üblicherweise tritt eine Situation in der solche Verzögerungen notwendig sind beim Zugriff mehrerer Tasks auf gemeinsame Ressourcen auf. Ein Beispiel hierfür ist das Erzeuger-Verbraucher Problem: Ein Task erzeugt Daten, die an einen anderen Task z. B. mittels eines Ringpuffers, übergeben werden sollen. Für die Synchronisation des Zugriffs auf diese Ressource werden in der Regel Semaphoren eingesetzt. Insbesondere statische Scheduler müssen einen gegenseitigen Ausschluss von Tasks bei der Planung berücksichtigen.

Ein weiteres Problem, was insbesondere bei Echtzeitsystemen relevant ist, ist das Phänomen der Prioritätsumkehr. Dazu kommt es wenn ein Task T_1 mit hoher Priorität auf eine Ressource zugreifen will, die von einem Task T_2 mit niedrigerer Priorität gehalten wird. In dem Fall ist es für Task T_1 nicht möglich den Task T_2 mit niedrigerer Priorität zu verdrängen. Zwischenzeitlich können aber andere Tasks mit einer Priorität, die höher ist als die von T_2 diesen verdrängen. Dadurch wird effektiv die Priorität von T_1 auf die von T_2 reduziert. Eine Lösung hierfür besteht in der Anwendung von Prioritätsvererbung [Sha et al. 1990] beim Zugriff auf gemeinsame Ressourcen.

2.3 Echtzeitbetriebssysteme

2.3.1 Aufgaben und Nutzen

Betriebsysteme sind ein wesentliches Hilfsmittel nicht nur bei der Programmierung von Desktop-Systemen sondern auch bei der von eingebetteten Systemen geworden. Sie erfüllen dabei keine Aufgabe im Sinn einer Anwendung sondern stellen anderen Programmen bestimmte Dienste bereit. Zu den wichtigsten Funktionen, die Betriebssysteme in der Regel haben, gehören Prozessverwaltung, Prozesssynchronisation, Interprozesskommunikation, Speicher- und Dateiverwaltung sowie die Verwaltung weiterer Geräte. Ziel hierbei ist es, den Anwendungsprogrammen mittels geeigneter Schnittstellen geordnet Zugang zu Betriebsmitteln, wie z.B. Hauptprozessor oder Arbeitsspeicher, zu geben. Aus diesen Aufgaben leiten sich auch gängige Definitionen des Begriffs ab. DIN 44300 definiert ein Betriebssystem beispielsweise als „Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.“

Für den besonderen Fall eines Echtzeitsystems werden natürlich auch an das Betriebssystem Anforderungen an die Einhaltung von Zeitbedingungen gestellt. Bei der Verwaltung von Prozessen, die Real-Time Tasks implementieren, kommt dadurch die Forderung nach Vorhersagbarkeit des zeitlichen Verhaltens dieser Prozesse hinzu [Zöbel, Albrecht 1995, S. 119 ff]. Ebenso sollte das Betriebssystem festgelegte Antwortzeiten bei der Verwaltung anderer Betriebsmittel einhalten. Wie bei nicht echtzeitfähigen Systemen auch, besteht eine allgemeine Forderung in der sparsamen Verwendung von Systemressourcen wie Arbeitsspeicher oder Prozessorzeit für Zwecke des Betriebssystems. Dies gilt insbesondere für Betriebssysteme, die für DSP-

basierende Systeme vorgesehen sind, da hier in der Regel nur wenig Speicher zur Verfügung steht und allein die auszuführende Anwendung das System stark auslastet.

Es ist prinzipiell möglich, Anwendungen auch ohne Unterstützung eines Betriebssystems zu verwenden. Besonders bei überschaubaren, eingebetteten Systemen wird dies auch immer noch praktiziert, z. B. durch die Verwendung eines Foreground/Background-Systems [Labrosse 2002, S. 36 ff]. Ein Betriebssystem bietet jedoch auch hier Vorteile, beispielsweise durch das zur Verfügung stellen von Hardwaretreibern oder die Möglichkeit zur Verwendung eines Task-Konzepts, mit dem die Aufgaben einer Anwendung in kleinere Teilaufgaben zerlegt werden können.

2.3.2 Einbindung von Anwendungssoftware

Im Folgenden soll auf zwei Aspekte der Implementierung von Echtzeitbetriebssystemen eingegangen werden. Das ist zum einen die Art ihrer Auslegung als eigenständige Software oder als Bibliothek. Zum anderen ist das die Programmierschnittstelle zum Anwendungsprogramm.

Was ihre Auslegung angeht, so kann hier zwischen verschiedene Formen differenziert werden. Ein gängiger Weg ist es, ein komplettes RTOS so zu spezifizieren und zu implementieren, das den notwendigen Anforderungen an einen Echtzeitbetrieb entspricht. Dies ist v. a. bei Systemen für eingebettete Anwendungen der Fall. Beispiele sind hier QNX [Hildebrand 1992], VxWorks [Windriver 2003] und LynxOS [Lynuxworks 2003]. Eine andere Möglichkeit besteht darin, ein bestehendes System um Echtzeitfähigkeiten zu erweitern. Dies kann auf verschiedene Arten geschehen, zum einen über Anpassungen des vorhandenen Kerns oder über die Ergänzung mit einem zusätzlichen RT-Scheduler. In letzteren Fall läuft dann der bisherige, nicht echtzeitfähige Kern inklusive aller seiner Prozesse als ein Prozess des zusätzlichen RT-Systems. Ein Beispiel für die Umsetzung dieses Ansatzes ist RT-Linux [Yodaiken 1997].

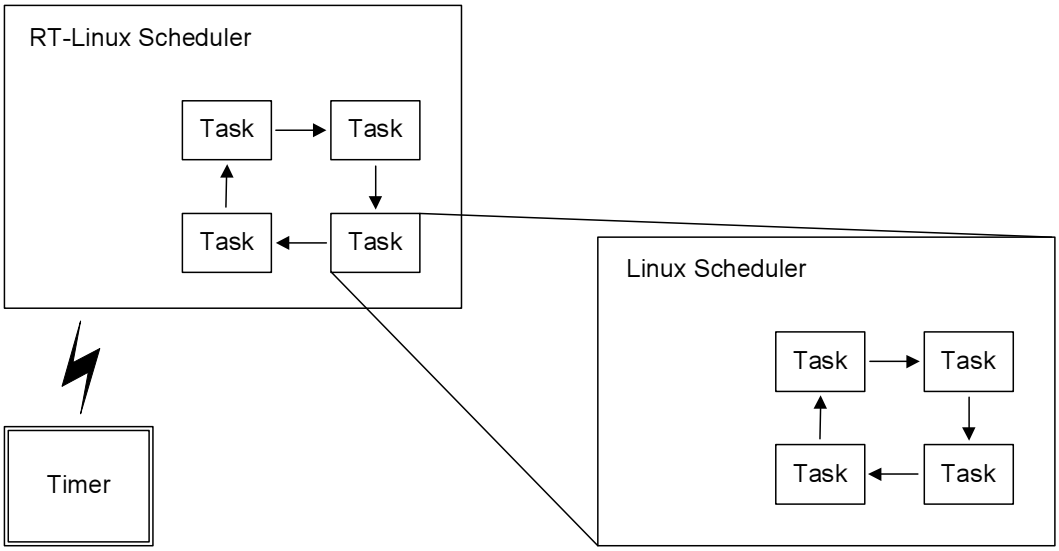


Abb. 2.4: Prinzip von RT-Linux

Bei der Einbindung des Anwendungsprogramms gibt es ebenfalls unterschiedliche Ansätze. Falls das RTOS das unterstützt, kann die Anwendung in Form von einem oder mehreren Modulen zur Laufzeit, z. B. beim Start, geladen und ausgeführt werden. Eine andere Möglichkeit ist, das Betriebssystem als Laufzeitumgebung (real-time executive) [Zöbel, Albrecht 1995, S. 123] bereits während der Entwicklung mit in die Anwendung zu integrieren. Dieses Verfahren bietet sich v. a. bei eingebetteten Systemen an.

Einen anderen Aspekt bei dem Zusammenspiel mit der Anwendungssoftware stellt die von der Systemsoftware hierfür zur Verfügung gestellte Programmierschnittstelle dar. Systeme wie QNX bedienen sich hierfür der Verwendung des POSIX Standards [IEEE 1998]. Dieser Standard definiert grundlegende Eigenschaften und Schnittstellen von Unix-ähnlichen Systemen sowie Erweiterungen für den Echtzeitbetrieb und gilt als allgemein akzeptiert. Durch die Verwendung eines POSIX-kompatiblen Systems kann somit eine gewisse Portabilität der Anwendung auf Quelltextebene erreicht werden. [Moses 1995] allerdings kritisiert die Verwendung von POSIX im Bereich von kleinen, eingebetteten Systemen. Als Nachteil wird der angewachsene Umfang des Standards sowie seine Konzentration auf Funktionen für Unix-Desktopsysteme genannt. Eine proprietäre Schnittstelle kann hier Vorteile bieten. Es sind Systeme, die sich auch für DSP-Anwendungen eignen verfügbar, die ihre Funktionen mit Hilfe eigener Aufrufe zur Verfügung stellen. Ein Beispiel hierfür ist µC/OS-II [Labrosse 2002].

2.3.3 Beispiele für Scheduling-Verfahren in RTOSs

Gängige Echtzeitbetriebssysteme bieten oft die Scheduling-Verfahren Round-Robin (RR) sowie Fixed Priority Preemptive an [Navet, Migge 2003]. Auch der POSIX Standard IEEE 1003.1b definiert neben Funktionen für Zeitmessung, Timer, Nachrichten, Semaphoren und asynchroner Ein- und Ausgabe auch die Modi "sched_rr" und "sched_fifo" für das einplanen von zeitkritischen Prozessen an. Im Gegensatz zu normalen Prozessen bieten Prozesse, die in einem der beiden Modi ausgeführt werden eine bessere Vorhersagbarkeit ihres Verhaltens. Bei beiden Verfahren können den Prozessen statische Prioritäten zugewiesen werden. Im Fall von "sched_fifo" verdrängen dabei grundsätzlich diejenigen Prozesse mit höherer die mit niedrigerer Priorität. Falls zwei Prozesse mit gleicher Priorität gleichzeitig lauffähig sind, wird der derjenige ausgeführt, der als erster lauffähig war. Unter der Annahme, dass alle Prozesse eine unterschiedliche Priorität besitzen, entspricht dieses Verfahren Fixed Priority Preemptive. Im Gegensatz dazu können bei "sched_rr" Prozesse mit gleicher Priorität sich zusätzlich gegenseitig verdrängen. Dies geschieht nach dem Zeitscheibenverfahren.

Unter RT-Linux steht u. a. ebenfalls ein Scheduler, der auf statischen Prioritäten basiert zur Verfügung. Dieser arbeitet prinzipiell wie bei "sched_fifo" oben beschrieben. Aufgrund des modularen Designs von RT-Linux sind aber auch andere Scheduling-Verfahren wie EDF und RM implementiert wurden [Yodaiken 1997].

2.3.4 Das Betriebsystem MicroC/OS-II

Bei MicroC/OS-II handelt es sich um ein Echtzeitbetriebssystem, dass auf verschiedenen Anwendungsgebieten eingesetzt werden kann. Dieses spezielle System wurde auch im Rahmen dieser Arbeit verwendet und wird aus diesem Grund hier näher vorgestellt.

2.3.4.1 Allgemeines

Das Micro-Controller Operating System ist in Version 2 für eine breite Palette von Systemen verfügbar. Es liegt auch einer Version für den ADSP-21065L vor, welcher weitgehend kompatibel zum verwendeten DSP ist. Das System wurde größtenteils in ANSI-C geschrieben. Einige hardwarenahe Funktionen sind in Assembler geschrieben. Durch das Vorliegen des vollständigen Quelltextes ist es leicht anpassbar bzw. erweiterbar. Das System bietet ein übersichtliches, proprietäres API (Application

Programming Interface) und kann in Form einer Bibliothek zu der Anwendung hinzugefügt werden. Es ist darüber hinaus kompakt und findet komplett im Speicher des eingesetzten Systems Platz. Nahezu alle Funktionen oder Funktionsgruppen lassen sich außerdem über das Konfigurieren einer zentralen Header-Datei deaktivieren um weiteren Speicherplatz zu sparen.

2.3.4.2 Funktionalität

MicroC/OS-II bietet Dienste aus den Bereichen Prozessverwaltung und Scheduling, Prozesssynchronisation, Interprozesskommunikation sowie Speicherverwaltung. Darüber hinaus sind rudimentäre Funktionen für das Sammeln von Daten über die Prozessorauslastung enthalten.

Das System unterstützt bis zu 64 leichtgewichtige Prozesse oder auch Threads, die allgemein als Tasks bezeichnet werden. Die von der Anwendung maximal benötigte Anzahl Tasks kann durch Definieren einer Konstante in einer Header-Datei vor dem Übersetzen des Systems festgelegt werden. Dies ist sinnvoll, da Speicher für die benötigten Verwaltungsinformationen statisch belegt wird. Im Gegensatz dazu kann die Größe des Stacks jedes Tasks einzeln und beim Anlegen des Tasks, also während der Laufzeit, festgelegt werden.

Das Scheduling von Tasks in µC/OS-II arbeitet preemptiv auf Basis von festen Prioritäten. Jedem Task kann beim Start eine individuelle Priorität zugewiesen werden. Da kein FIFO (First In First Out) oder Round-Robin Mechanismus unterstützt wird, dürfen keine zwei oder mehr Tasks dieselbe Priorität haben. Die niedrigste Priorität hat der systeminterne Idle-Task. Sobald ein Task mit einer höheren Priorität als der aktuelle lauffähig wird, wird der aktive Task von dem mit höherer Priorität verdrängt. Ein Task läuft also so lange, bis er entweder von einem anderen Task mit höherer Priorität verdrängt wird oder selbst in einen Wartezustand übergeht, wobei dann der Task nächst niedrigerer Priorität fortgesetzt wird. Ein Task kann in einen Wartezustand übergehen indem er eine entsprechende Systemfunktion aufruft, die seine Ausführung für einen bestimmten Zeitraum verzögert, oder indem er auf eines der im Folgenden noch näher beschriebenen Ereignisse wartet.

Im Rahmen der Prozesssynchronisation bietet das System drei verschiedene Objekttypen an. Es handelt sich dabei um Semaphoren, Mutexes und Event Flags. Tasks können zum einen auf die Signalisierung eines der Objekte warten sowie andererseits die Signalisierung eines Objekts auslösen. Die Anzahl der von der Anwendung maximal benutzbaren Objekte eines Typs wird wiederum in einer Header-Datei fest vorgegeben.

Semaphoren sind mit einem Zähler versehen, der von der Anwendung inkrementiert oder dekrementiert werden kann. Ein Semaphor wird signalisiert solange dieser Zähler größer Null ist. Mutexes sind binäre Semaphoren. Bei einer Flag Group handelt es sich im Wesentlichen um eine Reihe von Mutexes. Die Applikation kann auf das auftreten einer bestimmten Kombination von gesetzten bzw. nicht gesetzten Flags warten. Beim Umgang mit Metexes unterstützt das System Prioritätsvererbung um das Problem der Prioritätenumkehr zu berücksichtigen.

Für die Interprozesskommunikation stehen zwei Mechanismen bereit. Dabei handelt es sich um die Mailbox und um Nachrichtenschlangen (Message Queues). Eine Mailbox kann benutzt werden um eine Nachricht in Form einer Zeigervariablen zu übergeben. Ein Task kann dafür auf das eintreffen einer Nachricht in der Mailbox warten. Eine Nachrichtenschlage ist ähnlich, bietet jedoch die Möglichkeit mehrere Zeigervariabeln zu übergeben. Dieser werden dabei in einem Ringpuffer gespeichert bis sie wieder aus der Schlange entfernt werden.

Das Speichermanagement von µC/OS-II ist sehr rudimentär. Das System unterstützt keine ladbaren Programmmodule und auch kein Paging oder Swapping. Die einzige Funktion der Speicherverwaltung besteht darin, Speicherblöcke fester Größe auf Anfrage bereitzustellen. Dabei ist die Anzahl der maximal zur Verfügung stehenden Speicherblöcke bereits vorher festgelegt worden.

2.3.4.3 Erweiterungen

Da das Betriebssystem µC/OS-II kompakt, übersicht strukturiert und im Quelltext verfügbar ist, wurden davon ausgehend bereits Erweiterungen vorgenommen. Im Zusammenhang mit dieser Arbeit ist das SharcOS Projekt [JDC 2004] interessant. Hierbei handelt es sich um eine kommerzielle Erweiterung des Systems, welche speziell auf die Prozessoren der SHARC-Familie abzielt. SharcOS bietet vor allem Funktionen für Interruptmanagement an, welche dem Betriebssystem normalerweise fehlen. Es führt dabei das Konzept des SuperTasks ein, wobei es sich um einen besonders schnellen Interrupthandler handelt. Ein wesentlicher Nachteil des Systems ist sein gegenüber dem herkömmlichen µC/OS-II weiter gestiegener Ressourcenbedarf. Es wird insbesondere eine größere Menge Arbeitsspeicher benötigt.

2.4 Verteilte Systeme

Wie auch andere komplexe Anwendungssysteme ist die in dieser Arbeit untersuchte Steuerung eines Rasterkraftmikroskops nicht auf einen Rechner beschränkt, sondern erstreckt sich über mehrere Rechnerknoten[5], die in der Lage sind autonom zu arbeiten. Zusammen erfüllen die Knoten eine oder mehrere Aufgaben. Hierbei handelt es sich um ein verteiltes System. Um die Programmier- bzw. Handhabbarkeit eines solchen Systems zu verbessern, kann es sinnvoll sein, die Verteiltheit des Systems vor dem Benutzer bzw. dem Anwendungsprogrammierer zu verbergen oder Mittel zu deren besserer Handhabbarkeit bereitzustellen. Dieses Ziel kann z. B. durch den Einsatz entsprechender verteilter Betriebssysteme erreicht werden.

2.4.1 Eigenschaften und Anwendungen

Bedingt durch den Aufbau eines verteilten Systems aus unabhängigen Rechnerknoten ergeben sich dessen wesentliche Eigenschaften: Es existieren parallele Aktivitäten, die Betriebsmittel zur Erfüllung einer Gesamtaufgabe sind über mehrere Knoten verteilt und der Datenaustausch erfolgt über ein Kommunikationsmedium. Weiterhin können solche Systeme heterogen sein, also aus nicht gleichartigen bzw. spezialisierten Knoten bestehen. Ebenso kann die Anzahl von Knoten sehr unterschiedlich sein. Bei einer großen Anzahl von Knoten steigt die Wahrscheinlichkeit eines Ausfalls eines beliebigen Knotens des Systems an. Eine wesentliche Forderung ist daher oft, dass das Gesamtsystem im Fehlerfall zumindest solche Aufgaben weiter erfüllt, die nicht direkt von ausgefallenen Knoten abhängen. Im Rahmen einer Anpassung an unterschiedliche Aufgaben können in einem verteilten System Knoten entfernt oder neue Knoten hinzugefügt werden. Hieraus folgt die Forderung nach klar definierten, möglichst offenen Schnittstellen um neue Knoten einfach integrieren zu können.

Aufgrund dieser Eigenschaften von verteilten Systemen lassen die wesentlichen Ziele ihrer Anwendung erkennen. Es handelt sich hierbei vor allem um die Erhöhung der Fehlertoleranz, eine leichte Skalierbarkeit der Rechenleistung sowie Möglichkeiten der flexiblen Anpassung an unterschiedliche Anwendungen.

[5] siehe Kapitel 2.3 für einen Überblick über den genauen Aufbau

2.4.2 Verteilte Betriebssysteme

Zum Zweck der leichteren Programmierbarkeit von verteilten Systemen bietet es sich an, dass bereits die Systemsoftware der einzelnen Knoten Mittel für deren Integration in ein Gesamtsystem zur Verfügung stellt. Dabei handelt es sich um eine Kommunikationssoftware, die Verbindungen mit anderen Knoten herstellt und die die Grundlage für die Interaktion von Prozessen unterschiedlicher Knoten bietet. Die Interaktion kann z. B. in Form eines Nachrichtenaustauschs zwischen zwei Prozessen stattfinden. Dabei sorgt das verteilte Betriebssystem dafür, dass es für die beteiligten Prozesse unerheblich ist, ob sie auf demselben Knoten laufen oder nicht. Es wird nur eine Schnittstelle für beide Fälle angeboten. Der Transport der Nachrichten zwischen entfernten Knoten erfolgt dabei für die beteiligten Prozesse transparent. Die hierfür notwendigen Softwarekomponenten können zum einen direkt in den Systemkern integriert worden sein. Ein Beispiel hierfür das Open-Source System Amoeba [vrije 2004]. Es kann aber auch ein herkömmliches Betriebssystem durch zusätzliche Software erweitert werden.

2.4.3 Kommunikation in verteilten Systemen

Besonderer Bedeutung kommt in jedem verteilten System der Kommunikation zwischen den einzelnen Rechnerknoten zu. Neben der bereits im vorangegangenen Abschnitt genannten Möglichkeit zum Nachrichtenaustausch zwischen Prozessen sind auch weitere Formen der Interaktion zwischen Knoten möglich, z. B. die zeitliche Koordination von Abläufen, das Signalisieren von Ereignissen, der gemeinsame Zugriff auf entfernte Ressourcen oder Möglichkeit zur Lastverteilung. Die Realisierung der genannten Funktionalität setzt das Vorhandensein eines physischen Übertragungskanals zwischen den Knoten sowie geeigneter Kommunikationsprotokolle und -software voraus. Auch die in einem verteilten System zur Kommunikation benutzten Hard- und Softwarekomponenten orientieren sich normalerweise an dem ISO/OSI-Referenzmodell [ISO 1984] oder lassen sich zumindest einer oder mehrerer Schichten dieses Modells zuordnen.

Die Knoten eines verteilten Systems können auf verschiedene Art und Weise miteinander verbunden sein. Hier sind zum einen die eingesetzte Hardwaremedien und Protokolle als auch die durch die Knoten und deren Verbindungen untereinander entstehende Topologien von Interesse. Auf der Hardwareseite können Kommunikationsverbindungen z. B. mittels (switched-) Ethernet, WLAN nach IEEE 802.11b, Token Ring oder SCI (Scalable Coherent Interface) aufgebaut werden. Bei verteilten Echtzeitsystemen werden aber auch oft spezielle Busse wie der CAN- oder

der Profibus des gleichnamigen Herstellers verwendet. Mit Hilfe von Kommunikationsverbindungen können die Knoten z. B. in ring- oder gitterförmigen Strukturen angeordnet werden. Es ist aber auch ein einfaches, busartiges Netz oder ein Netz mit schaltbaren Punkt-zu-Punkt Verbindungen möglich.

3 Gerätetechnische Grundlagen

In diesem Kapitel wird auf Grundlagen wichtiger Aspekte des Anwendungsbereichs, welcher den Rahmen dieser Arbeit bildet, eingegangen. Aufgrund des großen Umfangs des Themengebiets kann in diesem Kapitel nicht auf alle Details eingegangen werden. Es wird jedoch an mehreren Stellen auf entsprechende Literatur verwiesen.

3.1 Die Rastersondenmikroskopie

Im Folgenden soll ein kurzer Überblick über den Bereich der Rastersondenmikroskopie (Scanning Probe Microscopy) gegeben werden. Anschließend wird auf das Verfahren der Rasterkraftmikroskopie näher eingegangen. Es werden die physikalischen Hintergründe kurz angeschnitten sowie der prinzipielle Geräteaufbau erläutert. Ebenso werden Beispiele für Anwendungsgebiete gegeben.

Ein prinzipielles Verständnis des Mikroskopieverfahrens auf der einen, als auch einiger wesentlicher, physikalischer Grundlagen auf der anderen Seite ist für Planung der Steuerungssoftware unabdingbar. So können z. B. einige zeitkritische Tasks und v. a. auch deren zeitliche Parameter aus den verfahrenstechnischen Gegebenheiten abgeleitet werden[6].

3.1.1 Funktionsprinzip und Verfahrensweisen

Das erste Rastersondenmikroskop (scanning probe microscope) wurde bereits 1982 von Binnig und Rohrer [Binnig et al. 1982] vorgestellt. Es handelte sich dabei um ein Rastertunnelmikroskop (STM). Mit dem STM war es möglich eine Probenoberfläche mit atomarer Auflösung abzubilden. Im Jahre 1986 erhielten beide hierfür den Nobelpreis für Physik. In der Folgezeit wurden noch weitere Mikroskope entwickelt, die z. T. unterschiedliche physikalische Effekte nutzen aber alle dem prinzipiellen Verfahren der Rastersondenmikroskopie zuzuordnen sind. Die wesentlichen Verfahren sind in Tabelle 3.1 aufgelistet. Das allen Geräten zugrundeliegende Prinzip ist in Abbildung 3.1 dargestellt.

[6] siehe Kapitel 5 und 6

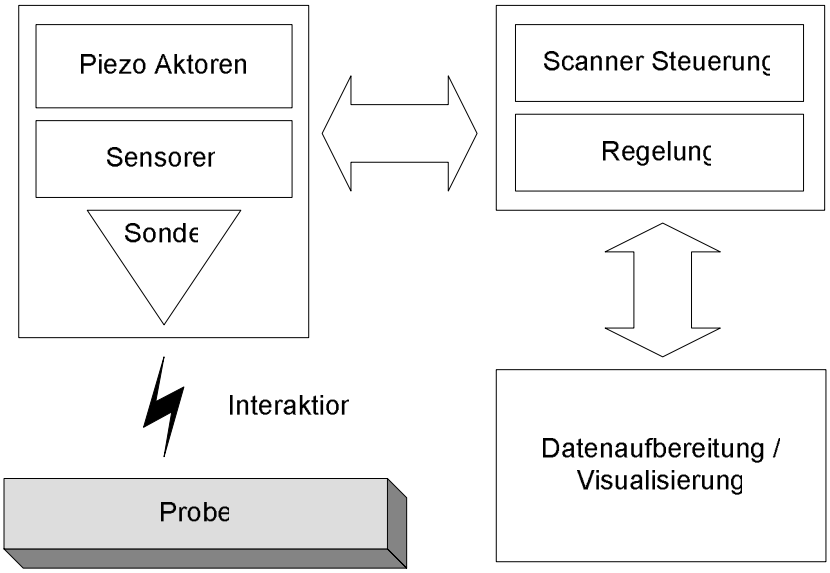


Abb. 3.1: Prinzip eines Rastersondenmikroskops

Bei allen Verfahren wird eine Sonde in Form einer mikroskopisch kleinen Spitze sehr nahe an die Probenoberfläche herangebracht. Der genaue Abstand ist abhängig vom konkreten Verfahren. In diesem Bereich kommt es zu einer Interaktion zwischen Sonde und Probe. Mit Hilfe von Piezo-Aktoren und einer entsprechenden Scanner Steuerung kann die Sonde in der xy-Ebene einen vorgegebenen Bereich der Probe zeilenweise abfahren. Die Größe dieses Bereichs kann sehr unterschiedlich sein. Aufgrund des Einsatzes von Piezokristallen als Aktoren, die sich durch ein von außen angelegtes, elektrisches Feldes mechanisch deformieren, ist es möglich, sehr kleine Bereiche einer Probe abzufahren. Es sind so je nach Geräteaufbau Scanbereiche mit Seitenlängen von kleiner 10nm bis größer 100µm möglich. Während dieses Vorgangs ist es möglich, eine Messgröße, welche die bei der Interaktion Sonde - Probe auftretenden physikalischen Effekte charakterisiert, zu messen, aufzuzeichnen und sie entsprechend visuell darzustellen. Zum anderen kann mit Hilfe weiterer Piezo-Aktoren die Sonde mit konstantem Abstand an der Probenoberfläche entlang geführt werden. Hierzu wird die Sonde bzw. die Probe mit Hilfe einer Regelung in z-Richtung verfahren. In diesem Fall wird die Ausgangsgröße des Reglers, also die für die Konstanthaltung des Abstands notwendige elektrische Spannung am Z-Piezo, für die visuelle Darstellung genutzt.

Das zuerst entwickelte STM nutzt den quantenmechanischen Tunneleffekt. Bei entsprechend geringer Annäherung der Sondenspitze an die Probenoberfläche fließt ein Tunnelstrom zwischen beiden, welcher gemessen werden kann. Dieser Strom hängt sehr stark vom Abstand des vordersten Atoms der Sondenspitze zum nächstliegenden Atom der Probe ab. Hieraus folgt nicht nur ein hohes Auflösungsvermögen des Mikroskops in z-Richtung sondern auch, dass benachbarte Atome der Sonde, die einen etwas größeren

Abstand zur Probe haben den Tunnelstrom kaum beeinflussen. Dieser Tatsache verdankt das Verfahren seine ebenfalls sehr hohe laterale Auflösung von weniger als einem Ångström. Das Verfahren ist auf elektrisch leitfähige Proben beschränkt. Es können einzelnen Atome in einem Kristallgitter gut sichtbar gemacht werden.

Beim Scanning Near field Optical Microscope (SNOM) [Pohl et al. 1984] bzw. Scanning Tunneling Optical Microscope (STOM) wird keine Siliziumspitze als Sonde eingesetzt sondern eine Glasfaser mit einer Lichtaustrittsöffnung die kleiner ist als die Wellenlänge des eingesetzten Lichts. Die Probenoberfläche wird mit dieser Glasfaser beleuchtet, das rückgestreute oder transmittierte Licht detektiert. Die Glasfaser befindet sich in sehr geringem Abstand zur Probenoberfläche, d. h. in einem Abstand der ca. einen Bruchteil der Wellenlänge des verwendeten Lichts entspricht. In diesem sog. Nahfeld begrenzen keine Beugungseffekte, wie bei herkömmlicher Lichtmikroskopie normalerweise üblich, die laterale Auflösung. Das SNOM arbeitet normalerweise mit konstantem Abstand Probe – Sonde. Um diesen Abstand konstant zu halten wird ähnlich wie beim dynamischen Modus des im Folgenden besprochenen Rasterkraftmikroskops (AFM), der Abstand zwischen Probe und Faser gemessen und die Spannung am Z-Piezo entsprechend geregelt. Mit einem SNOM erhält man so gleichzeitig ein Höhenprofil sowie eine optische Aufnahme der Probe. Die laterale Auflösung liegt bei 15-50 nm.

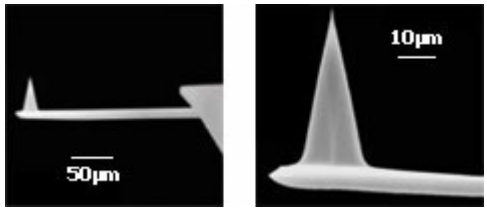
Tab. 3.1: Verschiedene Verfahren der Rastersondenmikroskopie

Verfahren	**Sonde**	**Wechselwirkung Probe - Sonde**
STM	Siliziumspitze	Tunnelstrom / quantenmechanischer Tunneleffekt
AFM	Siliziumspitze	Kraftwirkung zwischen Probe und Sondenspitze
SNOM	Glasfaserspitze	rückgestreutes bzw. transmittiertes Licht / optische Nahfeldeffekte
STOM	Glasfaserspitze	Tunnelphotonenstrom / optische Nahfeldeffekte

3.1.2 Die Rasterkraftmikroskopie

Die Rasterkraftmikroskopie wurde kurz nach dem STM ebenfalls von Binnig und Mitarbeitern [Binnig et al. 1986] entwickelt. Mittlerweile ist dieses Verfahren in verschiedenen Abwandlungen für diverse Anwendungsgebiete verfügbar. Bei der

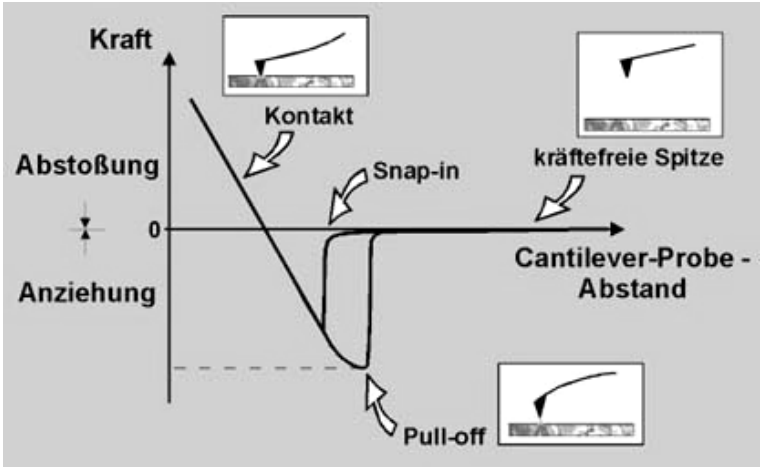
Rasterkraftmikroskopie wird als Sonde eine wenige Mikrometer große Siliziumspitze eingesetzt, die sich an einem Ende eines Biegebalkens (Cantilever) befestigt ist. Abbildung 3.2 zeigt eine solche Messspitze.



Quelle: [NT-MDT 2002]

Abb. 3.2: Cantilever mit Spitze

Die gesamte Struktur wird durch bestimmte Ätzverfahren auf Siliziumwafern gefertigt. Die Messspitze ist meist pyramidenförmig mit polygonaler Grundfläche. Bei Annäherung an die Probenoberfläche kommt es zu einer anziehenden, bei weiterer Annäherung dann zu einer abstoßenden Kraftwirkung zwischen Probe und Spitze, wodurch sich der Cantilever verbiegt. In Abbildung 3.3 ist dieses Verhalten in einem Diagramm dargestellt. Da die auftretende Kraft das Resultat von Wechselwirkungen mehrerer Atome ist, liegt das Auflösungsvermögen nicht so hoch wie beim STM. Es treten zusätzlich stärker Konvolutionseffekte auf, die ihre Ursache in der Geometrie der Tastspitze haben und zu einer Unschärfe der Abbildung führen.



Quelle: [BAM 2003]

Abb. 3.3: Kraft-Distanz-Kurve bei Annäherung der Sonde

In der Betriebsart mit konstantem Abstand der Sonde zur Oberfläche wird im Fall der Rasterkraftmikroskopie die Auslenkung des Cantilevers, die ein direktes Resultat der an der Messspitze auftretenden Kräfte ist, gemessen. Um die Auslenkung des Cantilevers zu bestimmen wird das Lichtzeigerprinzip angewendet. In Abbildung 3.4 sind die

Funktionsweise dieses Prinzips und die Umsetzung am AFM skizziert. Mit einem Laserstrahl wird die Rückseite des Cantilevers angeleuchtet. Der reflektierte Strahl trifft auf das Zentrum einer segmentierten Photodiode. Bei Auslenkung des Cantilevers verändert auch der reflektierte Laserstrahl seinen Auftreffpunkt. Über die Differenz der aus den Photoströmen der in Auslenkrichtung des Strahls angeordneten Segmente resultierenden Spannung kann die Auslenkung des Cantilevers bestimmt werden. Hieraus sowie aus seiner Federkonstante ergibt sich die auf die Messspitze wirkende Kraft. Die genannte Spannung kann also als Regelparameter für die Abstandsregelung während des Scanvorgangs verwendet werden. Diese eben beschriebene, einfachste Betriebsart des AFMs wird auch als statischer Modus bezeichnet. Üblich ist des Weiteren noch der sog. Dynamische Modus. Hierfür wird der Cantilever zusätzlich mit seiner Eigenfrequenz angeregt und so in Schwingung versetzt. Durch die Kräfte die bei Annäherung an die Oberfläche auf die Tastspitze wirken, wird die Schwingung gedämpft und die Resonanzfrequenz verschiebt sich. Dies kann ebenfalls gemessen und zur Abstandsregelung herangezogen werden. Je nach Anwendung können noch weitere Betriebsmodi eingesetzt werden z. B. das sog. Tapping.

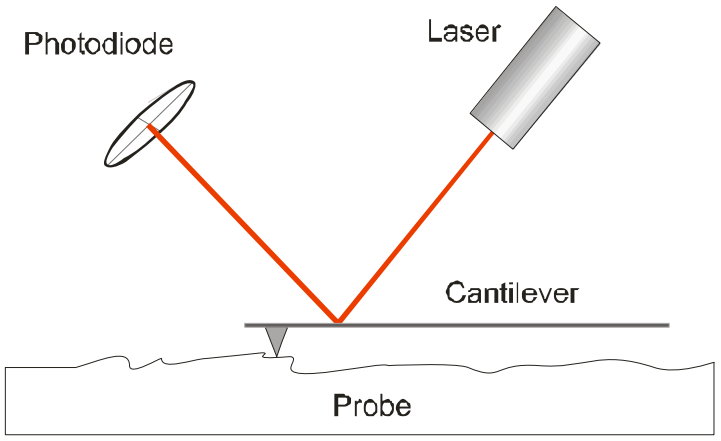


Abb. 3.4: Detektion der Kraftwirkung Probe - Messspitze am AFM

Nachdem die Probe abgerastert wurde liegt im Ergebnis ein topografisches Profil der Probenoberfläche vor. Abwandlungen des Verfahrens ermöglichen es zusätzlich, mechanische Eigenschaften, wie z.B. Reibung oder Härte der Probe, zu bestimmen. Mit speziell beschichteten Sonden ist es außerdem möglich elektrostatische oder magnetische Messungen vorzunehmen.

3.1.3 Anwendungsgebiete

Anwendungen für die Rasterkraftmikroskopie finden sich v. a. im medizinischen oder biologischen Bereich. Hier ist von Vorteil, dass das Verfahren nicht nur im Vakuum arbeitet und die Proben auch nicht erst mit Metall bedampft werden müssen wie das bei der herkömmlichen Rasterelektronenmikroskopie notwendig ist. Außerdem kann der Cantilever mit Messspitze auch in Flüssigkeiten eingetaucht werden und z. B. Messungen an lebenden Proben vorgenommen werden. In der Werkstoffwissenschaft kann die Rasterkraftmikroskopie für Mikro- und Nanostrukturuntersuchungen sowie für die Bestimmung von Werkstoffeigenschaften wie z. B. Härte, Elastizität oder magnetische Eigenschaften eingesetzt werden. Als Beispiel sei hier die Entwicklung von magnetischen Aufzeichnungsschichten für Festplatten genannt. Besonders interessant ist auch die Möglichkeit der Durchführung von Mikro- bzw. Nanomanipulation an der Probe mit Hilfe der Messspitze.

Die Anwendungsgebiete sind also vielfältig. Aufgrund der teilweise komplizierten Handhabung und eines normalerweise langsamen Scanvorgangs, der je nach Gerät und gewünschter Qualität der Aufnahme bis zu mehrere Stunden in Anspruch nehmen kann, finden Rasterkraftmikroskope jedoch nur selten außerhalb von Forschung und Entwicklung Anwendung.

Auch bei dem in dieser Arbeit im Mittelpunkt stehenden Gerät handelt es sich um ein AFM nach dem oben beschriebenen Prinzip. Konkret wird hier bei feststehender Sonde die Probe bewegt. Der Abstand Sonde – Probe wird während des Scanvorgangs konstant gehalten. Es ist sowohl statischer als auch dynamischer Modus möglich.

3.2 Digitale Signalprozessoren

Der im Rahmen dieser Arbeit verwendete Steuerrechner basiert auf Digitalen Signalprozessoren. Im folgenden Abschnitt wird daher ein kurzer Überblick über diese Art von Prozessoren gegeben. Es wird dabei auf ihre Besonderheiten im Verglich zu herkömmlichen Prozessoren eingegangen. Weiterhin werden ihre Einsatzgebiete kurz umrissen.

Bei einem Digitalen Signalprozessor (DSP) handelt es sich um einen Mikroprozessor, der speziell für Signalverarbeitungsaufgaben ausgelegt wurde. Die digitale Signalverarbeitung verfolgt den Ansatz, mit Hilfe von Rechenalgorithmen solche Aufgaben auszuführen, für die ansonsten eine u. U. wesentlich komplexere elektronische Schaltung nötig sein würde. Durch die Möglichkeit der Implementierung

der Rechenalgorithmen in Software bieten DSPs eine hohe Flexibilität. Mit Hilfe des Kriteriums der flexiblen Programmierbarkeit lassen sie sich von applikationsspezifischen Bauelementen wie ASSPs (Application Specific Standard Products), FPGAs (Field Programmable Gate Arrays) und ASICs (Application Specific Integrated Circuits) auf der einen, sowie Mediaprozessoren und Embedded CPUs auf der anderen Seite abgrenzen [BDTI 2003]. Daneben existiert noch die Klasse der oft kostengünstigen Mikrocontroller, welche sich wegen ihrer flexibeln Programmierbarkeit gut für Steuerungs-, wegen ihrer geringen Rechenleistung aber schlecht für Signalverarbeitungsaufgaben eignen. (vgl.: [Eyre, Bier 2000])

Ein übliches Szenario bei der digitalen Signalverarbeitung ist das kontinuierliche Digitalisieren eines Signals, die Bearbeitung der dabei entstehenden Daten durch einen DSP, z. B. mit einem Algorithmus zur digitalen Filterung, und deren Umwandlung zurück in ein analoges Signal. Um diese Aufgabe zu erfüllen muss der eingesetzte DSP nicht nur die Kapazitäten zur Ausführung des entsprechenden Algorithmus besitzen, sondern er muss auch evtl. vorgegebene Zeitbedingungen für die Verarbeitung einhalten können.

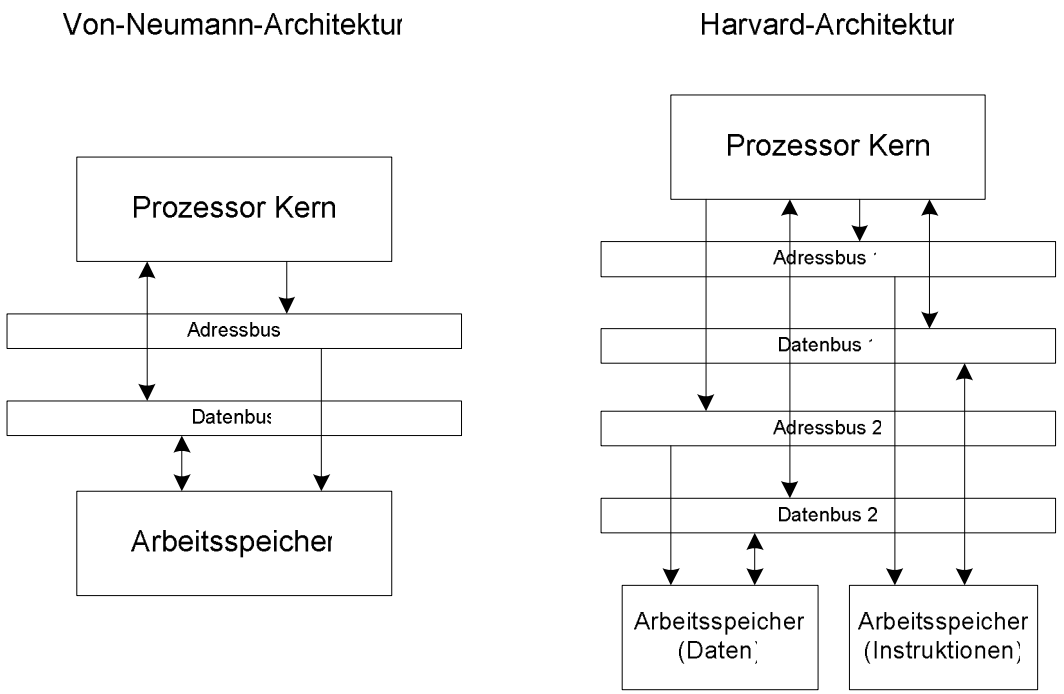
Aufgrund dieser Anforderungen ergibt sich ein Befehlssatz, der darauf optimiert ist, dass alle Befehle in konstanter Zeit abgearbeitet werden. Der Vorteil ist eine für den Programmierer leicht zu ermittelnde Ausführungszeit von Programmsegmenten. Herkömmliche CPUs versuchen hingegen mittels diverser Techniken wie pipelining, branch-prediction, speculative execution etc. die durchschnittliche Ausführungszeit von Programmsequenzen zu minimieren. Dadurch wird die Vorhersage der Ausführungszeit jedoch komplizierter, da diese nun auch vom Zustand des Systems vor der Ausführung abhängig ist. Der hier eingesetzte ADSP-21061 aus der SHARC Familie ist beispielsweise in der Lage, die allermeisten Anweisungen in einem Taktzyklus auszuführen, unabhängig von den vorhergehenden Anweisungen. Um das ermöglichen wird u. a. auf eine lange Pipeline verzichtet, die Speicheradressierung erfolgt in Wörtern, die den zu verarbeitenden Daten entsprechen, und es steht eine große Anzahl von Registern zur Verfügung. Der Aufbau und die Möglichkeiten dieses speziellen DSPs werden zusammen mit den Implikationen für die Systemsoftware in Abschnitt 3 dieses Kapitels näher erläutert. Um den Datenfluss zwischen Speicher und Prozessorkern zu optimieren wird anstelle der bei Desktoprechner üblichen Von-Neumann- bei DSPs oft die Harvard-Architektur[7] verwendet. Bei den meisten Designs wird zusätzlich schneller Speicher in Form von On-Chip SRAM eingesetzt. Weiterhin stehen oft Möglichkeiten für DMA-Transfers bereit, um Daten im Hintergrund anzuliefern oder abzutransportieren. (vgl.: [Cravotta 2002])

[7] siehe auch den folgenden Abschnitt

Die Spezialisierung auf Signalverarbeitungsaufgaben im Allgemeinen, bzw. auf einige Teilaspekte der digitalen Signalverarbeitung, birgt besonders für den Einsatz im Embedded-Bereich Vorteile. [Campbell, 1998] führt hier das Verhältnis von Verarbeitungsgeschwindigkeit zu Leistungsaufnahme bei der Bearbeitung definierter Aufgaben als Kriterium an. Bei der Durchführung von Fast-Fourier-Transformationen ergibt sich dabei ein deutlicher Vorteil für DSPs im Vergleich zu herkömmlichen RISC Prozessoren.

3.2.1 Die Harvard-Architektur

DSPs sind oft nach der Harvard Architektur aufgebaut. Dies ergibt sich aus den Anforderungen an DSPs, große Mengen an Daten zu verarbeiten oder zumindest während der Verarbeitung häufig auf den Arbeitsspeicher zuzugreifen [Lapsley et al. 1996]. Um große Mengen an Daten effizient in und aus dem Speicher zu transportieren bietet die Harvard- Vorteile gegenüber der Von-Neumann Architektur.



vgl.: [Lapsley et al. 1996]

Abb. 3.5: Harvard- und Von-Neumann-Architektur

Der wesentliche Unterschied beider Architekturen ist, dass der Von-Neumannsche Flaschenhals, den der Zugang zum Arbeitspeicher darstellt, in der Harvard-Architektur

durch die Verwendung von getrennten Daten- und Instruktionsbussen geweitet wird. Dadurch können Instruktionen und Daten parallel anstelle von sequenziell gelesen werden. Abbildung 3.5 vergleicht den schematischen Aufbau beider Architekturen. Einige DSP Familien führen das Konzept noch weiter und verwenden zusätzlich noch einen zweiten Datenbus.

Ein wesentlicher Nachteil der Harvard Architektur ist dagegen das Vorhandensein von zwei getrennten Speicherbereichen für Instruktionen und Daten. Dies schränkt die Flexibilität der Speicherverwaltung ein, was besonders bei DSPs, welche oft nur über sehr wenig Speicher verfügen, Nachteile haben kann.

3.2.2 Anwendungsgebiete

Aufgrund ihrer vielfältigen Einsatzmöglichkeiten, die sich durch ihre flexible Programmierbarkeit ergeben, haben DSPs in vielen Anwendungsbereichen Einzug gehalten [Marti, Plettl 2003]. In der folgenden Tabelle sind exemplarisch einige Anwendungsfälle, in denen der Einsatz von DSPs in Frage kommt, dargestellt.

Tab. 3.2: Anwendungsgebiete von DSPs

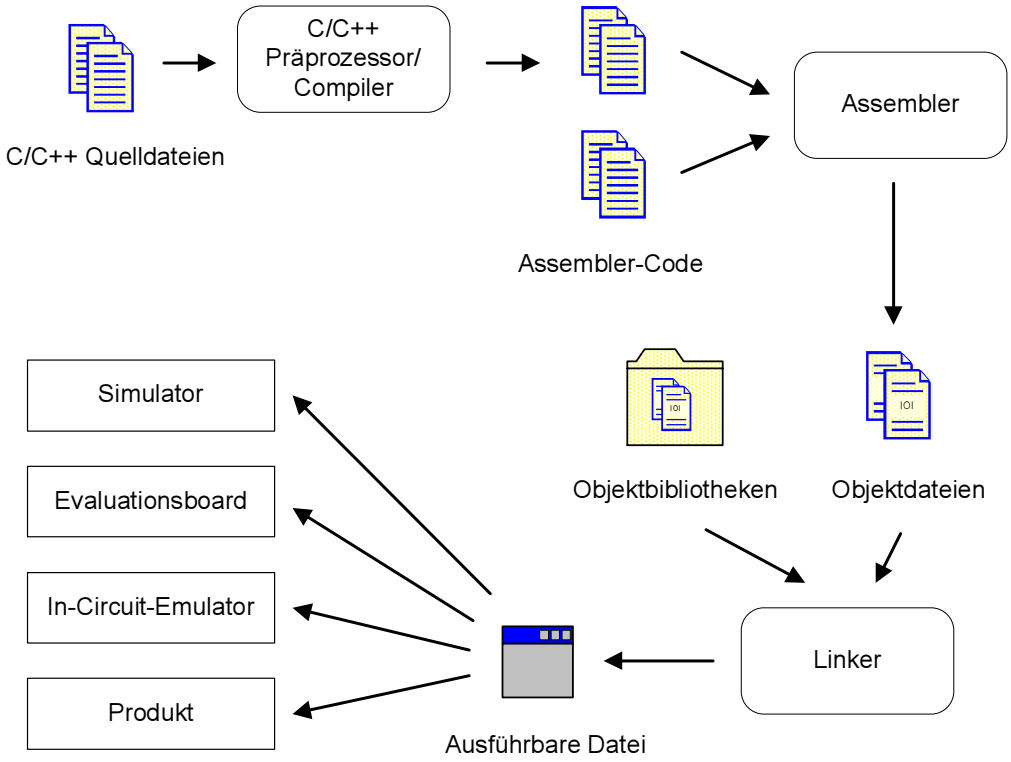
Anwendungsbereich	**Gerät**	**Aufgabe des DSPs (u. a.)**
Computertechnik	Modem	Signalanalyse / -synthese
- " -	Festplatte	Hochgeschwindigkeitsregelung, Signalkodierung / -dekodierung
Telekommunikation	Bildtelefon	Videokompression
- " -	Mobiltelefon	Signalkodierung / -dekodierung
- " -	Vermittlungs-Stelle	versch. Aufgaben, z. B. Rausch- und Echounterdrückung mittels digitaler Filter
Fototechnik	Digitalkamera	Bildanalyse (Autofocus), Bildverbesserung, Datenkompression (JPEG)
Audio und Video	DVD-Player	MPEG-II-Dekodierung
	MP3-Player	Audio-Dekodierung, Digitalfilter
Regelungstechnik	Roboter	diverse Anwendungen möglich; Beispiel: Nahtverfolgung bei Schweißrobotern

Der Einsatz von DSPs in den genannten Bereichen wird u. a. vorangetrieben durch den parallelen Einsatz von digitaler Datenarchivierung. Da größere Mengen gleichartiger Daten mittlerweile v. a. in Geräten der Unterhaltungselektronik aber auch in wissenschaftlichen Anwendungen wie der Geophysik [Müller 2003] in digitaler Form verarbeitet und gespeichert werden müssen, bietet sich hier der Einsatz von DSPs an. Aufgrund der hohen Rechenleistung bei gleichzeitiger, freier Programmierbarkeit von moderner DSPs bieten sich diese, eingebettet in entsprechende Geräte, für eine gleichzeitige Nutzung für Steuerungsaufgaben an. Als Beispiele für solche Geräte kommen u. a. DVD oder MP3-Player in Frage.

3.2.3 Softwareentwicklung für DSP-Anwendungen

Die grundlegenden Werkzeuge für die Entwicklung von Software für DSP-basierte Systeme unterscheiden sich kaum von denen für PCs oder Workstations. Hierbei handelt es sich vor allem um C oder C++ Compiler, Assembler und Linker. Dazu kommt üblicherweise ein Debugger in Verbindung mit einem Simulator bzw. einem entsprechenden Evaluationsboard, welche den DSP nachbilden bzw. enthalten. Bei Produkten, die nicht in einer größeren Serie gebaut werden sollen, wird oft ein solches Evaluationsboard, welches normalerweise vom Hersteller des DSP's für Test- oder Demozwecke geliefert wird, im fertigen Produkt verwendet [BDTI 2000]. Ansonsten bietet sich für Debugging-Zwecke der Einsatz eins In-Circuit Emulators (ICE) an. Dieser emuliert die Funktionen des DSP's im Kontext des Gesamtprodukts und bietet dem Entwickler Möglichkeiten zur Überwachung und Fehlersuche. Abbildung 3.6 zeigt das Zusammenwirken der auch im PC-Bereich üblichen Entwicklungstools bei der Softwareentwicklung für DSPs.

Die mittlerweile recht große Auswahl an DSP-Familien unterschiedlicher Hersteller mit teilweise sehr ähnlichem Funktionsumfang [Cravotta 2002] führt dazu, dass bei der Entscheidung für einen speziellen DSP auch die verfügbare Entwicklungsumgebung und hier v. a. auch die Möglichkeiten zur Softwareentwicklung in den Vordergrund treten. Von den Herstellern der jeweiligen DSPs werden oft eigene Entwicklungsumgebungen angeboten. Diese integrieren die üblichen Werkzeuge wie Compiler, Debugger, Simulator etc. und bieten über das Vorhandensein der C-Standardbibliothek weitere Sammlungen mit optimierten Signalverarbeitungsroutinen. Ein Beispiel für eine solche Entwicklungsumgebung ist "VisualDSP++" von Analog Devices [ADI 2003].



vgl.: [BDTI 2000]

Abb. 3.6: Werkzeuge bei der Softwareentwicklung für DSP-Systeme

Trotz der Verfügbarkeit von Hochsprachen wie C und C++ wird oft auch weiterhin Assembler eingesetzt, da die verfügbaren Compiler normalerweise nicht ausreichend gut optimierten Code liefern. Gründe hierfür liegen in einer großen Varianz von speziellen Hardwareoptimierungen auch innerhalb einer DSP-Familie. Lösungsansätze für dieses Problem sind in Form von Strategien für retargierbare, optimierende Compiler vorhanden [Leupers 1997].

Es sind von Drittherstellern neben verschiedenen Bibliotheken für Signalverarbeitungsaufgaben und zur Ansteuerung von Geräten außerdem eine Reihe von Betriebsystemen speziell für DSPs verfügbar. Diese beschäftigen sich hauptsächlich mit dem Bereitstellen von Funktionen zur Prozessverwaltung und Synchronisation bieten aber auch teilweise Kommunikationsschnittstellen und Unterstützung für Multi-DSP Systeme. Beispiele für solche Systemsoftware sind OSEck [OSE 2003] und µC/OS-II.

Eine weitere Möglichkeit der Softwareentwicklung für DSPs ist der Einsatz von grafischen Rapid-Prototyping Systemen. Hierbei handelt es sich um Software, die eine Bibliothek mit vorgefertigten Komponenten zur Verfügung stellt, aus denen dann der Entwickler grafisch ein Modell des Systems zusammensetzen kann. Anhand dieses

Modells wird im Anschluss automatisch Quellcode bzw. ein ausführbares Programm generiert welches die gewünschte Funktionalität umsetzt. Ein Beispiel für eine solche Entwicklungsumgebung stellt Matlab/Simulink [MathWorks 2003] dar.

3.3 Aufbau des vorliegenden Systems

3.3.1 Überblick

Das Gesamtsystem besteht aus verschiedenen Hard- und Softwarekomponenten, die man unter Berücksichtigung von funktionalen Gesichtspunkten in drei Gruppen unterteilen kann. Hierbei handelt es sich um einen PC für den Benutzer, den Steuerrechner und die Stage. Abbildung 3.7 zeigt diese drei Gruppen im Zusammenhang.

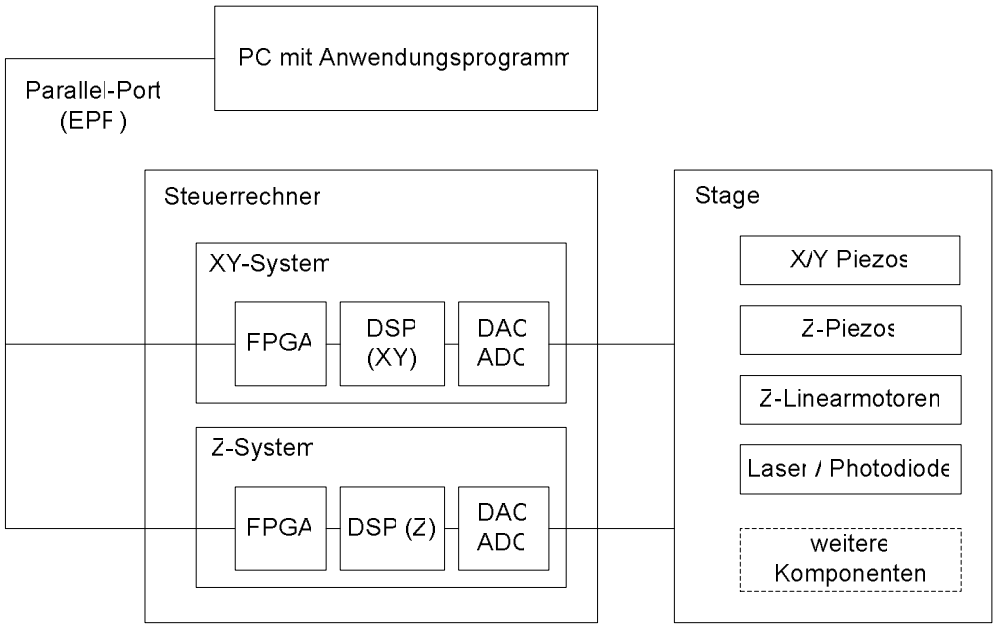


Abb. 3.7: Überblick über den Systemaufbau

Bei dem PC handelt es sich um ein Windows NT basiertes System, auf welchem dem Benutzer ein Anwendungsprogramm zur Durchführung von Scanvorgängen zur Verfügung steht. Das Anwendungsprogramm „QuickScan“ liegt im Quelltext vor und bietet neben der Möglichkeit zum durchführen des Scanvorgangs und zum Sammeln der dabei anfallenden Daten auch Werkzeuge zum Einstellen verschiedene Parameter sowie der anschließenden Darstellung der Messergebnisse. Der PC verfügt über eine

herkömmliche Parallelschnittstelle, die vom dem Anwendungsprogramm angesprochen wird und über die die Kommunikation mit beiden Teilsystemen des Steuerrechners erfolgt.

Der Steuerrechner besteht aus zwei unabhängigen Teilsystemen, dem XY- und dem Z-System. Diese Bezeichnungen leiten sich aus den von den Teilsystemen angesprochenen Komponenten der Stage und damit von den von ihnen zu bewältigenden Aufgaben ab. Details hierzu werden in den folgenden Abschnitten beschrieben. Beide Teilsysteme des Steuerrechners bestehen im Wesentlichen aus gleichen Hardwarekomponenten. Das Kernstück bildet ein leicht modifiziertes Evaluations-board „EZ-KIT lite“ vom Hersteller Analog Devices welches mit einem Prozessor vom Typ SHARC ADSP-21061 ausgestattet ist. Dieses Board ist mit einem EPROM (Electrical Programmable Read Only Memory) Sockel sowie einer RS-232 Schnittstelle zum Aufspielen von Software ausgestattet. Es wurde erweitert durch einem zusätzlichen FPGA (Field Programmable Gate Array), der einen Parallelport implementiert, über welchen die Kommunikation mit dem PC erfolgt. Weiterhin stehen jedem Teilsystem eine Reihe von ADCs (Analog-Digital-Converter) und DACs (Digital-Analog-Converter) zu Verfügung, über die die einzelnen Komponenten der Stage angesprochen werden.

Zur Stage werden hier alle mechanischen Teile des Mikroskops gezählt. Die Stage nimmt die zu untersuchende Probe auf und realisiert deren Abtastung. Mit ihr verbunden sind außerdem weitere analoge, elektrische Komponenten, die für den Betrieb der mechanischen Teile notwendig sind. Außerdem ist ein herkömmliches Lichtmikroskop integriert, welches den Benutzer beim einlegen und justieren der Probe unterstützt.

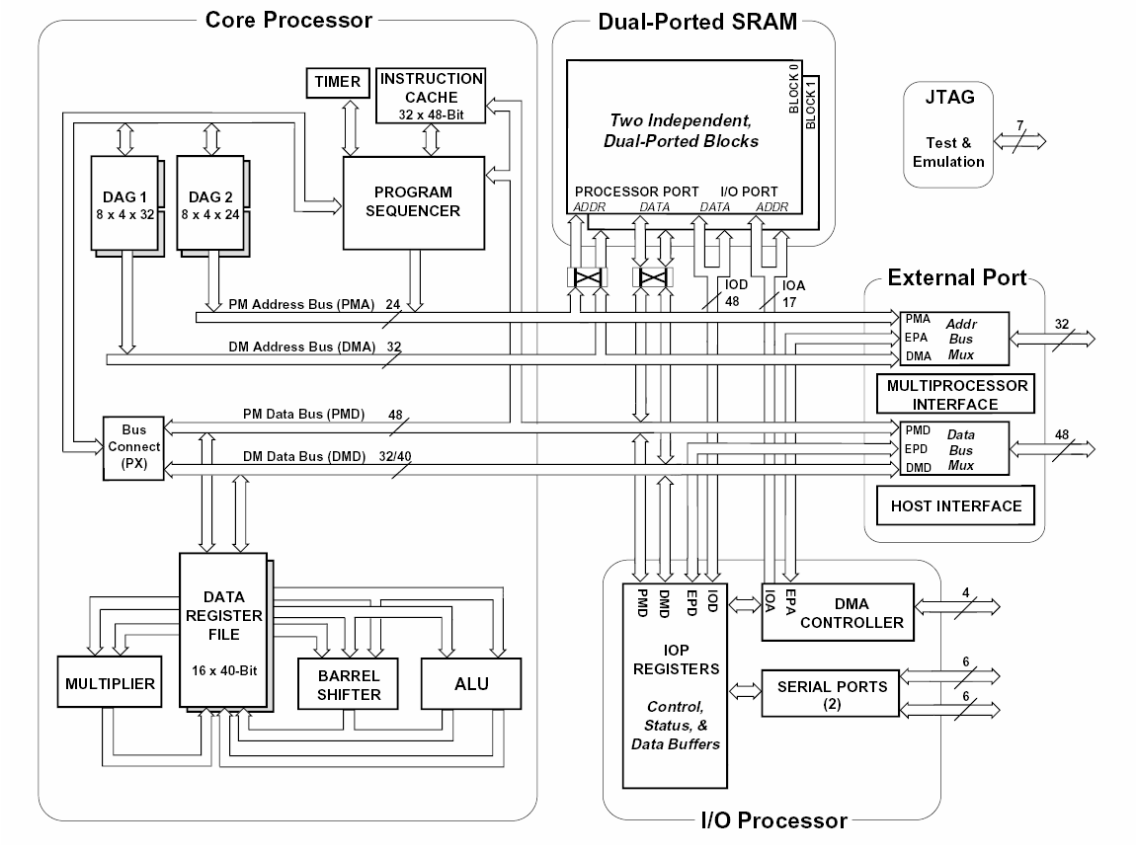
3.3.2 Der ADSP-21061

DSPs vom Typ SHARC ADSP-21061 bilden den Kern der beiden Teilsysteme des Steuerrechners. Es handelt sich um 32-Bit Prozessoren, die sowohl Fest- als auch Gleitkommazahlen verarbeiten können. Dieser DSP-Typ wird mit 40 MHz getaktet und verfügt über einen internen Speicher von 128 kB. Der interne Aufbau ist an der Harvard Architektur angelehnt. Um die hier unterschiedlichen Busse für Daten und Instruktionen optimal zu unterstützen wurde der interne Speicher mit Hilfe von zweikanaligem SRAM (Static Random Access Memory) implementiert.

Der Speicher wird linear adressiert. Die kleinsten adressierbaren Einheiten sind 32- bzw. 48-Bit Wörter. Es wird vor dem ausführen eines Benutzerprogramms festgelegt,

unter welchen Adressen auf 32- (Daten, Stack, etc.) und unter welchen auf 48-Bit Speichereinheiten (Instruktionen) zugegriffen werden soll. Zur Signalisierung externer Ereignisse stehen vier Interrupteingänge zur Verfügung. Zwei weitere Interrupts können durch einen integrierten, programmierbaren Timer ausgelöst werden.

Der Prozessor verfügt weiterhin über 16 allgemeine und eine Reihe spezieller Register[8], die Kopien haben, zwischen denen zwecks schneller Unterbrechungsbehandlung umgeschaltet werden kann. Dieser Shadow-Registersatz bietet sich für die Umsetzung eines Tasks an, zu dem mit besonders wenig Overhead umgeschaltet werden kann. Dieses spezielle Feature wird vom zur Verfügung stehenden C-Compiler nicht unterstützt. Es eignet sich jedoch gut, um das später vorgestellte SuperTask-Konzept auf Betriebssystemebene zu unterstützen.



Quelle: SHARC User's Manual

Abb. 3.8: SHARC ADSP-21061 Block Diagramm

[8] in Abbildung 3.8 mit einem Schatten hinterlegt

Insgesamt ist der SHARC auf kurze, deterministische Ausführungszeiten von Instruktionen sowie auf die kontinuierliche Ein- und Ausgabe von Datenströmen ausgelegt. Ersteres ist daran zu erkennen, dass fast alle Instruktionen in einem Takt abgearbeitet werden (Sprünge benötigen drei Takte). Für die Ein- und Ausgabe steht ein DMA-Controller bereit, der in diesem Fall den Datentransfer zum FPGA für den Parallelport übernimmt, sowie ein externer Port, über welchen die DACs und ADCs zum Steuern der Stage angesprochen werden.

3.3.3 XY-System

Bei dem XY-System handelt es sich, wie bereits in Abschnitt 3.3.1 beschrieben, um eines der beiden Teilsysteme des Steuerrechners.

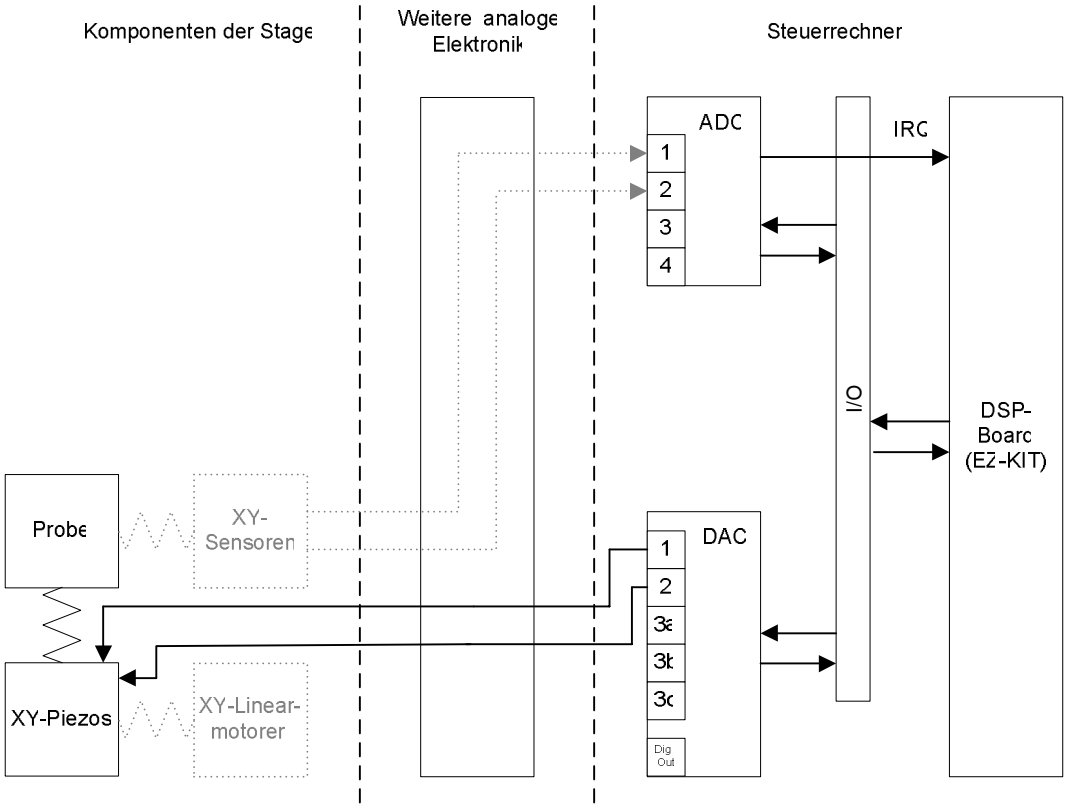


Abb. 3.9: XY-Systems mit angeschlossenen Komponenten

Es besteht wie auch das Z-System aus einem DSP-Board mit angeschlossenem FPGA für die Kommunikation mit dem PC sowie externen ADCs mit vier 16 Bit Eingängen (1-4) sowie externen DACs mit zwei 16-Bit- (1, 2) und drei gekoppelten 12-Bit-Ausgängen (3a-3c). Auf dem DAC-Board sind weiterhin noch mehrere digitale

Ausgänge untergebracht. An den drei gekoppelten 12-Bit-Ausgängen kann nur ein gemeinsamer Spannungswert ausgegeben werden. Sie sind jedoch über Relais individuell ein- und ausschaltbar.

Die DACs sowie die ADCs sind über den externen Port des SHARC Prozessors angebunden. Die entsprechenden Register zu ihrer Steuerung sind im Adressbereich des Prozessors eingeblendet. Die ADCs können mit einer Abtastfrequenz von 100 kHz arbeiten und sind in der Lage, einen neuen Abtastwert dem Prozessor mit Hilfe von IRQ0 zu signalisieren.

Das XY-System ist für die Ansteuerung derjenigen Piezoaktoren zuständig, die die zu untersuchende Probe in der horizontalen Ebene verschieben. Diese Aktoren dienen zur Feinpositionierung der Probe und führen bei feststehender Messspitze die zeilenweise Scanbewegung aus, indem sie die Probe relativ zur Messspitze bewegen. Die Piezos für die X- bzw. Y-Richtung sind jeweils mit einem Ausgang der DACs verbunden. Zwischengeschaltet sind noch entsprechende Verstärker. Entsprechend der an den Ausgängen angelegten Spannung dehnen sich die Piezokristalle mehr oder weniger aus, was wiederum zum Anfahren einer bestimmten Position in der XY-Ebene führt. Da die Längenausdehnung der Piezos nicht genau proportional zur angelegten Spannung ist, ist es sinnvoll, Sensoren zu integrieren, die die tatsächlich angefahrene Position ermitteln und als Spannungswerte an den Eingängen der ADCs zu Verfügung stellen. Diese in Abbildung 3.9 grau gestrichelt dargestellten Sensoren stellen eine mögliche zukünftige Erweiterung dar. Weiterhin sind noch zwei Linearmotoren vorhanden, die zur Grobpositionierung der Probe dienen. Diese sind jedoch nicht per Software steuerbar sondern werden unabhängig mittels zusätzlicher Schalter bedient.

3.3.4 Z-System

Das Z-System ist das zweite Teilsystem des Steuerrechners und verfügt neben dem DSP-Board genau wie das XY-System auch über einen FPGA zur Kommunikation, sowie externe ADCs und DACs.

Die Aufgabe des Z-Systems besteht im Wesentlichen in der Steuerung aller Komponenten, die den Abstand Sonde – Probe (Z-Richtung) beeinflussen bzw. messen. Die Messspitze ist an einer unbeweglichen Halterung montiert und steht somit auch in Z-Richtung fest. Es wird die Probe mit Hilfe dreier parallel arbeitender Piezoaktoren von unten an die Sonde angenähert oder weggeführt. Die Ausdehnung dieser Piezos und damit der Abstand Probe – Sonde wird durch die Spannung am Ausgang des ersten Ports der DACs vorgegeben. Der gesamte Aufbau bestehend aus Probe und Piezos kann

weiterhin mittels dreier Linearmotoren grob an die Messspitze angenähert oder von dieser weggeführt werden. Die Motoren können über die gekoppelten 12 Bit Ausgänge der DACs gesteuert werden. Dabei kann eine beliebige Kombination von Motoren je nach ausgegebener Spannung vor oder zurück gefahren werden. Da die Motoren in einem Dreieck angeordnet sind, ist es auch möglich die Probe geringfügig zu kippen.

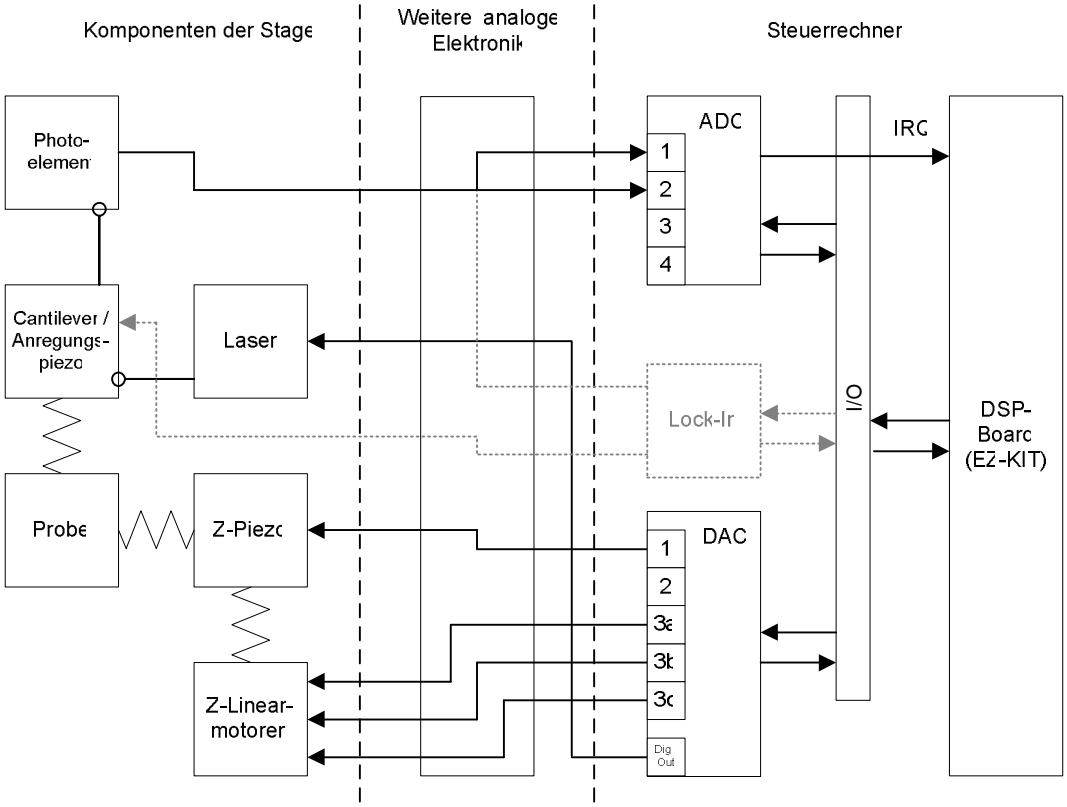


Abb. 3.10: Z-Systems mit angeschlossenen Komponenten

Zur Messung der Kraftwirkung zwischen Messspitze und Probe kommt das in Kapitel 3.1.2 beschriebene Lichtzeigerprinzip zum Einsatz. Als Lichtquelle dient ein Laser, der über einen der Digitalausgänge, die sich zusätzlich auf dem DAC-Board befinden ein- bzw. ausgeschaltet werden kann. Die Auslenkung des vom Biegebalken (Cantilever) reflektierten Laserstrahls wird von einem segmentierten Photoelement gemessen und steht in Form von zwei Spannungswerten an den Eingängen eins und zwei der ADCs zur Verfügung. Die beiden Spannungswerte repräsentieren dabei die Auslenkung in vertikaler als auch die in horizontaler Richtung.

Eine weitere Komponente, die für den Betrieb im dynamischen Modus[9] benötigt wird ist der sog. Lock-In Verstärker. Dieser ist ebenfalls an den externen Port des SHARC-

[9] Modus mit Anregung des Cantilevers mit seiner Eigenfrequenz; siehe auch Kapitel 3.1.2

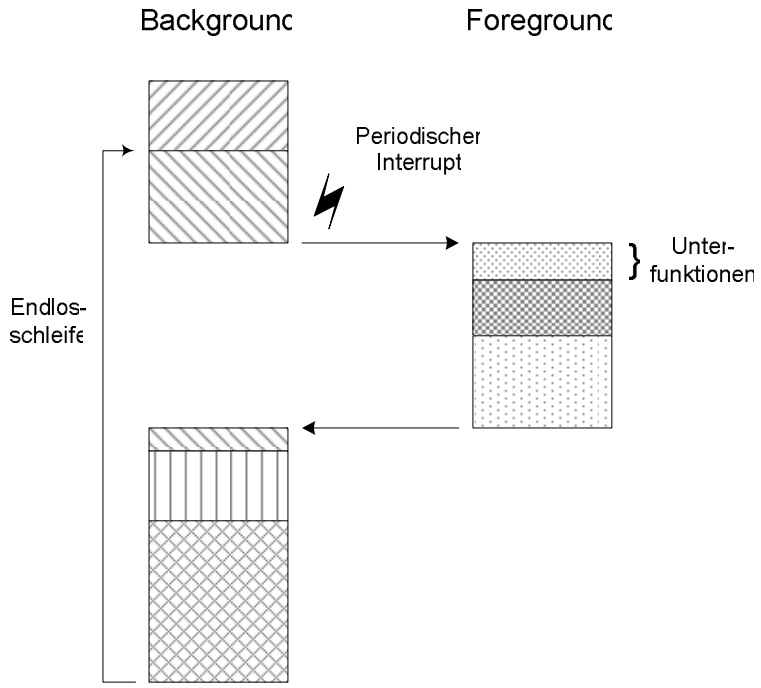
DSPs angeschlossen und dient zum erzeugen der Anregungsfrequenz des Biegebalkens. Weiterhin erzeugt er aus dem vom Photoelement gelieferten Signal unter Berücksichtigung der Eigenfrequenz des Cantilevers ein Maß für die Kraftwirkung Probe – Messspitze. Diese Komponente ist zwar ein wichtiger Bestandteil des Gesamtsystems, ist aber für die einfache, statische Betriebsart nicht notwendig. Für die Betrachtungen in dieser Arbeit wird sie daher als optionale Komponente betrachtet.

4 Ist-Zustand der bestehenden Softwarelösung

Grundlage für diese Arbeit bildet ein bereits vorhandener und funktionstüchtiger Aufbau eines Rasterkraftmikroskops. Dementsprechend existiert auch bereits eine Softwarelösung zur Steuerung des Geräts. Diese wurde vom insolventen Hersteller des Steuerrechners ursprünglich als Teil eines geschlossenen Gesamtprodukts konzipiert und besteht aus zwei Programmen für je ein Teilsystem des Steuerrechners sowie einem Anwendungsprogramm für den PC.

4.1 DSP-Software

Bei der DSP-Software handelt es sich um die Software des Steuerrechners. Es sind individuelle Versionen für das XY- sowie das Z-System vorhanden, die sich vom Aufbau her ähneln. Der Quelltext der in C geschriebenen Software steht zur Verfügung. Die Software ist jeweils als Foreground/Background-System aufgebaut. Nach der Initialisierung geht das Hauptprogramm in eine Endlosschleife. In dieser werden hintereinander verschiedene Unterfunktionen aufgerufen, die die folgenden Aufgaben erledigen: Übertragen von Daten aus dem Sendepuffer zum PC, überprüfen des Empfangspuffers auf neue Daten, interpretieren dieser Daten als Kommandos und ggf. ausführen der empfangenen Kommandos.



vgl.: [Labrosse 2002, S. 36]

Abb. 4.1: Foreground/Background-System der bestehenden Lösung

Diese Schleife bildet den sog. Background. Den Foreground bilden zeitkritische Funktionen wie Regelung und Messwerterfassung, die innerhalb eines Interrupthandlers und je nach Betriebszustand des Teilsystems nacheinander abgearbeitet werden. Dieser ist mit dem Interrupt der ADC-Karte verbunden und wird periodisch aufgerufen.

Die Software implementiert eine Art Statusmaschine. Es werden Parameter die den aktuellen Zustand des Systems kennzeichnen, in einer Reihe globaler Variablen gespeichert. Zu diesen Parametern gehören z. B. die zuletzt ausgelesenen Werte der ADCs, die momentan ausgegebenen Werte der DACs, das letzte empfangene Datenwort, die vom Interrupthandler aktuell aufzurufenden Unterfunktionen sowie sämtliche Parameter, die diese Funktionen benötigen. Die Änderung des Zustandes erfolgt zum einen durch Abläufe innerhalb des Systems. Hierzu zählt z. B. das sich ändern von Messwerten der ADCs. Zum anderen wird der Zustand durch Kommandos vom PC geändert. Beispielsweise kann dieser das Starten der Abstandsregelung signalisieren. Die Ausführung des Kommandos besteht darin, den Zustand des Teilsystems so zu ändern, dass bei diesem Beispiel von nun am im Interrupthandler die Unterfunktion für die Regelung mit aufgerufen wird.

4.2 Anwendungsprogramm

Bei dem Anwendungsprogramm handelt es sich um die 32-Bit-Windows-Software „QuickScan“. Das Programm bietet dem Benutzer mit Hilfe eine grafische Oberfläche die Möglichkeit zum Einstellen diverser Parameter der Hardware, zum manuellen steuern verschiedener Hardwarefunktionen wie z. B. der Z-Linearmotoren sowie letztendlich zum scannen der Probe und der grafischen Darstellung der Ergebnisse. Es können außerdem einfache Analysen der Messergebnisse durchgeführt werden.

Die Software ist in C++ unter Verwendung der Microsoft Foundation Classes (MFC) geschrieben. Auch hier liegt der Quelltext vor. Sie ist grob modular aufgebaut. Eine Hauptkomponente ist das Rahmenprogramm, das Teile der grafischen Oberfläche implementiert und das für die Verwaltung und Darstellung der aufgenommenen Bilder zuständig ist. Eine weitere Komponente ist ein Modul, das für die Rasterkraftmikroskopie spezifische Funktionen und Teile der Oberfläche bereitstellt, die Funktionen des zu steuernden Geräts kennt sowie die Kommunikation mit dem Steuerrechner implementiert. Ein ähnliches Modul existiert noch für die Steuerung eines Elektronenmikroskops. Dieses Modul wir allerdings nicht benutzt.

4.3 Die Kommunikationslösung

Das bestehende Konzept kann man unter zwei Teilaspekten betrachten. Dabei handelt es sich zum einen um die Hardwarelösung, zum anderen um die Softwarekomponenten die diese Hardware nutzbar machen. Dabei ist zwischen der Software für die beiden Teilsystem des Steuerrechners sowie die für den Anwendungs-PC zu unterscheiden.

Das System arbeitet mit einer Verbindung über den EPP (Enhanced Parallel Port) des PC. Hierbei handelt es sich um einen in IEEE-1284 standardisierten Anschluss[10] welcher auf dem herkömmlichen Druckerport basiert und welcher in dieser Form an den meisten PCs verfügbar ist. Auf Seiten der beiden Teilsysteme des Steuerrechners wurde eine anwendungsspezifische Hardwarelösung für die Implementierung der Schnittstelle gewählt. Jedes der zwei Teilsysteme verfügt dabei über einen eigenen Anschluss an den gemeinsam genutzten EPP. Die Hardwarelösung, die den Anschluss der DSP-Boards an den EPP realisiert wurde, ist mit einer FPGA-basierten Schaltung aufgebaut. Diese Schaltung ist auf der einen Seite über den 8 Bit Parallel Port mit dem Windows-PC verbunden. Auf der anderen Seite über den externen 32-Bit-Prozessorbus an den DSP gekoppelt. Diese Anordnung ist in Abbildung 4.2 dargestellt.

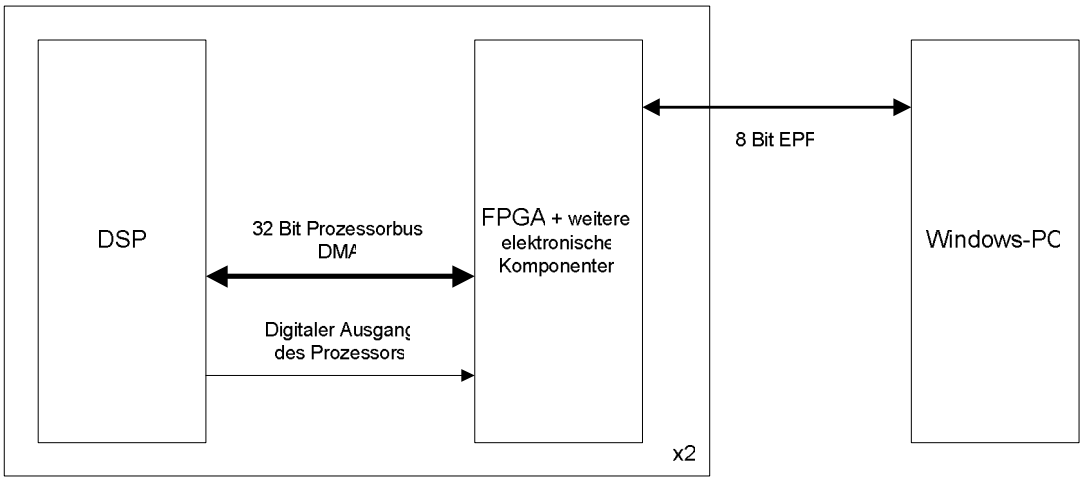


Abb. 4.2: Hardwarekomponenten der Kommunikationsschnittstelle

Die Übertragung von Daten zwischen FPGA und PC erfolgt byteweise in zusammenhängenden Schritten von je 4 Bytes. Im FPGA werden jeweils vier Übertragungen von je einem Byte zu einem 32-Bit Wort zusammengefasst und an den DSP weitergeleitet. Bei der Übertragung zum PC werden entsprechend 32-Bit in vier Bytes aufgeteilt übertragen. Dies geschieht transparent. Der Datenaustausch zum DSP

[10] weitere Informationen siehe Anhang B

erfolgt mittels DMA-Transfers aus dem Arbeitsspeicher des DSPs. Zusätzlich verfügt der FPGA über einen kleinen FIFO-Puffer. Das Vorliegen von zu sendenden Daten wird dem FPGA über einen digitalen Ausgang des Prozessors signalisiert.

Diese Hardware ist für beide Teilsysteme identisch. Beide sind dabei an denselben EPP des PCs angeschlossen, so dass sich hier eine Bus-Topologie ergibt. Um eine gemeinsame Nutzung des Ports zu gewährleisten, wurden dafür bereits in der Hardware entsprechende Vorkehrungen getroffen. Diese entsprechen dabei nicht der IEEE-1284.3 Spezifikation, welche ein Schema zur gemeinsamen Nutzung des Ports durch mehrere Geräte spezifiziert. In der vorliegenden Lösung wird durch Nutzung der im EPP Standard vorgesehenen Möglichkeit zur Unterscheidung von Übertragungen von Daten und Adressen mittels der Leitungen Data- bzw. Address Strobe genutzt, um durch Übertragungen von bestimmten Werten im Adressmodus jeweils einen der FPGAs zu deaktivieren. Der PC fungiert hier als Master und selektiert das jeweils aktive Gerät.

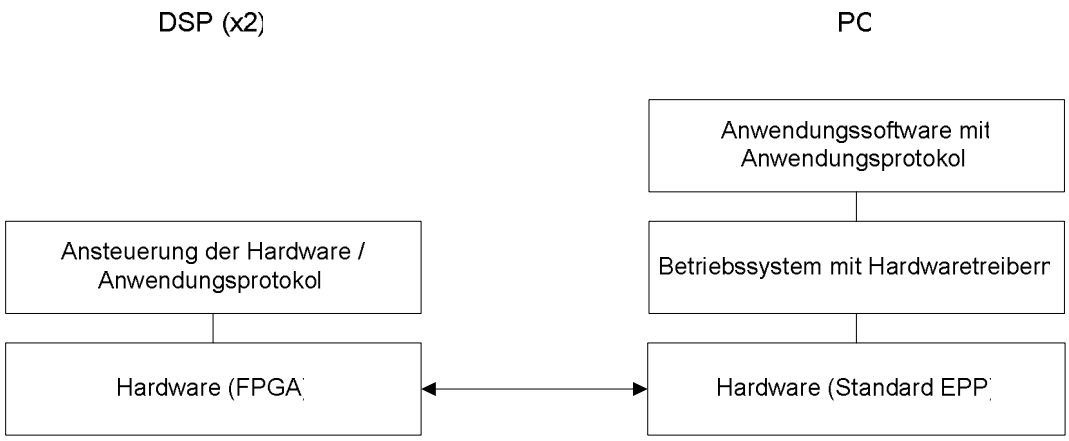


Abb. 4.3: Schema der Schichten des bestehenden Kommunikationsmodells

Auf der Softwareseite setzt die bestehende Steuerungssoftware direkt auf der Kommunikationshardware auf. Sie implementiert dabei ein einfaches Anwendungsprotokoll, welches die Fähigkeiten der Hardware, 32-Bit-Wörter am Stück übertragen zu können ausnutzt. Dabei ist jeder übertragene 32-Bit-Wert aufgeteilt in einen 8-Bit-Befehlsteil und einen 24-Bit-Teil für mögliche Nutzdaten. Das Senden geschieht durch Schreiben des zu übertragenen Wertes in einen Puffer im Arbeitsspeicher. Vom Hintergrundsystem wird festgestellt, ob Daten in diesem Puffer vorliegen, und bei Bedarf ein entsprechender DMA-Transfer zum FPGA aufgesetzt, und diesem signalisiert, dass Daten vorliegen. Der Empfang von Daten gestaltet sich ähnlich. Hier wird vom Hintergrundsystem geprüft, ob Daten in einen weiteren Puffer geschrieben wurden. Ist die der Fall werden diese als Befehle interpretiert und

entsprechend ausgeführt. Eine klare Einteilung in Schichten wie dies im ISO/OSI Referenzmodell praktiziert wird ist, wie in Abbildung 4.3 zu erkennen, dabei nur in Ansätzen gegeben. Auf der PC-Seite läuft die Anwendungssoftware als normales Benutzerprogramm und setzt auf entsprechende Systemfunktionen und Gerätetreiber auf. Diese wiederum steuern die Port-Hardware. Das Anwendungsprogramm implementiert die entsprechenden Funktionen zum Übertragen von 32 Bit Werten als einzelne Bytes sowie das selektieren des aktiven Teilsystems. Darauf aufbauend implementiert es ebenfalls das Anwendungsprotokoll. Diese Software ist bereits modular strukturiert.

4.4 Nachteile der existierenden Lösung

Die vorliegende Lösung hat verschiedene Nachteile, die zum einen den Betrieb des Geräts in seiner ursprünglichen Form betreffen, zum anderen auch die Weiterentwicklung des Gesamtsystems einschränken.

Ein Kritikpunkt der existierenden Lösung, der das Ergebnis der Bilderfassung störend beeinflussen kann ist die unzureichende Synchronisation der Signalisierung von Ereignissen auf den beiden Teilsystemen des Steuerrechners. Dieser Mangel ist von Bedeutung wenn während des Scanvorgangs das Abtasten einer Zeile auf dem Z-System sowie das Ausführen der Scanbewegung auf dem XY-System synchron gestartet werden müssen. Der Grund für hier möglicherweise auftretende zeitliche Diskrepanzen liegt darin, dass die beiden Teilsysteme hintereinander vom Anwendungsprogramm benachrichtigt werden. Dabei kann die nicht echtzeitfähige Windows Anwendung zwischen dem Senden der einzelnen Benachrichtigungen unterbrochen werden. Die Folge eines unterschiedlichen Starts beider Vorgänge ist eine verschlechterte Qualität des aufgenommenen Bildes. Der eben angesprochene Effekt ist allerdings sehr selten zu beobachten, da die Wahrscheinlichkeit einer Unterbrechung des Anwendungsprogramms genau zwischen dem Senden der beiden Benachrichtigungen gering ist.

Neben der genannten Unzulänglichkeit der existierenden Lösung liegt der hauptsächliche Grund für eine Neukonzeption der Software in der schlechten Erweiterbarkeit der Lösung. Besonders die bei Modifikationen und Erweiterungen von Teilfunktionen des Interrupthandlers stößt man schnell an die Grenzen der bestehenden Lösung. Dies liegt an der wenigen Rechenzeit die für die Abarbeitung des Interrupthandlers zur Verfügung steht. Es muss sichergestellt sein, dass das Hintergrundsystem nicht zu lange oder gar vollständig vom Vordergrundsystem

verdrängt wird, da dieses wichtige Funktionen bereitstellt. Um dies zu gewährleisten ist es notwendig die WCET des Interrupthandlers bei jeder Überarbeitung zu bestimmen, was sich als schwierig erwiesen hat[11]. Weiterhin erhöhen sich mit jeder Erweiterung sowohl des Vordergrund als auch des Hintergrundsystems die Antwortzeiten aller im Hintergrundsystem implementierten Unterfunktionen, da die Anzahl der Durchläufe seiner Hauptschleife pro Zeitintervall mit dem Anwachsen der Ausführungszeit eines Schleifendurchlaufs abnimmt. Einzelne Teilaufgaben können somit in diesem Teilsystem nicht unabhängig modelliert werden.

4.5 Zielsetzungen bei der Konzeption

Die Zielsetzungen bei der Konzeption einer verbesserten Steuerungssoftware ergeben sich zum einen aus den oben geschilderten Unzulänglichkeiten der bestehenden Lösung. Es ergibt sich hier also die Forderung nach einer Möglichkeit für eine bessere Synchronisation zwischen beiden Teilsystemen. Im Vordergrund steht bei der Konzeption allerdings eine leichtere Modifizierbarkeit der Steuerungssoftware als auch deren bessere Erweiterbarkeit. Hierunter ist zum einen die Möglichkeit einer möglichst unabhängigen Modellierbarkeit von bestehenden und vor allem zukünftigen Teilaufgaben der Software zu verstehen. Aber auch die Frage nach der Robustheit der Steuerungssoftware als Entwicklungsumgebung ist in dem Zusammenhang wichtig. Hier ist vor allem die bisherige Notwendigkeit zur Bestimmung von WCETs für eine rechnerische Überprüfung der Echtzeitfähigkeit, sowie die mit einer falsch angesetzten WCET verbundene Möglichkeit eines Systemversagens zu beanstanden.

Da der bestehende Programmcode bis auf die angesprochenen Nachteile erprobt ist und funktioniert erscheint es nicht wünschendwert, eine vollständige Neuimplementierung vorzunehmen. Dies bezieht sich einerseits auf die Steuerungssoftware. Andererseits sollte das Anwendungsprogramm auf PC-Seite so wenig wie möglich angepasst werden müssen. Der Grund hierfür ist die Größe und Komplexität des Moduls mit für das Rasterkraftmikroskop spezifischen Funktionen. Diese Zielvorstellung hat besondere Folgen für das Kommunikationskonzept, da die bestehende Lösung, speziell das Kommunikationsprotokoll, tief in der Anwendung verwurzelt ist. Leider wurde hier ursprünglich kein auf klar abgegrenzten Schichten basierendes Kommunikationsmodell verwendet. Das zu entwickelnde Kommunikationskonzept soll eine spätere Erweiterung der Steuerungssoftware für direkte Mikromanipulation und die hierfür notwendige Möglichkeit zur Übertragung von Daten mit niedrigen Latenzzeiten berücksichtigen.

[11] siehe auch die Grundlagen in Abschnitt 2.2.2

5 Analyse der Aufgaben der Steuerungssoftware

Die Konzeption eines Real-Time Software Systems für eine spezielle Anwendung umfasst v. a. das Aufteilen der von der Anwendung zu erledigenden Aufgaben in Teilaufgaben, die dann auf Rechenprozesse abgebildet werden können. Hierfür sind verschiedene Schritte notwendig. In diesem Kapitel wird anfänglich auf die Anwendungsszenarien des Gesamtsystems eingegangen. In diesem Kontext erfolg anschließend die Zerlegung des Ablaufs eines Scanvorgangs in Teilschritte und deren Untersuchung hinsichtlich der sich ergebenen Aufgaben der Steuerungssoftware. Grundsätzlich werden diese Aufgaben bereits von der existierenden Softwarelösung bearbeitet. Hierbei ist jedoch keine klare Einteilung in funktionale Gruppen oder Einzelaufgaben zu erkennen. Aus diesem Grund wurde dies hier basierend auf den Anforderungen der Anwendung neu erarbeitet.

5.1 Anwendungsszenarien des Gesamtsystems

Bei den Anwendungsszenarien handelt es sich zum einen um Aufgaben, die das Gesamtsystem mit der vorgefundenen Softwarelösung wenn auch nur eingeschränkt bereits erledigen kann. Zum anderen handelt es sich um mögliche Einsatzgebiete, für die zukünftig konkrete Lösungen erarbeitet werden sollen und die die Fähigkeiten des Gesamtsystems erweitern würden. Die Architektur des Steuerungssystems sollte also so ausgelegt sein, dass es diese entsprechenden Anwendungen bereits berücksichtigt oder sich später zumindest entsprechend leicht anpassen bzw. erweitern lässt.

5.1.1 Abbildung von Proben

Die Bilderfassung stellt die wesentliche Funktion des Geräts dar. Es wird hierbei durch langsames, zeilenweises abtasten der Probenoberfläche näherungsweise ein Höhenprofil aufgenommen. Abbildung 5.1 zeigt ein solches mit dem Gerät aufgenommenes Profil einer biologischen Probe. Das vorliegende Gerät ist vor allem für die Untersuchung biologischer Proben konzipiert. Merkmale in der Struktur der zu untersuchenden Proben können ab einer Größe von ca. 10-20 nm sichtbar gemacht werden.

Bei der Konzeption der Steuerungssoftware im Rahmen dieser Arbeit spielen diejenigen Einzelaufgaben der Software eine zentrale Rolle, die für die Abbildung von Proben benötigt werden. Aus diesem Grund werden die notwendigen Arbeitsschritte sowie die Aufgaben der Steuerungssoftware für dieses Anwendungsszenario in den folgenden Abschnitten 5.2 bis 5.4 detailliert untersucht.

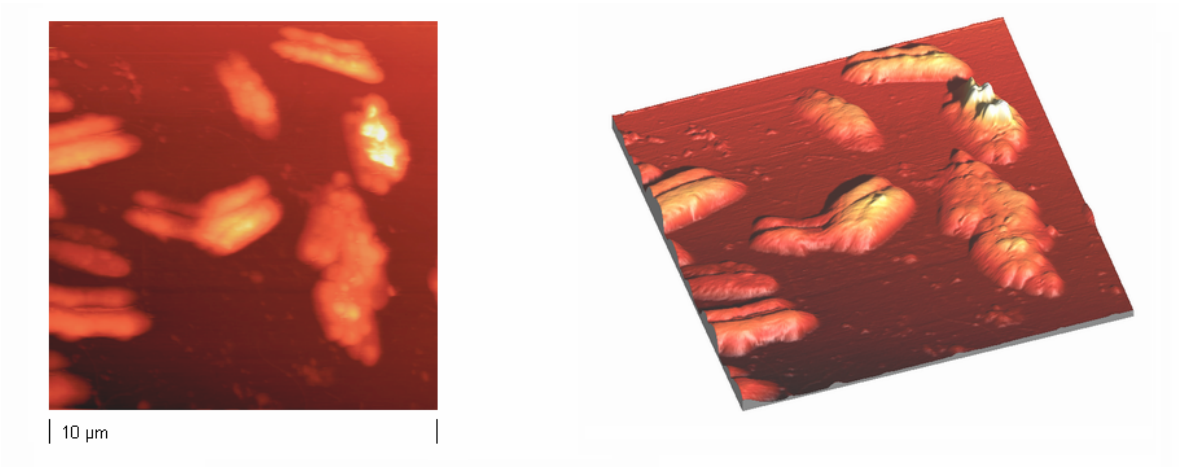


Abb. 5.1: Mit dem AFM aufgenommenes, topografisches Profil

5.1.2 Weitere Untersuchungen an Proben

Neben dem Aufnehmen eines topografischen Profils der Probenoberfläche sind noch weitere Untersuchungen an der Probe denkbar. Konkret sind punktuelle Elastizitätsmessungen an biologischem Gewebe geplant. Bei einer von [Radmacher 1997] beschriebenen Methode wird an einzelnen Stellen der Probe die Messspitze angenähert und teilweise in die Probe hineingedrückt und die resultierende Kraft-Abstands Kurve aufgenommen und ausgewertet.

5.1.3 Werkzeug zur Micromanipulation

Aufgrund der prinzipiellen Funktionsweise des AFM eignet es sich nicht nur zur Abbildung von Probenoberflächen oder zum Durchführen von weiteren Messungen, sondern auch als Werkzeug zur Mikro- bzw. Nanomanipulation. Hierbei wird die sehr feine Messspitze eingesetzt, um auf die Probe gezielt einzuwirken. Besonders im Bereich der medizinischen und biologischen Forschung wurden Rasterkraftmikroskope erfolgreich zur Manipulation von Proben eingesetzt. Vor allem Untersuchungen an Makromolekülen sind dabei in den Vordergrund getreten [Thalhammer et al. 1997]. Eine spezielle Möglichkeit für die Manipulation von Proben stellt die manuelle Interaktion des Benutzers mit der Probe unter zu Hilfenahme eines speziellen Force-Feedback Joysticks dar [Guthold et al. 1999]. Bei dieser Methode kann der Benutzer die Messspitze direkt steuern und die auftretenden Wechselwirkungen mit der Probe in Form von am Joystick auftretenden Kräften ertasten. Da eine solche direkte Steuerung

des vorliegenden Geräts geplant ist, werden die hierfür notwendigen Mechanismen bei den in dieser Arbeit vorgestellten Lösungsansätzen berücksichtigt.

5.2 Arbeitsschritte bei der Bilderfassung

Um in den folgenden Abschnitten Einzelaufgaben der Steuerungssoftware abzuleiten ist es notwendig, die in Abbildung 5.2 dargestellten Arbeitsschritte, die der Benutzer an dem System durchführt, und welche zur Erstellung einer Aufnahme notwendig sind, zu kennen.

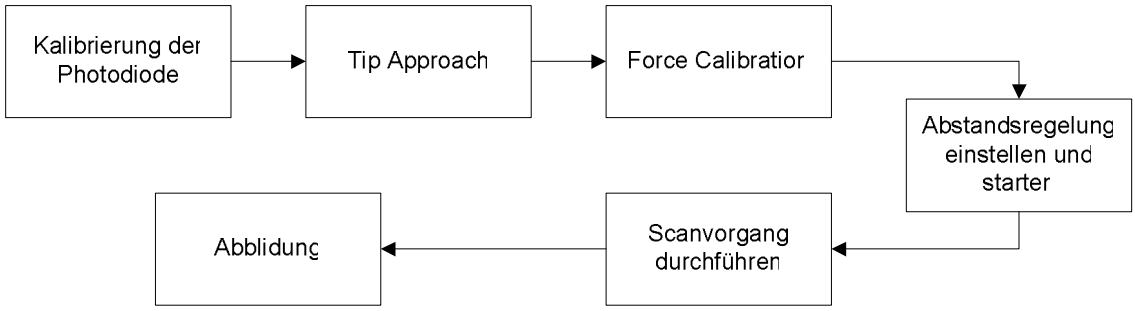


Abb. 5.2: Übersicht der Arbeitsschritte bei der Bilderfassung

Nach Einlegen der zu untersuchenden Probe, Auswählen des zu untersuchenden Bereichs mit dem angeschlossenen Lichtmikroskop, Aktivieren des Steuerrechners und Starten des Anwendungsprogramms auf dem PC ist das System betriebsbereit. Im ersten Schritt erfolgt nach dem Einschalten des Lasers die Kalibrierung der Photodiode. Dies ist notwendig, da sich durch äußere Einflüsse wie Erschütterungen oder Temperaturschwankungen der Strahlengang des Lasers ständig verändert. Das Anwendungsprogramm stellt dem Benutzer eine Funktion ähnlich einem Oszilloskop zur Verfügung, bei der der zeitliche Verlauf der vom Z-System gemessenen Spannungen der Photodiode angezeigt werden. Durch Beurteilung des Signals kann der Benutzer entsprechende manuelle Einstellungen direkt an der Mikroskop-Hardware vornehmen.

Im zweiten Schritt erfolgt die Annäherung der Probe an die Messspitze, der sog. Tip Approach. Es werden sowohl die Z-Piezos als auch die Z-Linearmotoren eingesetzt. Der Prozess stoppt automatisch wenn ein vom Benutzer vorgegebener Wert für die Auslenkung des Cantilevers erreicht wurde.

Ein weiterer Schritt besteht darin, eine Kraft-Abstands Kurve aufzunehmen. Hierfür wird die Probe mit Hilfe der Z-Piezos so weit wie möglich von der Messspitze entfernt und anschließend dieser wieder genähert. Der Verlauf des Ausschlags des Cantilevers

wird in Form von dem von der Photodiode gelieferten Spannungsverlauf aufgezeichnet und von der Anwendungssoftware grafisch dargestellt. Der Benutzer kann aus der Kurve verschiedene Rückschlüsse ziehen, z. B. ob die vorangegangene Annäherung erfolgreich war.

Im vierten Schritt wird in Vorbereitung eines Scanvorgangs die Abstandsregelung der Messspitze gestartet. Der Benutzer gibt dafür verschiedene Regelparameter vor und bestätigt die Aktivierung der Regelung. Deren korrekte Funktion wird dabei mit Hilfe der bereits im ersten Schritt benutzten Oszilloskop-Funktion beurteilt.

Den letzten Schritt stellt der eigentliche Scanvorgang dar. Auch hierfür kann der Benutzer noch verschiedene Parameter vorgeben, z. B. die Geschwindigkeit, mit der abgetastet werden soll, und die Auflösung des resultierenden Bildes. Nach dem Starten des Vorgangs wird die Probe zeilenweise abgetastet, wobei die Ergebnisse der Messung grafisch dargestellt werden.

Für den Fall, dass im dynamischen Modus gearbeitet werden soll, erfolgt zwischen dem ersten und zweiten Schritt noch das einstellen des Lock-In Verstärkers des Z-Systems auf die Resonanzfrequenz des Cantilevers. Der Lock-In Verstärker verfügt hierfür über einen programmierbaren Frequenzgenerator. Dieser Fall soll im Folgenden nicht näher betrachtet werden. Das Einstellen des Lock-In Verstärkers würde eine weitere, unabhängige Funktionsgruppe der Steuerungssoftware darstellen.

5.3 Identifikation von Funktionsgruppen der Steuerungssoftware

Entsprechend der im vorangegangen Abschnitt geschilderten Schritte beim Arbeiten mit dem Mikroskop ergeben sich spezifische Aufgaben der Steuerungssoftware, die sich in mehrere unabhängige Funktionsgruppen einteilen lassen. Hierbei handelt es sich nicht um eine Möglichkeit zur Gliederung der Steuerungssoftware in einzelne Tasks sondern um ein Hilfsmittel zum besseren Verständnis der mit Hilfe der Software zu realisierenden Funktionen sowie von deren Zusammenhängen.

Es ist zu berücksichtigen, dass einige dieser Funktionen gleichzeitig aktiv sein können, andere sich jedoch gegenseitig ausschließen. Tabelle 5.1 stellt diese Zusammenhänge dar. Sie ist dabei nicht symmetrisch. Beispielsweise ist beim Ausführen der Scanbewegung das Oszilloskop optional, die Abstandsregelung muss jedoch in jedem Fall aktiv sein. Umgekehrt muss, während die Abstandsregelung aktiv ist, nicht unbedingt eine Scanbewegung ausgeführt werden.

Tab. 5.1: Erlaubte Kombinationen von Funktionsgruppen

A \ B	Oszilloskop	Tip Approach	Force Calibration	Abstands-regelung	Abtasten einer Linie	Scan-bewegung
Scan-bewegung	●	○	○	◙	◙	–
Abtasten einer Zeile	●	○	○	◙	–	◙
Abstands-regelung	●	○	○	–	●	●
Force Calibration	●	○	–	○	○	○
Tip Approach	●	–	○	○	○	○
Oszilloskop	–	●	●	●	●	●

● wenn A aktiv ist kann auch B aktiv sein
◙ wenn A aktiv ist muss B auch aktiv sein
○ wenn A aktiv ist darf B nicht aktiv sein

5.3.1 Oszilloskop

Die Oszilloskop-Funktion bietet die Möglichkeit, den zeitlichen Verlauf verschiedener Größen des XY- oder Z-Systems dem Benutzer zugänglich zu machen. Bei diesen Größen handelt es sich z. B. um die Ausgangsspannungen der Photodiode beim Z-System. Es ist allerdings vorgesehen, nicht nur die Eingangsspannungen aller beschalteten ADCs sondern wahlweise auch die über die DACs ausgegebenen Werte aufzuzeichnen.

Die für die Realisierung des Oszilloskops notwendigen Abläufe im XY- bzw. Z-System sind in Abbildung 5.3 dargestellt. Das Anwendungsprogramm auf dem PC gibt Parameter für die aufzuzeichnenden Signale sowie die Länge des Aufzeichungsintervalls vor und startet den Vorgang auf dem DSP. Es werden hier nun als erstes Messwerte über den vorgegebenen Zeitraum, der maximal einige hundert Millisekunden lang ist, in einem Puffer gesammelt.

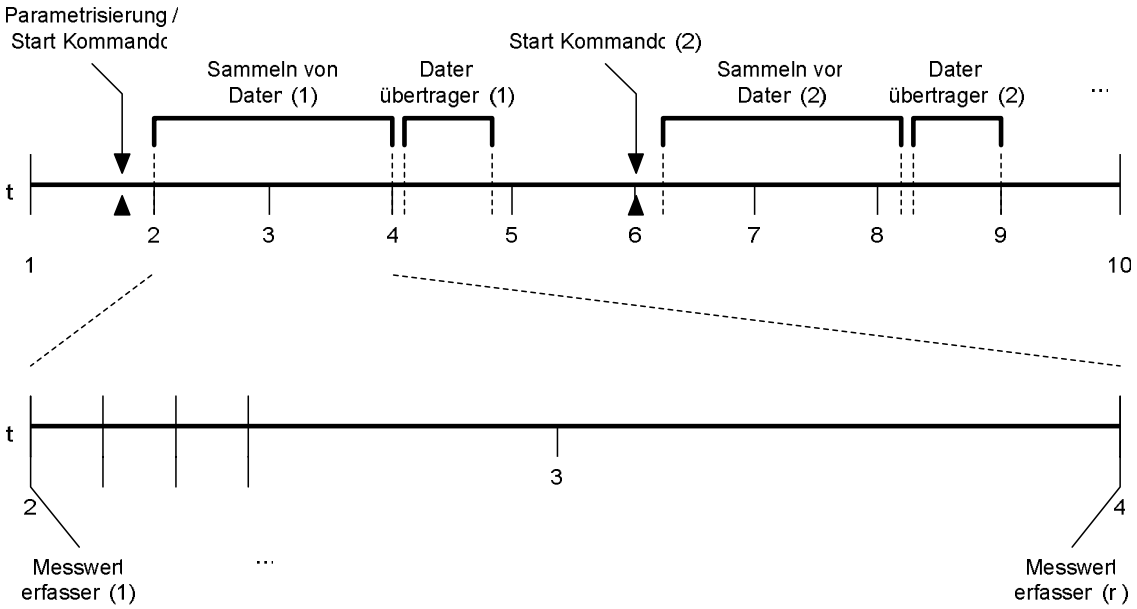


Abb. 5.3: Zeitlicher Ablauf der Oszilloskop-Funktion

Die Erfassung der Messwerte erfolgt dabei in Perioden, deren Länge ein entsprechendes Vielfaches der durch die Abtastrate der ADCs vorgegeben Periodenlänge ist. Nach Ablauf des Aufzeichungsintervalls werden die Messwerte zum PC übertragen und dort als Kurvenverlauf dargestellt. Nachdem der PC die Daten also verarbeitet hat stößt er die Aufzeichnung der nächsten Messwertreihe an durch den DSP an. Dieser Vorgang wiederholt sich kontinuierlich ca. zwei bis fünf Mal pro Sekunde. Alternativ zum Sammeln der Werte in einem Puffer ist je nach dem mit welcher Frequenz neue Messwerte anfallen auch eine direkte Übertragung an den PC denkbar.

5.3.2 Tip Approach

Die Annäherung der Probe an die Messspitze stellt eine wesentliche Funktion des Z-Systems dar, die von diesem weitgehend selbstständig ausgeführt wird.

Der Benutzer gibt einen Schwellwert für die Durchbiegung des Cantilevers in Form eines Spannungswertes der Photodiode vor und startet den Vorgang der Annäherung. Dieser besteht aus zwei Schritten, die sich fortlaufend wiederholen, bis der vorgegebene Spannungswert an der Photodiode erreicht oder überschritten wurde. Im ersten, dem Annäherungsschritt, wird die Probe mit Hilfe der Z-Piezos schrittweise in Richtung Messspitze bewegt. Nach jedem Schritt wird die Spannung an der Photodiode gemessen und mit dem Schwellwert verglichen und gegebenenfalls abgebrochen. Die Periodenlänge dieser Schritte wird ebenfalls durch die Abtastrate der ADCs vorgegeben.

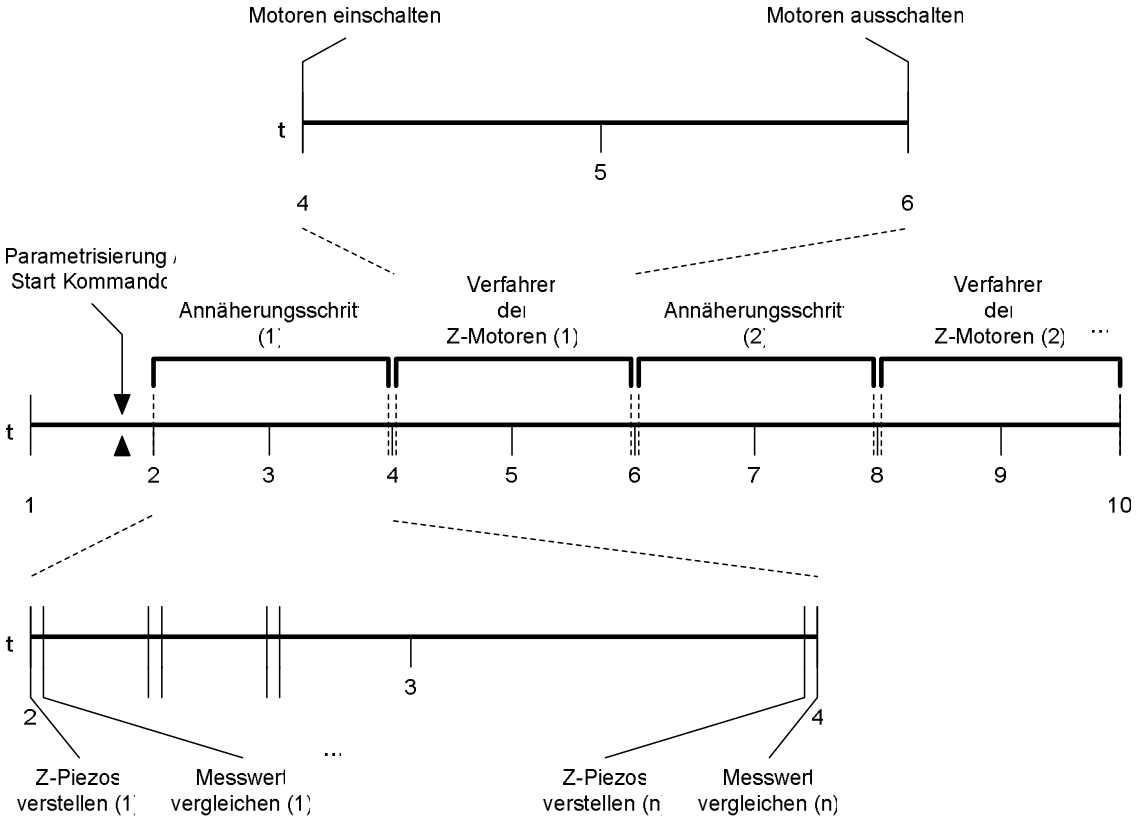


Abb. 5.4: Zeitlicher Ablauf des Tip Approach

Falls die Z-Piezos während des Annäherungsschritts ihre maximale Auslenkung erreicht haben, aber der Schwellwert nicht überschritten wurde, so ist davon auszugehen, dass die Entfernung zur Messspitze noch zu groß ist. Es werden nun im zweiten Schritt die Z-Linearmotoren benutzt, die den gesamten Aufbau näher an die Messspitze heranfahren. Dafür sind diese ein-, und nach einer definierten Zeit wieder auszuschalten. Diese liegt im Bereich von mehreren Millisekunden.

5.3.3 Force Calibration

Bei der sog. Force Calibration wir eine Kraft-Abstands Kurve aufgenommen. Das Verfahren ähnelt einem Annäherungsschritt beim Tip Approach. Um Beschädigungen an der Hardware und der Probe zu vermeiden, wird ebenfalls ein Schwellwert für die Auslenkung des Cantilevers vorgegeben, bei dem die Annäherung abgebrochen wird.

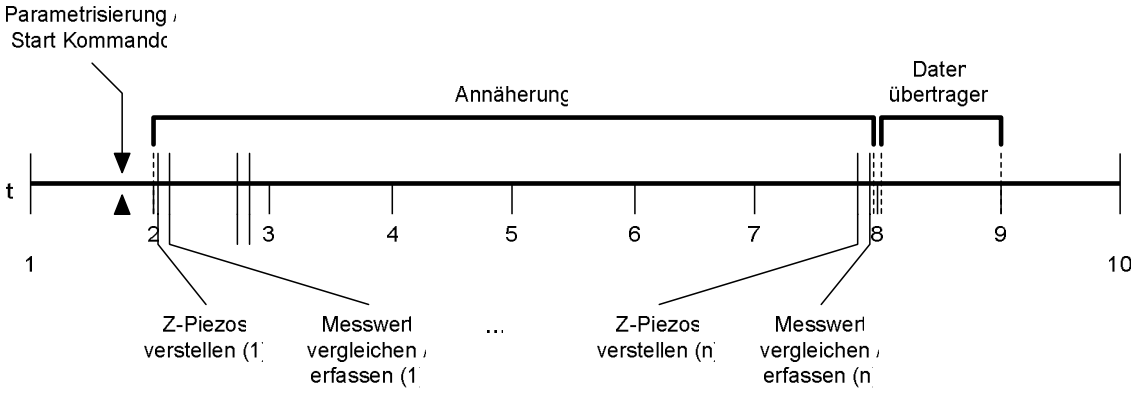


Abb. 5.5: Aufnehmen der Kraft-Abstands Kurve

Die Messwerte werden zusätzlich jedoch in einem Puffer gesammelt und nach Beendigung der Annäherung zum PC übertragen, wo sie dann grafisch dargestellt werden. Alternativ ist es wiederum möglich, die Messwerte direkt während der Annäherungsphase zu übertragen.

5.3.4 Digitale Abstandsregelung

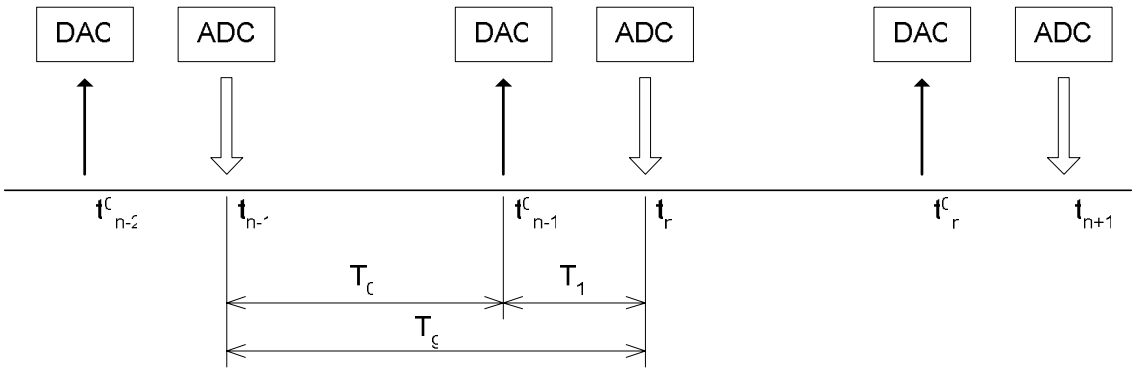
Die Abstandsregelung stellt den wichtigsten Teil der Aufgaben des Z-Systems dar. Das kontinuierliche Messen der Auslenkung des Cantilevers und das Nachführen seiner Position senkrecht zur Probe verhindert nicht nur Kollision der Messspitze mit der Topografie der Probe, sondern ist der zentrale Kern des bildgebenden Verfahrens.

Das Problem der Abstandsregelung kann für den Fall des statischen Modus formuliert werden als Problem, den Kontakt mit einer sich bewegenden Oberfläche zu halten, wobei die auftretende Kraft zwischen Oberfläche und Sonde konstant gehalten werden soll. Hierfür ist eine Regelung notwendig.

Obwohl die Regelung zeitdiskret mithilfe des DSP stattfinden soll, kann ein quasikontinuierliches Regelverhalten als Ziel gesehen werden. Das heißt, dass die durch zeitliche Diskretisierung sowie durch Quantisierung der beteiligten Größen auftretenden Fehler das Verhalten der Regelung nur vernachlässigbar beeinflussen. Der Einfluss auf das Verhalten der Regelung kann als Abweichung vom angenommenen Verhalten bei Realisierung mit einem kontinuierlichen Regler gesehen werden.

Eine zentrale Rolle spielt bei jedem Regler die sog. Regelgröße. Hierbei handelt es sich im dem Fall um den Ausschlag des Cantilevers. Der Ausschlag des Cantilevers wird mittels des Lichtzeigerprinzips gemessen. Die dabei aus dem Photostrom an der Photodiode entstehende Spannung repräsentiert den Ist-Wert und ergibt damit die sog.

Rückführgröße. Diese wird mit einem Sollwert, der sog. Führungsgröße verglichen. Der Sollwert wird normalerweise vom Benutzer während der Einstellung der Regelung vorgegeben. Die Differenz von Soll- und Ist-Wert ist die Regelabweichung und wird vom Regelalgorithmus verwendet, um eine Ausgangsgröße zu ermitteln, die einer Stelleinrichtung zugeführt wird welche die Regelgröße beeinflusst. Hierbei handelt es sich um die Spannung, die die Ausdehnung der Z-Piezos zur Folge hat. Außerdem wird die Regelgröße durch externe Einflüsse gestört. Dabei handelt es sich im Idealfall um Änderungen in der Topographie der Probe, während diese unter der Messspitze vorbeigeführt wird. Ziel der Regelung ist es, den Ist- mit dem Soll-Wert in Übereinstimmung zu halten. Die Funktionsweise ist in Abbildung 5.6 dargestellt.



vgl.: [Gorinevsky et al. 1997, S.225]

Abb. 5.6: Prinzip der digitalen Regelung

Zu Begin eines jeden Kontrollzyklus, der beginnend zum den Zeitpunkten t_i genau T_g Zeit in Anspruch nimmt, muss das Kontrollprogramm den aktuell gemessenen Wert der Regelgröße über einen ADC (Analog-Digital-Converter) einlesen. Unter Berücksichtigung dieses Wertes wird dann die Ausgangsgröße des Reglers berechnet und anschließend über einen DAC (Digital-Analog-Converter) ausgegeben. Der gesamte Vorgang nimmt die Zeit T_0 in Anspruch. Nach Ablauf der Zeit T_1 beginnt der nächste Kontrollzyklus.

Die Anstandsregelung wird vom Benutzer gestartet. Außerdem gibt dieser vorher hierfür notwendige Parameter vor. Die Regelung läuft bis sie wieder explizit gestoppt wird. Das Zeitintervall T_g wird wiederum durch die Abtastrate der ADCs vorgegeben.

5.3.5 Abtasten einer Zeile

Das Abtasten einer Zeile ist Teil des Messvorgangs und wird vom Z-System während des Scannens ausgeführt. Der Ablauf ist ähnlich dem bei der Oszilloskop-Funktion. Auch hier werden vom Anwendungsprogramm verschiedene Parameter, wie die Länge des Zeitraums, in dem abgetastet werden soll, sowie die Anzahl der aufzunehmenden Pixel pro Zeile übertragen. Die Dauer der Abtastung wurde vorher aus der vom Benutzer vorgegebenen Abtastgeschwindigkeit und der Länge der abzutastenden Zeile berechnet. Anschließend erfolgt, nachdem der Start signalisiert wurde, das Sammeln von Daten über den vorgegebenen Zeitraum hinweg. Gesammelt werden die Werte der Ausgangsgröße des Reglers sowie bei Bedarf die der Regelabweichung. Die Intervalllänge, in der diese Werte ermittelt und entweder direkt an den PC übertragen oder in einem Puffer für die anschließenden Übertragung gespeichert werden, ergibt sich aus der Dauer der Abtastung, der Anzahl der aufzunehmenden Pixel sowie der zwecks Rauschunterdrückung pro Pixel aufzusummierenden Messwerte. Da aufgrund der mechanischen Eigenschaften des Systems sehr hohe Scangeschwindigkeiten nicht sinnvoll sind, lässt sich die Anzahl der abgetasteten Zeilen auf ca. 5 pro Sekunde beschränken.

5.3.6 Ausführen der Scanbewegung

Das ausführen der Scanbewegung ist die Hauptaufgabe des XY-Systems. Dabei wird gleichzeitig zum Abtasten einer Zeile durch das Z-System die Probe in der XY-Ebene so verschoben, dass die vom Z-System aufgenommenen Messwerte denen einer bestimmten Linie auf der Probe entsprechen.

Nachdem die Parameter für Start- und Endpunkt der Linie sowie für die Verfahrgeschwindigkeit übertragen wurden, wird mit den Piezoaktoren die Startposition der Linie angefahren. Sobald der Start signalisiert wurde, beginnt dann das langsame Abfahren dieser Linie. Dazu wird die Stellung der entsprechenden Piezoaktoren in Intervallen geändert, deren Länge wiederum durch die Abtastrate der ADCs des Systems vorgegeben werden. Dabei ist unerheblich, dass die ADCs in der momentanen Konfiguration gar keine verwertbaren Messwerte liefern, sondern nur als hochauflösender Timer eingesetzt werden.

Nach Beendigung einer Linie wiederholt sich der gesamte Vorgang bei Bedarf für die nächste Linie.

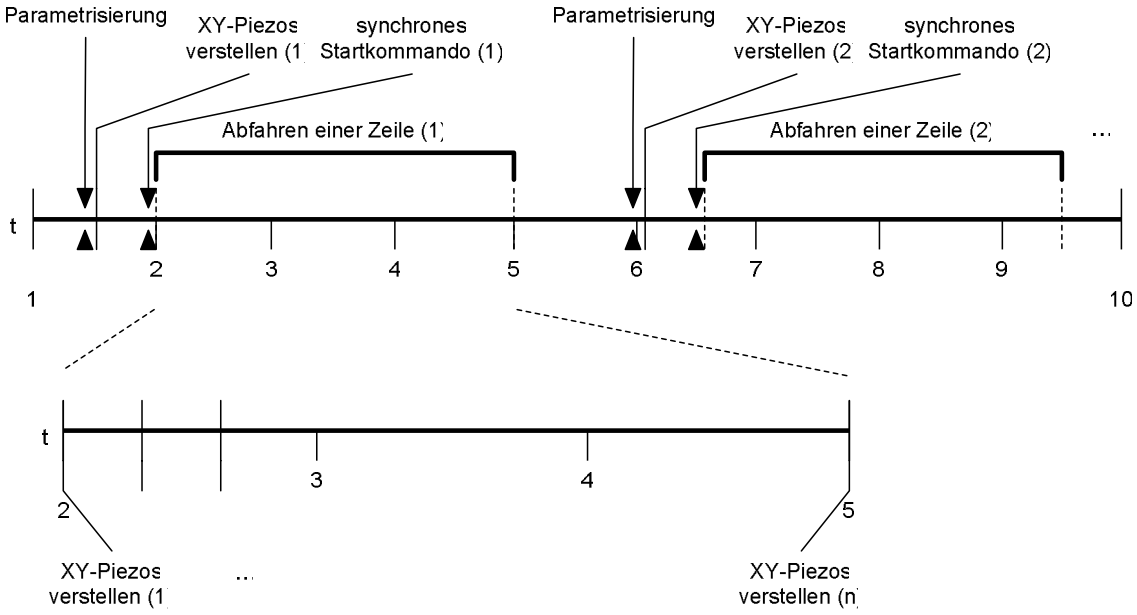


Abb. 5.7: Ablauf der XY-Scanbewegung

5.3.7 Weitere Aufgaben

Neben den Aufgaben, die in den vorausgegangenen Kapiteln in sechs Funktionsgruppen eingeteilt wurden, existieren noch weitere kleinere Aufgaben, die bearbeitet werden müssen. Hierbei handelt es sich durchweg um das direkte Ansprechen einzelnen Teile der Hardware bzw. um das auslesen von Werten der ADCs.

Zu diesen Aufgaben gehören beispielsweise das mittels des Anwendungsprogramms realisierte ein- und ausschalten des Lasers oder das manuelle Betätigen der einzelnen Z-Linearmotoren. Weiterhin ist es für Diagnosezwecke vorgesehen, dass der Benutzer von dort aus die aktuellen Werte aller ADCs auslesen und auf allen DACs sowie den digitalen Ausgängen Werte ausgeben kann. Für diese Zwecke sind keine sich über einen längeren Zeitraum erstreckenden Abläufe innerhalb der DSP-Software auszuführen. Sobald ein entsprechendes Kommando empfangen wird, kann dieses unmittelbar ausgeführt und damit abgeschlossen werden.

5.4 Zusammenfassung der zeitlichen Anforderungen der Teilaufgaben

Je nach aktiver Funktionsgruppe müssen unterschiedliche Teilaufgaben ausgeführt werden. Bei der Oszilloskop-Funktion sind dies das Sammeln von Daten, indem periodisch Messwerte erfasst werden, und das anschließende Übertragen der gesammelten Daten. Dieser Vorgang wiederholt sich dabei etwa alle 200-500 ms und

wird frühestens erst dann durch ein Kommando vom Anwendungs-PC erneut angestoßen, wenn die Übertragung der Messwerte abgeschlossen ist. Beim Tip Approach werden im Annäherungsschritt zusätzlich zur Messwerterfassung nach jedem neuen Messwert auch die Piezoaktoren verstellt. Die Messwerte werden gesteuert durch einen Interrupt periodisch mit einer Periodenlänge von 10 µs erfasst. Im folgenden Schritt des Verfahrens der Linearmotoren wird im Wesentlichen eine bestimmte Anzahl Messperioden abgewartet und vorher bzw. nachher die Motorspannung ein- bzw. ausgeschaltet. Bei der Abstandsregelung wird hingegen kontinuierlich die Regelabweichung periodisch bestimmt und nach Anwendung eines Regelsalgorithmus die Ausgangsgröße des Reglers angepasst, also die Piezoaktoren angesteuert. Die bei den anderen Funktionen auftretenden zeitlichen Anforderungen sind analog zu den denen bei den eben geschilderten Fällen.

Es ist zu erkennen, dass in den einzelnen Funktionsgruppen jeweils eine Teilaufgabe auftreten kann, die periodisch ist und eine hohe Frequenz hat. Diese Teilaufgaben sind alle mit den ADCs gekoppelt und werden beim Auftreten eines neuen Messwerts ausgeführt. Sie arbeiten dabei mit einer Frequenz von 100 kHz, welches der angesprochenen Periodenlänge von 10µs entspricht.

Weitere Aufgaben umfassen v. a. das Übertragen von gesammelten Messwerten. Dies ist beim Abtasten einer Zeile, der Force Calibration und der Oszilloskop-Funktion der Fall. Alle diese weiteren Aufgaben fallen klar erkennbar sporadisch oder pseudo-periodisch, ausgelöst durch Befehle des Anwendungs-PCs an. Dabei ist die Periodenlänge deutlich größer und liegt im Bereich von über 100 ms. Weiterhin kann beim Abtasten einer Linie, dem Tip Approach, der Oszilloskop-Funktion und der Force Calibration auf die Fertigstellung der Übertragung von Messwerten zum PC gewartet werden, bevor das Sammeln neuer Messwerte angestoßen wird. Damit ist die Übertragung nicht zeitkritisch. Bei der Oszilloskop-Funktion soll die Durchführung und Übertragung von Messungen zwar in regelmäßigen Intervallen erfolgen. Verzögerungen hier sind zwar nicht wünschenswert aber ebenfalls nicht kritisch, da der Anwendungs-PC auf die vollständige Übertragung einer Messwertreihe wartet bis er die Erfassung der nächsten per Befehl anstößt.

5.5 Geplante zukünftige Aufgaben

Mögliche zukünftige Aufgaben ergeben sich aus den bereits genannten weiteren Anwendungsszenarien, jenseits der reinen Abbildung von Proben.

Die Vorgehensweise bei der örtlichen Elastizitätsmessung ähnelt sehr stark der bei der bereits geschilderten Force Calibration. Auch hier wir eine Kraft-Abstands Kurve aufgenommen. Dementsprechend könnten die Funktionen der Steuerungssoftware zur Force Calibration mit minimalen Änderungen auch für die Elastizitätsmessung herangezogen werden. Für die Auswertung der Kraft-Abstands Kurve unter dem Aspekt der Ermittlung von entsprechenden Kennzahlen für die Elastizität müsste dann lediglich die Anwendungssoftware auf dem PC erweitert werden.

Bei der direkten Mikromanipulation mit Hilfe eines Joysticks mit Kraftrückkopplung treten jedoch neue Aspekte in den Vordergrund, die bisher nicht ausreichend berücksichtigt worden sind. Um diese Art der Steuerung zu realisieren ist es notwendig, die vom Benutzer vorgegebenen Änderungen der Position der Spitze zeitnah in reale Positionsänderungen umzusetzen. Die im Rahmen von Diagnosezwecken vorgesehenen Möglichkeiten zum direkten Ansprechen einzelner Teile der Hardware bilden hierfür bereits eine ausreichende Grundlage. Des Weiteren müssen aber auch Rückmeldungen, v. a. über die ertastete Topografie periodisch, und mit geringen Verzögerungen zurückgeliefert werden.

6 Konzeption der Steuerungssoftware

Die im vorangegangenen Kapitel identifizierten, in Funktionsgruppen eingeteilten und auf ihre zeitlichen Anforderungen hin untersuchten Aufgaben sollen zur Gewährleistung der Basisfunktionalität des Systems von der Steuerungssoftware natürlich unterstützt werden. Weiterhin vorgestellte spezielle oder zukünftige Aufgaben sind dabei hinsichtlich späterer Weiterentwicklungen oder Erweiterungen von Interesse. Dabei muss ihr Einsatz aber schon jetzt Beachtung finden.

Ausgehend vom Stand der bereits vorgefundenen Softwarekomponenten sowie unter Berücksichtigung von zukünftigen Erweiterungen werden in diesem Kapitel Ansätze zur Realisierung der Steuerungssoftware vorgestellt. Unter Berücksichtigung der bereits gewonnenen Erkenntnisse aus dem letzten Kapitel wird ein Konzept für das Scheduling von Rechenprozessen abgeleitet, mit deren Hilfe die Aufgaben später implementiert werden sollen.

6.1 Prinzipielle Lösungsansätze

Beim grundlegenden Design der Steuerungssoftware können unter Beachtung der Zielsetzungen aus Kapitel 4.4 verschiedene Lösungsstrategien zum Einsatz kommen. Ein Ansatz wäre, das vorgefundene Foreground/Background-System weitestgehend beizubehalten, die gewünschten Erweiterungen bzw. Veränderungen vorzunehmen und dann durch Monitoring der Aktivitäten des Interrupthandlers und des Hintergrundsystems zu beobachten. Überlastsituationen könnten so erkannt und Informationen darüber dem Softwareentwickler in der Testphase zur Verfügung gestellt werden. Es wäre in diesem Szenario außerdem möglich, das System automatisch in einen sicheren Zustand zu versetzten um Fehlfunktionen zu vermeiden. Dieser Ansatz würde es ermöglichen, die bestehende Programmstruktur größtenteils beizubehalten. Eine solche Lösung hat jedoch den Nachteil, dass nicht garantiert werden kann, dass eventuelle Überlastsituationen bereits in der Testphase entdeckt werden, und dass diese dann später beim Anwender auftreten.

Um die Antwortzeiten wichtiger Teilaufgaben des Hintergrundsystems, wie z. B. der Kommunikation, durch Erweiterungen des Systems nicht unkontrollierbar zu verlängern, bietet sich ein Aufteilen der Funktionalität in einzelne Teilaufgaben an. Diese Tasks können mit unterschiedlichen Prioritäten versehen und unabhängig voneinander ausgeführt werden. In Abschnitt 5.4 wurde festgestellt, dass es Aufgaben gibt, die mit einer Periode von kurzer Dauer ausgeführt werden müssen. Diese sind außerdem zeitkritisch. Eine Lösungsmöglichkeit besteht darin, diese in einem Task

hoher zusammenzufassen. Andere Aufgaben sind nicht zeitkritisch. Diese werden mit Hilfe von Tasks niedrigerer Priorität implementiert. Für die korrekte Funktion des Systems ist es dann hinreichend, wenn die Tasks niedrigerer Priorität nicht vollständig von den Tasks mit höherer Priorität verdrängt werden.

Bei der Nutzung von unterschiedlichen Tasks kommt die Frage nach deren konkreter Umsetzung auf. Es bietet sich hier der Einsatz von entsprechender Systemsoftware an. Die Hauptaufgabe gängiger Systemsoftware für Echtzeitanwendungen ist es, die Aufteilung einer Anwendung in mehrere Prozesse oder Tasks zu ermöglichen und diese dann zu verwalten. Sie beschäftigt sich bei der Verwaltung v. a. mit dem Zuteilen von CPU-Zeit an einzelne Tasks, also dem Scheduling. Der Einsatz von Systemsoftware erleichtert außerdem die Entwicklung der Anwendung, da sie auf Funktionen, die bereits von der Systemsoftware bereitgestellt werden, zurückgreifen kann. Dies führt beiläufig zu einem gewissen Grad an Unabhängigkeit von der zugrunde liegenden Hardware, da die Anwendung ohne große Änderungen auf andere Plattformen, für die die verwendetet oder eine kompatible Systemsoftware verfügbar ist, portiert werden kann. Dieser Vorteil dürfte aber für die hier betrachtete Anwendung keine Rolle spielen. Ein Nachteil beim Einsatz zusätzlicher Systemsoftware ist dagegen der Mehrbedarf an Speicher sowie eine stärkere CPU-Auslastung durch den Overhead bei der Verwaltung der einzelnen Tasks.

6.1.1 Auswahl geeigneter Systemsoftware

Für das vorliegende System steht nur wenig Systemsoftware zur Auswahl. Dies liegt auf der einen Seite an der geringen Verbreitung der Plattform. Auf der anderen Seite sind die Ressourcen des DSP sehr beschränkt. Besonders der kleine Speicher von 128 kB erweist sich für Systeme wie RT-Linux als unzureichend. Der Hersteller liefert für diesen DSP eine Bibliothek mit, die zwar kein Prozessmanagement unterstützt, aber die zumindest Funktionen zum Interruptdispatching anbietet. Der DSP verfügt ausserdem hardwareseitig bereits über Fähigkeiten für Interruptnesting sowie für Interrupts mit unterschiedlicher Priorität. Damit lässt sich bei Verwendung mehrerer interruptgetriggerter Tasks leicht ein einfaches prioritätenbasiertes System realisieren. Alternativ zu der Implementierung einer eigenen Systemsoftware gibt es aber selbst für die verwendete Prozessorfamilie noch mehrere Lösungen von Drittanbietern.

Diese Systeme bieten alle ähnliche Funktionalität hinsichtlich Prozessverwaltung, Scheduling, Interprozesskommunikation und Synchronisation. Teilweise sind auch Zusatzmodule wie z. B. Dateisystemtreiber verfügbar. Obwohl es Ansätze für

Schedulingalgorithmen gibt, die ihre Planungen auf der Grundlage der einzuhaltenden Zeitbedingungen durchführen, sind in der Praxis normalerweise preemptive, prioritätenbasierte Ansätze vorzufinden. [Gallmeister 1995, S. 150] Dabei liegt die Verantwortung für die Einhaltung von Zeitbedingungen im Wesentlichen in dem Design der Anwendung, d. h. an der geeigneten Parametrisierung von Prozessprioritäten im Zusammenhang mit dem Einsatz von Intervalltimern oder Ähnlichem.

Im Rahmen dieser Arbeit soll MicroC/OS-II [Labrosse 2002] als Systemsoftware zum Einsatz kommen. Dieses System ist für einen dem ADSP-21061 ähnlichen Prozessor verfügbar und bietet bereits neben prioritätenbasierten, preemptivem Multitasking alle wesentlichen Funktionen zur Prozesssynchronisation und -kommunikation. Eine Alternative zu µC/OS-II wäre Nucleus PLUS [Mentor Graphics 2004]. Für die Wahl von µC/OS-II war neben der Verfügbarkeit des Quelltextes die außerdem verfügbare, sehr gute Dokumentation entscheidend. Dieses System und seine Funktionen wurden bereits in Abschnitt 3.3.5 genauer vorgestellt.

6.1.2 Einsatz von MicroC/OS-II im Rahmen dieser Arbeit

Die von µC/OS-II gebotenen Möglichkeiten bilden eine gute Grundlage für die angestrebte Gliederung von Abläufen innerhalb der Steuerungssoftware in Teilaufgaben und deren Implementierung mit Hilfe von mehreren Rechenprozessen. Dem System fehlen allerdings noch wesentliche Funktionen, die für das vorliegende Problem von Bedeutung sind. Dabei handelt es sich um das Fehlen von Möglichkeiten zum Behandeln von Unterbrechungsanforderungen, wie sie von den ADCs ausgelöst werden. Die Dokumentation begnügt sich auf die Aussage, dass die Anwendung für die Implementierung von Interrupthandlern sorgen muss. Es sind jedoch bereits Vorkehrungen vorhanden, um das Aufrufen einer Reihe von Systemfunktionen aus einem Interrupthandler heraus zu ermöglichen. Die Kopplung einer Unterbrechungsanforderung mit einem regulären Task ist nicht jedoch vorgesehen. Ein weiterer Punkt ist, dass bei der Implementierung von bestimmten Teilaufgaben wie z. B. der Abstandsregelung mittels der angebotenen Tasks der prozentuale Anteil der Prozessorzeit, die für Kontextwechsel verwendet wird, stark ansteigt. Beide Probleme sollen durch eine Erweiterung des Systems gelöst werden.

6.2 Das Konzept des Supertasks

Im Rahmen von DSP-Anwendungen werden oft in kurzen Abständen Daten von einem Gerät wie einem ADC bereitgestellt, welche umgehend verarbeitet werden müssen. Hierfür ist eine schnelle Reaktion auf Unterbrechungsanforderungen notwendig. MicroC/OS-II bietet dafür leider keine eigene Lösung an. Es fallen jedoch auch im vorliegenden System Aufgaben an, die periodisch mit hoher Frequenz und von einem Interrupt angestoßen ausgeführt werden müssen. Als Lösung bietet sich eine Ergänzung des Betriebssystems um Möglichkeiten zur Verwaltung solcher Tasks an. Im Hinblick auf die in den folgenden Abschnitten beschriebene Erweiterung des Ansatzes erscheint der Einsatz des bereits in Abschnitt 2.3.4.3 beschriebenen SharcOS unter Berücksichtigung des damit einhergehenden erhöhten Ressourcenbedarfs als nicht sinnvoll. Es wird vielmehr eine eigene, schlanke Erweiterung angestrebt.

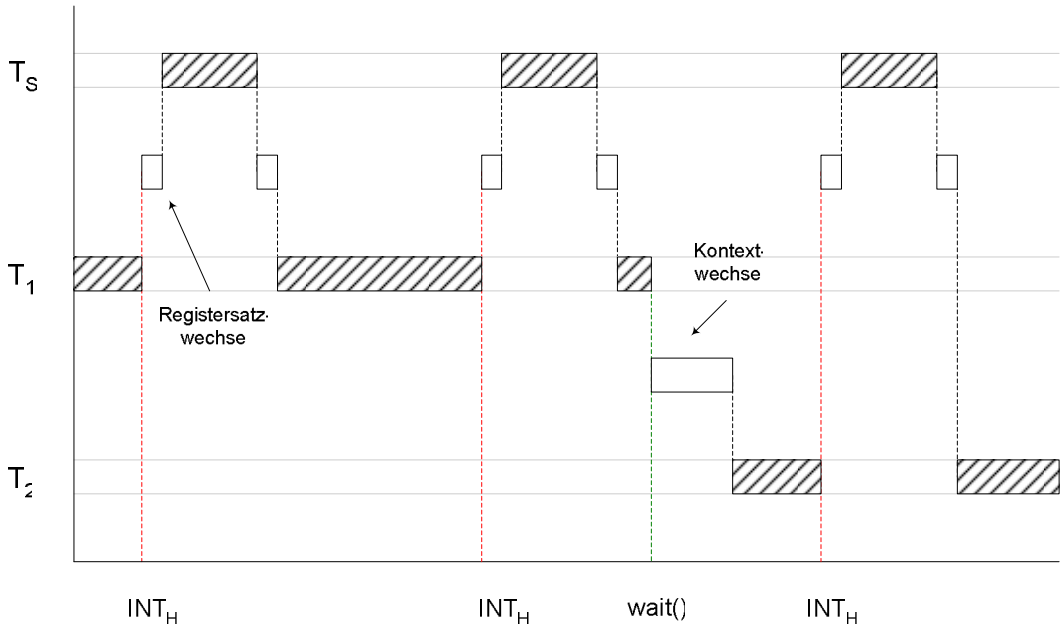


Abb. 6.1: Konzept des Supertasks

Bei der Erweiterung wurden zwei Besonderheiten des vorliegenden Systems berücksichtigt. Es gibt zum einen, wie bereits angesprochen, pro Teilsystem genau einen Interrupt der periodisch mit hoher Frequenz auftritt und behandelt werden muss. Zum anderen verfügt der Prozessor über einen zweiten Registersatz, der alternativ zum standardmäßig aktivierten benutzt werden kann. Dies ermöglicht die Einführung eines einzelnen Tasks welcher besonders schnell auf Unterbrechungsanforderungen reagieren kann.

Bei der Aktivierung dieses Supertasks erfolgt dann ein Wechsel des Registersatzes wodurch es nicht notwendig ist, die vergleichsweise große Menge an Registern des SHARC-DSP auf dem Stack zu sichern und anschließend von dort wiederherzustellen. Der Supertask hat außerdem effektiv die höchste Priorität. Durch das Auftreten des ihn auslösenden Interrupts wird jeder andere Task verdrängt. Außerdem werden Vorkehrungen getroffen, damit er während seiner Ausführung nicht verdrängt werden kann.

Abbildung 6.1 illustriert das Systemverhalten bei der Verwendung eines Supertasks T_S der durch den Interrupt INT_{HI} ausgelöst wird. Dieser verdrängt den Task mit zweithöchster Priorität T_1. Der hierfür nötige Wechsel des Registersatzes nimmt nur eine sehr kurze Zeit in Anspruch. Nach Beendigung des Supertasks wird der zuletzt ausgeführte Task fortgesetzt. Ein normaler Taskwechsel findet anschließend statt, indem T_1 seine Ausführung verzögert und der lauffähige Task T_2 mit niedrigerer Priorität fortgesetzt wird.

6.3 Aussetzen einzelner Instanzen des Supertasks

6.3.1 Motivation

Der Supertask soll alle Aufgaben übernehmen, deren Ausführung periodisch mit den von den ADCs vorgegebenen 100 kHz erfolgen muss. Da dies mehrere Aufgaben sind, die nicht alle gleichzeitig aktiv sein müssen, kann damit auch die WCET des Supertasks unterschiedlich sein. Vor dem Hintergrund eines späteren Hinzufügens von weiteren Aufgaben, die vom Supertask mit ausgeführt werden müssen sowie der möglichen Änderung des Regelalgorithmus ist es möglich, das die WCET des Supertasks größer wird als die Zeit zwischen zwei Interrupts. Die entstehende Überlastsituation würde nicht nur zum Absinken der Regelfrequenz bzw. zum Verlieren einzelner Messwerte führen sondern auch die Ausführung aller anderen Tasks verhindern. Damit kann es beispielsweise durch das Ausfallen der Kommunikation zum PC zu einem Versagen des Gesamtsystems kommen. Es ist also notwendig, dass der Supertask nicht die volle Rechenleistung in Anspruch nimmt.

Eine Forderung bei der Konzeption war die Gewährleistung einer leichteren Modifizierbarkeit der Steuerungssoftware als auch deren bessere Erweiterbarkeit. Ein rechnerisches Ermitteln der WCET gestaltet sich schwierig. Da die dem Supertask zur Verfügung stehende Rechenzeit sehr knapp ist und somit wenig Spielraum besteht, ist

also durchaus damit zu rechnen, dass es während des Betriebs zu Zeitüberschreitungen kommen kann. Das Aussetzen einzelner Instanzen des Supertasks oder zumindest von Teilaufgaben, die innerhalb des Supertasks abgearbeitet werden ist hier eine Möglichkeit zur Verhinderung eines Versagens des Systems. Besonders in Hinblick auf die Regelung, welche die aufwendigste Aufgabe des Supertasks ist, erscheint das Aussetzen einzelner Instanzen eine vertretbare Lösung. In dem folgenden Kapitel 6.4.4 wird hierauf und auf die Auswirkungen auf das Systemverhalten näher eingegangen.

6.3.2 Ansätze

Mit dem Aussetzen einzelner Instanzen des Supertasks soll erreicht werden, dass die Zeit, die von diesem Task beansprucht wird eine bestimmte Grenze nicht überschreitet. Dies hat zum Ziel, dass im Hintergrund laufende, zeitlich weniger kritische Tasks, die aber für das Funktionieren des Gesamtsystems kritisch sein können, nicht komplett verdrängt werden. Von Bedeutung ist hier die Kommunikation. Falls diese vollständig verdrängt wird, ist das Teilsystem aus Sicht des Steuerrechners nicht mehr betriebsbereit. In Verbindung mit einer entsprechenden Statistik über das erfolgte Aussetzen von Instanzen ergibt sich automatisch die Möglichkeit, während der Entwicklung der Anwendung Informationen über Teile ihres Laufzeitverhaltens zu sammeln. Dies kann geschehen, ohne dabei in Überlastsituationen mit Beschädigungen der zu steuernden Hardware rechnen zu müssen.

Für das Aussetzen von Instanzen sind mehrere Ansätze denkbar. Der am besten geeignete hängt v. a. von den Anforderungen der Anwendung ab. Abbildung 6.2 zeigt zwei mögliche Varianten für einen Supertask, der genau eine Aufgabe ausführt. In jedem Fall wird dem Supertask ausgehend von seinem Aktivierungszeitpunkt eine Deadline gesetzt. Das Überschreiten einer (virtuellen) Deadline des Supertasks wird registriert. Bei jeder Aktivierung wird ausgehend von eventuell vorangegangenen Zeitüberschreitungen entschieden, ob der Supertask mit dem Ausführen seiner Aufgabe beginnen soll oder ob diese ausgesetzt werden muss.

Dies ist in der Abbildung in Variante a) dargestellt. In Variante b) wird die Ausführung der Aufgabe nicht ausgesetzt, sondern von einer alternativen Implementierung übernommen, die einen minimalen Funktionsumfang realisiert, dafür aber in sehr kurzer Zeit ausgeführt werden kann. Zum Zweck des Aussetzens des Regelungsalgorithmus können beide Varianten vorteilhaft sein. Es ist deshalb angestrebt, dass die Systemsoftware der Anwendung beide Möglichkeiten zur Verfügung stellt.

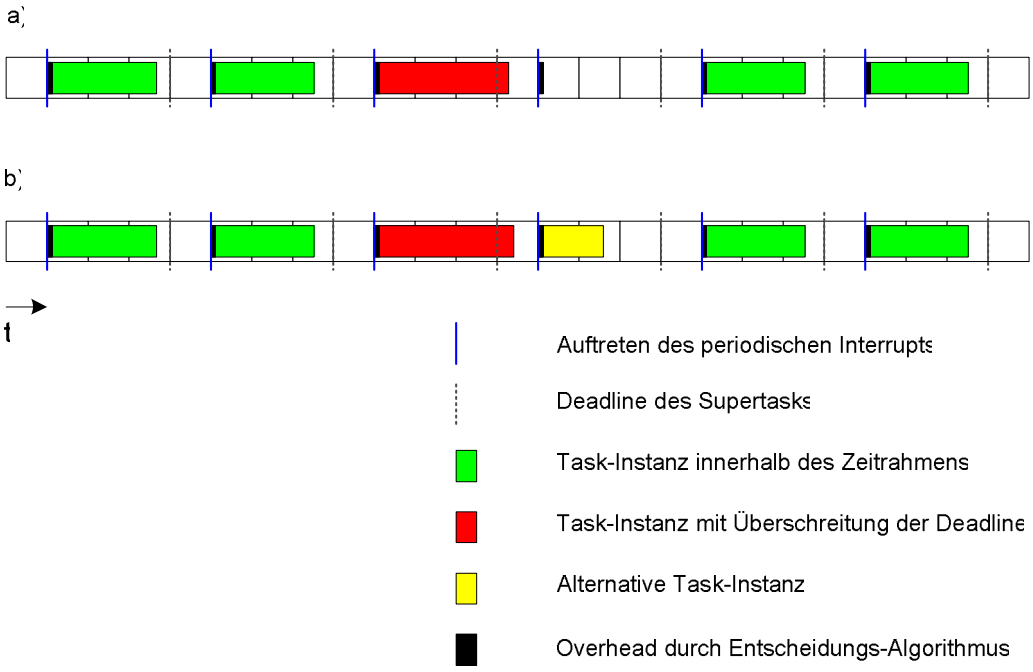


Abb. 6.2: Aussetzen des Supertasks

6.3.3 Vergleich mit dem Ansatz des TaskPair-Schedulings

Der eben erarbeitete Ansatz des Aussetzens von Task-Instanzen, welcher weiter unten noch näher untersucht und präzisiert wird, ist teilweise vergleichbar mit dem in Kapitel 2.2.4.2 beschriebenen TaskPair-Scheduling. In beiden Fällen soll erreicht werden, dass durch das mögliche Überschreiten einer Deadline eines Tasks die folgenden Tasks ebenfalls ihre Deadlines überschreiten könnten. In beiden Fällen wird dies erkauft durch das teilweise Aufgeben von Funktionalität.

Beim TaskPair-Scheduling steht dabei das Einhalten einer einzelnen Deadline im Vordergrund. In Fällen in denen dies scheinbar nicht erreicht werden kann, wird der aktuelle Task abgebrochen und ein Except-Task ausgeführt, der eine minimale Funktionalität garantiert. Zeitüberschreitungen der folgenden Tasks werden also verhindert, indem eine Zeitüberschreitung des aktuellen Tasks verhindert wird. Bei der hier erarbeiteten Methode mit Aussetzen von Task-Instanzen steht das Verhindern der Überschreitung einer maximalen Prozessorauslastung innerhalb eines größeren Zeitrahmens im Vordergrund. Es ist einzelnen Instanzen dabei erlaubt ihre virtuelle Deadline zu verletzen. Sie können im Extremfall sogar länger ausgeführt werden als es von ihrer Periodendauer her eigentlich möglich ist. Durch Auslassen von nachfolgenden Instanzen wird vielmehr die Einhaltung maximalen Prozessorauslastung in einem größeren Zeitrahmen garantiert. Die Größe des Zeitrahmens, in dem eine bestimmte

Maximalauslastung durch den Supertask garantiert werden kann, hängt von der Frequenz der Aktivierung des Supertasks ab. Es ist notwendig, dass ausreichend viele Task-Instanzen innerhalb des größeren Zeitrahmens liegen um die angestrebte Auslastung möglichst genau zu erreichen. Dies ist beim TaskPair-Scheduling nicht notwendig.

6.3.4 Auswirkungen auf das Systemverhalten

Einzelne Instanzen eines Tasks können nur dann ausgesetzt werden, wenn sie zumindest teilweise redundant sind, also ihre Aufgaben auch von einer später folgenden Instanz vollständig oder zumindest in ausreichender Art und Weise erledigt werden können. In dem Modell mit alternativem Teil kann außerdem die Aufgabe zumindest noch in minimaler Form von der aktuellen Instanz erledigt werden. Vorraussetzung ist natürlich, dass eine Erledigung der konkreten Aufgabe in minimaler Form möglich und sinnvoll ist.

Bei dem Problem der Abstandsregelung erscheint ein Aussetzen von Instanzen sinnvoll anwendbar. Durch das Fehlen einer Reaktion auf eine auftretende Regelabweichung würde sich diese bis zum Auftreten der nächsten Instanz wahrscheinlich vergrößern. Eine vergrößerte Regelabweichung kann in der nächsten Instanz des Regelungstasks mit einer stärkeren Anpassung der Ausgangsgröße korrigiert werden, solange sie noch nicht einen Grenzwert, bei dem eine Beschädigung des Systems auftritt, überschritten hat.

Bei anderen Aufgaben, wie der Messwerterfassung bei der Force Calibration, dem Abtasten einer Zeile oder der Oszilloskop-Funktion ist das Aussetzen einzelner Instanzen schwieriger. Hier würden als Folge Messwerte fehlen. Dies könnte unter Umständen noch akzeptabel sein, falls vereinzelt sehr wenige Messwerte fehlen würden. Ein entsprechendes Verhalten müsste dann aber garantiert werden können. Es ist in dem vorliegenden Fall einfacher, kein Aussetzten von diesen beiden Teilaufgaben zuzulassen. Diese Lösung ist akzeptabel, da das Erfassen von Messwerten in kurzer, konstanter Zeit erfolgen kann.

Der Rahmen in dem ein Aussetzen des Regelungstasks ohne katastrophale Folgen möglich ist ergibt sich aus den Eigenschaften der zu steuernden mechanischen Komponenten. Diese Eigenschaften werden im Folgenden näher untersucht.

Der Betrag der im Abschnitt 5.3.4 erläuterten Periodenlänge T_g und damit die Frequenz der Regelung f_R richtet sich nach den zu erwartenden maximalen Frequenzen der Regel- bzw. Rückführgröße sowie den maximal sinnvollen Frequenzen der Ausgangsgröße.

Weiterhin ist bei der Ausgangsgröße eine mindestens einzuhaltende Frequenz zu berücksichtigen. In den nächsten beiden Abschnitten sollen die angesprochenen Frequenzen anhand von vorliegenden technischen Daten der Mikroskophardware näherungsweise bestimmt werden. Anschließend werden hieraus sinnvolle minimale sowie maximal notwendige Periodenlängen für die Kontrollzyklen abgeleitet.

6.3.4.1 Arbeitsfrequenzen der Z-Aktoren

Was die zu erwartenden maximalen Frequenzen der Regelgröße angeht, so ist beim Z-System lediglich der Bereich unterhalb der Eigenfrequenz des Cantilevers zur Messung der Topografie sinnvoll verwendbar. Typische Eigenfrequenzen der verwendbaren Cantilever liegen im Bereich 10 bis 200 kHz. Das Messverfahren für die Cantilever-Schwingungen unterliegt ebenfalls Beschränkungen. Hier wäre einmal die Reaktionsgeschwindigkeit der Photodiode zu nennen. Das verwendete Modell kann laut Hersteller Signale von 20 MHz (sog. Cut-off-Frequency[12] der Photodiode) verarbeiten[13]. Ein weiterer limitierender Faktor sind die nachgeschalteten, analogen Komponenten der Messelektronik, wie z. B. der Signalverstärker. Deren Reaktionsgeschwindigkeit ist für das konkrete Gerät unbekannt, sollte jedoch deutlich niedriger liegen und stellt somit einen Tiefpassfilter für das über den DAC ausgegebene Signal dar.

Die maximale Frequenz, die für die Ausgangsgröße des Reglers sinnvoll ist, hängt von den Eigenschaften der elektronischen Komponenten ab, die sich zwischen DAC und Piezoaktoren befinden. Weiterhin hängt sie auch von den Resonanzfrequenzen der Aktoren ab. Die Z-Piezos sind dabei die Bauteile mit den kürzesten Reaktionszeiten. Die Resonanzfrequenz von typischerweise verwendeten, unbelasteten Piezoelementen liegt in einer Größenordnung von ca. 20 kHz[14]. Ein Betrieb des Aktors oberhalb der Resonanzfrequenz führt nicht mehr zur Ausführung der gewünschten Bewegung und kann zum Ausfall des Bauteils führen. Dieser Wert dieser Frequenz verringert sich weiter durch die Belastung des Aktors mit der zu bewegenden Masse, also dem Probenteller mit Probe. Die neue Resonanzfrequenz (f_0') mit zusätzlich angekoppelter Masse (M) verringert sich entsprechend Gleichung (1) [PI 2003].

$$f_0' = f_0 \cdot \sqrt{\frac{m_{eff}}{M + m_{eff}}} \qquad (1)$$

[12] ist bei einer sinusförmigen Anregung der Photodiode definiert als die Frequenz, bei der eine Dämpfung des Ausgangssignals von 3 dB gegenüber dem Ausgangssignal bei einer Frequenz von 100 kHz auftritt [Hamamatsu 2003]

[13] Angaben des Herstellers Hamamatsu für Silizium Photodiode S4349

[14] Angaben des Herstellers Physik Instrumente GmbH & Co. für die Bauteile P-820 bzw. P-802

Bei einer angenommenen effektiven Masse des Piezoaktors (m_{eff}) von 4 g und einer zusätzlichen Masse von 10 g ergibt sich eine maximale Arbeitsfrequenz von etwa 10,7 kHz. Damit folgt nach dem Abtasttheorem von Shannon eine Frequenz der zeitdiskreten Regelung von 21,4 kHz [Shannon 1949]. Das Abtasttheorem besagt im Wesentlichen, dass eine Funktion mit einer bestimmten Frequenz aus einer diskreten Funktion mit doppelter Frequenz rekonstruiert werden kann. Eine höhere Frequenz als 21,4 kHz ist dennoch sinnvoll, da sich dann die Qualität des diskreten Signals teilweise deutlich erhöhen kann.

Die minimale für den Regelungstask notwendige Frequenz richtet sich nach der Regelabweichung, die maximal auftreten darf. Versuche haben ergeben, dass der aktive Bereich der zur Detektierung des Ausschlags des Cantilevers eingesetzten Photodiode den limitierenden Faktor darstellt. Abbildung 6.3 zeigt die Größenverhältnisse des Aufbaus.

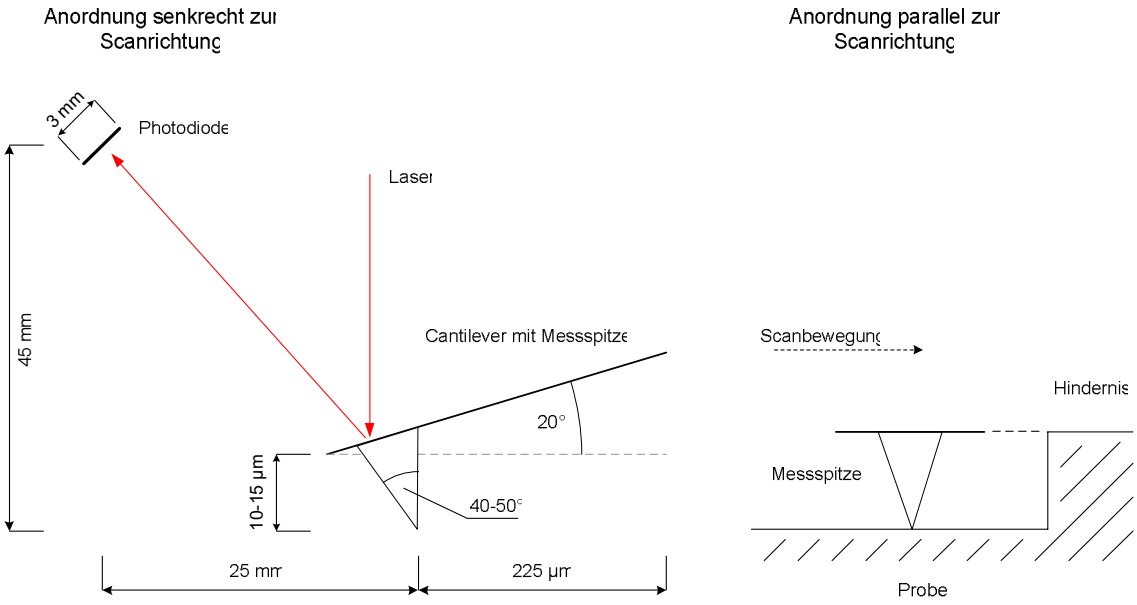


Abb. 6.3: Bestimmung des maximalen Ausschlags des Cantilevers

Sobald sich der Cantilever weit genug verbiegt, verlässt der reflektierte Laserstrahl den aktiven Bereich der Photodiode. Dies führt zum Versagen des Messverfahrens. Bei weiterer zunehmender Durchbiegung bricht der Cantilever schließlich ab. Es ist also zu vermeiden, dass der Laser mehr als a_{max} = 1,5 mm vom Zentrum der Photodiode abgelenkt wird. Bei einer Entfernung zwischen Messspitze und Photodiode von 25 mm in x- und 45 mm y-Richtung ergibt sich ein Gesamtabstand l von ca. 51,5 mm. Da der

Laserstrahl senkrecht auf den Mittelpunkt der Photodiode fällt, ergibt sich der maximale Auslenkwinkel des Cantilevers α von 1,67° durch Gleichung (2).

$$\tan\alpha = \frac{a_{\max}}{l} \qquad (2)$$

Bei einer Länge des Cantilevers von ca. 225 µm[15] ergibt sich hieraus analog das maximal mögliche Anheben der Messspitze von etwa 6,6 µm. Dies ist etwa die Hälfte der Höhe der Messspitze, welche 10-15 µm beträgt. Das Hindernis welches eine maximale Änderung des Ausschlags des Cantilevers pro Zeiteinheit hervorruft ist wie in Abbildung 4.4 dargestellt eine senkrechte Wand. Für diesen Fall lässt sich mit Hilfe des halben Öffnungswinkel der pyramidenförmigen Messspitze von durchschnittlich 22.5° wiederum der zurückgelegte Weg s_{max} in Scanrichtung ermitteln. Dieser beträgt annähernd 2,7 µm. Bei einer maximal realisierbaren Scangeschwindigkeit von 200 µm/s bedeutet dies, dass nach Aussetzten der Regelung für einen Zeitraum von 13,5 ms eine Beschädigung der Hardware auftreten kann.

Zusammenfassend kann man sagen, dass bei einer verwendeten Abtastrate von 100 kHz bereits beim Auslassen jeder zweiten Instanz ein spürbares Nachlassen der Bildqualität zu erwarten ist. Ausserdem ist bei einem Aussetzen von ca. 1350 Instanzen des Regelungstasks in Folge ist mit Beschädigungen der Hardware zu rechnen.

6.3.4.2 Aktoren im XY-System

Die Methode des Aussetzens von Instanzen eines Regelungstasks kann auch bei dem XY-System eine Rolle spielen. Obwohl im Moment hier beim Ausführen der Scanbewegung eine ungeregelte Positionierung realisiert ist, kann im Rahmen der späteren Erweiterung mittels Sensoren zur Positionsmessung hier eine ähnliche Regelung zum Einsatz kommen wie im Z-System.

Der Wert für die Resonanzfrequenz des vorhandenen XY-Positioniersystems dürfte allerdings deutlich niedriger liegen. Bei ähnlichen, fertig verfügbaren Baugruppen liegt die Resonanzfrequenz (unbelastet) in der Größenordung von ca. 500 Hz[16]. Auch ist hier die Gefahr einer Beschädigung der Hardware nicht gegeben. Ebenso tritt eine Beeinträchtigung der Scanqualität aufgrund der langsamen Arbeitsweise der Aktoren deutlich später ein als beim Z-System.

[15] Angaben des Herstellers NanoWorld AG für Sensortyp NCLR
[16] Angaben des Herstellers Physik Instrumente GmbH & Co. für Bauteil P-734

6.3.4.3 Anpassung des Regelungsalgorithmus

Es wurde festgestellt, dass das Aussetzten einzelner Instanzen des Regelungstasks möglich ist, da sich schlimmstenfalls die Regelabweichung erhöht und die nächste Instanz eine entsprechend stärkere Gegenreaktion auslösen muss. Dabei verschlechtert sich jedoch die Qualität der Regelung, da größere Regelabweichungen auftreten. Damit entfernt sich die Implementierung von dem angestrebten quasikontinuierlichen Regelverhalten[17].

Ein weiteres Problem ist, dass manche Regelalgorithmen u. U. von einer festen Zeitspanne zwischen zwei Instanzen ausgehen. Der im Moment implementierte PID-Regler gehört dazu.

Ein PID-Regler nutzt zur Berechnung der Ausgangsgröße drei Anteile, den P-, I-, und D-Anteil. Er arbeitet nach der in Gleichung (3) dargestellten Formel, wobei k_p, T_I und T_D vom Benutzer vorgegebene Parameter sind. Es ist zu erkennen, dass er dabei nicht nur auf den Betrag der Regelabweichung, sondern auch auf den Betrag der Summe der vorangegangenen Abweichungen sowie auf die Änderung der Regelabweichung reagiert.

$$u(t) = k_p \cdot \left(e(t) + \frac{1}{T_I} \cdot \int e(t)dt + T_D \cdot \frac{de(t)}{dt} \right) \qquad (3)$$

u(t)	*Ausgangsgröße*
k_p	*Verstärkungsfaktor*
e(t)	*Regelabweichung*
T_I	*Zeitkonstante I-Anteil*
T_D	*Zeitkonstante D-Anteil*

In der zeitdiskreten Implementierung [Feindt 1994, S. 105], wie sie in Gleichung (4) dargestellt ist, ergibt sich durch die integrierenden und differenzierenden Anteile eine Abhängigkeit von der Zeit T_S. Die Zeitspanne zwischen zwei Instanzen ist dabei in den neuen Parametern q_i enthalten.

$$u_k = u_{k-1} + q_0 \cdot e_k + q_1 \cdot e_{k-1} + q_2 \cdot e_{k-2} \qquad (4)$$

$$q_0 = k_p \cdot \left(1 + \frac{T_S}{T_I} + \frac{T_D}{T_S} \right)$$

$$q_1 = k_p \cdot \left(-1 - \frac{2 \cdot T_D}{T_S} \right)$$

[17] siehe Abschnitt 5.3.4

$$q_2 = k_p \cdot \frac{T_D}{T_S}$$

u_i *Ausgangsgröße zum Zeitpunkt i*

e_i *Regelabweichung zum Zeitpunkt i*

T_S *Zeit zwischen aufeinander folgenden Instanzen*

Für eine optimale Implementierung des PID-Reglers unter Berücksichtigung nicht konstanter Zeitintervalle zwischen Instanzen des Regelungstasks ist eine dynamische Anpassung der genannten Parameter erforderlich. Eine mögliche Lösung ist, die Regelparameter automatisch neu zu berechnen, sobald sich die Zeitspanne zwischen der aktuellen und der vorhergehenden Instanz geändert hat. Alternativ kann auch eine Lookup-Tabelle mit vorberechneten Werten verwendet werden. In beiden Fällen ist es notwendig, einer Taskinstanz die Anzahl der zwischenzeitlich ausgesetzten Instanzen als Parameter zugänglich zu machen.

6.4 Realisierung des Aussetzens von Instanzen

6.4.1 Anforderungen an den Algorithmus

In Abschnitt 6.3 wurde bereits der hinter dem Verfahren mit Aussetzen von Task-Instanzen steckende Ansatz sowie die notwendigen Vorraussetzungen zum Anwenden des Verfahrens erläutert. Nun soll ein Algorithmus vorgestellt werden, mit welchem das Verfahren implementiert werden kann.

Der Algorithmus sollte vor jeder Aktivierung des Supertasks entscheiden, ob die aktuelle Instanz tatsächlich ausgeführt werden soll, oder ob sie zur Einhaltung einer vorher festgelegten maximalen Auslastung des Prozessors durch diesen Task nicht aktiviert werden darf. Allgemeiner ausgedrückt soll er entscheiden, ob die Auslastung zu hoch werden kann, und wenn ja, welche Instanzen genau ihre Ausführungszeit reduzieren müssen.

Eine weitere, grundlegende Anforderung an den Algorithmus ergibt sich aus der Tatsache, dass auf dem vorliegenden System nur sehr wenig Rechenzeit für die Abarbeitung einer Instanz des Supertasks zur Verfügung steht. Es handelt es sich dabei um etwa 400 Taktzyklen. Diese Zeit sollte vor allem der Bearbeitung der eigentlichen Aufgabe zur Verfügung stehen. Der zusätzliche Aufwand, der zum Treffen der Entscheidung, ob die aktuelle Instanz ihre Ausführungszeit verringern muss oder nicht notwendig ist, sollte also so gering wie möglich sein. Gleichzeitig soll die zur

Verfügung stehende Rechenzeit so gut wie möglich genutzt werden, um möglichst viele Instanzen auszuführen. Dabei ist es besonders für den Regelungstask wünschenswert, wenn ausgeführte und nicht ausgeführte Instanzen möglichst gleichmäßig über die Zeit verteilt werden.

6.4.2 Der Algorithmus

Der Algorithmus soll wie bereits in Abschnitt 6.4 angesprochen gewährleisten, dass in einem bestimmten Zeitabschnitt die für die Ausführung des Supertasks bereitgestellte Prozessorzeit einen vorgegebenen Wert nicht überschreitet. Als Länge des Zeitabschnitts wurde die Periodenlänge des Timer-Interrupts gewählt. Da innerhalb dieses Zeitabschnitts der Supertask sehr häufig[18] und für kurze Zeit aktiviert wird, kann eine Beschränkung der durch ihn beanspruchten Prozessorzeit durch das Aussetzen einzelner Instanzen ausreichend genau gesteuert werden. Da die konkreten Ausführungszeiten der Task-Instanzen nicht vorher bekannt sind, ist es nicht möglich, ein Ausführungsschema vor Beginn eines Intervalls festzulegen. Auch Statistiken von vorangegangenen Intervallen können nicht herangezogen werden, da sie das zu erwartende Verhalten nur näherungsweise beschreiben. Das Problem wird stattdessen durch eine Art Akzeptanztest unmittelbar vor der Ausführung jeder einzelnen Instanz realisiert.

Bei der Konzeption des folgenden Algorithmus wurde in erster Linie auf die Erfüllung der Forderung nach einem möglichst geringen Overhead geachtet. Ein erster Lösungsansatz, der diese Forderung erfüllt besteht in der Anwendung eines Greedy-Algorithmus. Dieser würde die Ausführung einer Task-Instanz solange immer genehmigen bis deren akkumulierte Ausführungszeit die für das Intervall maximal vorgegebene Zeit erreicht hat. Alle bis zum Ende des Intervalls folgenden Instanzen würden dann ausgesetzt werden. Dieser Ansatz hat den entscheidenden Nachteil, dass am Ende des Intervalls eine Häufung von nicht ausgeführten Instanzen auftritt. Der in Abbildung 6.4 im Pseudocode angegebene Algorithmus löst das Problem, indem er versucht, die ausgelassenen Instanzen möglichst gleichmäßig über das Intervall zu verteilen.

Initial stehen Informationen über den Prozentsatz der maximal zu benutzenden Prozessorzeit innerhalb eines jeden Intervalls, die Länge des Intervalls sowie die Anzahl der zu erwartenden Instanzen zur Verfügung. Hieraus ergibt sich ein Grenzwert

[18] bei der aktuellen Konfiguration ca. 100 mal

(*limitTime*) für die Ausführungszeit einer einzelnen Instanz. Weiterhin wird ein maximal erlaubter Übertrag festgelegt, auf den später noch eingegangen wird.

```
;globale Variablen
global var error, lastEndTime, lastBeginTime
global var limitTime = [maximal vorgegebene Ausführungzeit]
global var maxCarryOver = [maximal erlaubter Übertrag]

;testet, ob aktuelle Instanz ausgeführt werden soll
function Shall_I_Execute

      ;Fehlerwert um Ausführungszeit der vorangegangenen
      ;Instanz erhöhen
      error = error + (lastEndTime - lastBeginTime)

      ;maximal erlaubte Ausführungzeit vom Fehlerwert
      ;abziehen
      error = error - limitTime

      ;wenn Ausführungszeit nicht überschritten wurde
      ; -> ausführen
      if (error <= 0) then
            return true
      elseif
            return false
      endelse

endfunction

;wird mit Timer-Interrupt aufgerufen
function On_Time_Tick

      ;Limitierung des übertragenen Zeitguthabens
      if (error < (0 - maxCarryOver)) then
            error = 0 - maxCarryOver
      endif

endfunction
```

Abb. 6.4: Pseudocode des Algorithmus zum Aussetzen von Task-Instanzen

Bei der Aktivierung einer Task-Instanz wird unmittelbar nach der Unterbrechung des aktiven Tasks der Startzeitpunkt der Unterbrechung gemessen. Kurz vor dem Fortsetzen des unterbrochenen Tasks wird dementsprechend der Endzeitpunkt ermittelt. Der Akzeptanztest gestaltet sich dann wie in der Funktion *Shall_I_Execute*, welche in Abbildung 6.4 dargestellt ist. Es wird ein Fehlerwert (*error*) geführt, in den die Differenz aus dem gegebenen Grenzwert für die Ausführungszeit und der tatsächlichen Ausführungszeit der vorangegangenen Instanz eingeht. Sobald dieser Wert positiv wird,

könnte für das aktuelle Intervall die Ausführungszeit überschritten werden. Die nächste Instanz wird dann ausgesetzt, woraufhin sich der Wert wieder verringert. Sobald der Fehlerwert auf oder unter Null sinkt, kann wieder eine Instanz ausgeführt werden. Der Fehlerwert stellt effektiv die Summe der Laufzeiten aller bisher im Intervall aufgetretenen Task-Instanzen abzüglich der bis dahin maximal zu verbrauchenden Prozessorzeit dar.

Es ist möglich, dass der Fehlerwert negativ wird, wenn die Ausführungszeiten des Supertasks kleiner sind als die maximal zu beanspruchende Prozessorzeit. Dabei wird dann ein Zeitguthaben angesammelt, welches später im Intervall aufgebraucht werden kann.

Zwischen Intervallen stellt sich die Frage des Übertrags von Zeitguthaben ins nächste Intervall. Dies ist zu vermeiden, da es so zu Überscheitungen der einzuhaltenden maximalen CPU-Auslastung im nächsten Intervall kommen kann. Es wird jedoch trotzdem ermöglicht, ein begrenztes Zeitguthaben mit in das nächste Intervall zu übernehmen, da so eine bessere Kontinuität in der Abfolge von auszusetzenden Instanzen hergestellt werden kann. Das maximal zu übertragende Zeitguthaben (`maxCarryOver`) wurde festgelegt auf die Differenz zwischen der Periodenlänge des Supertasks und der maximal pro einer solchen Periode zu beanspruchenden Ausführungszeit (`limitTime`). Eine eventuelle Zeitüberschreitung wird ebenfalls in das nächste Intervall übernommen.

Es wird deutlich, dass der Algorithmus die Einhaltung der angestrebten maximalen Auslastung nicht vollständig garantieren kann. Dies liegt im prinzipiellen Ansatz, Task-Instanzen auszuführen, ohne die Möglichkeit zu haben, diese abzubrechen, wie dies beim TaskPair-Scheduling der Fall ist. Es ist allerdings möglich, die maximale Überschreitung der beanspruchten Prozessorzeit pro Intervall zu definieren, in dem man hierfür eine maximale Ausführungszeit der Instanzen vorgibt. Diese muss nicht der tatsächlichen WCET entsprechen und kann hierfür vergleichsweise hoch, z. B. im Bereich des Doppelten der Periodenlänge des Supertasks, angesetzt werden.

Da die maximale Überschreitung innerhalb der gesamten Periode nicht höher sein kann als die maximale Überschreitung einer Instanz, ergibt sich unter dieser Bedingung eine garantierte, maximal in Anspruch genommene Prozessorzeit (CPU_{max}) wie sie in Gleichung (5) dargestellt ist.

$$CPU_{max} = T_{limit}\,(n\text{-}1) + T_{carryover} + T_{max} \qquad (5)$$

T_{limit}	*Grenzwert der Ausführungszeit einer einzelnen Instanz*
$T_{carryover}$	*maximal übertragendes Zeitguthaben*
n	*Anzahl Instanzen pro Intervall*
T_{max}	*Grenzwert für die Ausführungszeit einer Instanz*

Unter den gegebenen Bedingungen, also 100 Instanzen pro Intervall von einer Millisekunde, einem Grenzwert für die Ausführungszeit einer Instanz von dem doppelten der Periodenlänge des Task und einer vorgegebenen Beschränkung der Prozessorzeitnutzung auf 50% ergibt sich eine garantierte maximale Auslastung durch diesen Task von 52%.

Verbesserungsmöglichkeiten für das Verfahren ergeben sich aus der Tatsache, dass für den Fall, dass die restlichen Tasks innerhalb eines gegebenen Intervalls ihre garantierte Prozessorzeit nicht benötigen der Supertask ausgesetzt werden würde, obwohl eigentlich genügend Prozessorzeit zur Verfügung stehen würde. Ein Lösungsansatz wäre eine dynamische Anpassung des gesetzten Grenzwerts. Es müsste hierfür untersucht werden, ob eine dynamische Anpassung möglich ist und wie ein geeignetes Verfahren hierfür aussehen würde.

6.5 Kommunikation

Die Kommunikation zwischen den beiden Teilsystemen des Steuerrechners und dem PC stellt einen weiteren wichtigen Teil der Konzeption der Steuerungssoftware dar. Eine Kommunikationslösung ist notwendig, da das System aus mehreren selbständig arbeitenden Teilsystemen besteht, die Steuerinformationen sowie Messwerte austauschen müssen. Die Art und Weise mit der die Kommunikation stattfindet, beeinflusst außerdem die Überlegungen zur Realisierung der Funktionalität der Anwendung mit Hilfe einzelner Prozesse. Die bestehende Lösung, deren Nachteile sowie Anforderungen an eine Verbesserung wurden bereits in vorangegangenen Kapiteln beschrieben.

6.5.1 Zu übermittelnde Daten

Bei den zu übermittelnden Daten handelt es sich wie angesprochen um 32 Bit große Pakete. Aus dem Anwendungsprotokoll ergibt sich dabei ihre entsprechende Bedeutung und damit eventuelle zeitliche Anforderungen.

Das Anwendungsprotokoll ist angelehnt an den Master-Slave Ansatz der Kommunikationsschnittstelle. Es werden dabei einseitig Befehle vom Anwendungsprogramm an die beiden Teilsystem des Steuerrechners gesendet. Diese werden, nachdem sie dort empfangen wurden, interpretiert und ausgeführt. Hierbei handelt es sich durchweg um vergleichsweise einfache Befehle, z. B. das Setzen einzelner Parameter, oder das Starten eines Vorgangs. Nach Ausführung eines Befehls wird dieser quittiert. Gegebenenfalls werden zu diesem Zeitpunkt zusätzlich noch einzelne Informationen die die Ausführung des Befehls betreffen an den PC gesendet. Bei der Übertragung von Befehlen fallen also nur geringe Datenmengen an.

Ein weiterer Punkt ist die spontane Übertragung von verschiedenartigen Nutzdaten zum PC. Hierzu zählen beispielsweise die Messwertreihen, die bei der Oszilloskop-Funktion oder dem Abtasten einer Zeile aufgenommen werden. Es handelt sich dabei um bis zu mehrere hundert Werte, die ebenfalls jeweils in ein oder zwei 32 Bit Pakete verpackt gesendet werden.

Aus der in Kapitel 5.3 erfolgten detaillierten Darstellung der Aufgaben der Steuerungssoftware ist ersichtlich, dass die Übertragung der anfallenden Daten keinen harten Zeitbedingungen unterliegen. Fast alle Befehle sollten zwar möglichst schnell ausgeführt und bestätigt werden, eine Ausführung zu einem bestimmten Zeitpunkt ist jedoch nicht notwendig. Eine Ausnahme ist das Signalisieren des Abtastens einer Zeile und des gleichzeitigen Ausführens der Scanbewegung. Hier führt ein versetztes Starten beider Vorgänge grundsätzlich zu verfälschten Messwerten. Auch das spontane Übertragen von gesammelten Messwerten unterliegt keiner zeitlichen Beschränkung, da wie in Kapitel 5.3 beschrieben die Messwerte einer Zeile während des Messvorgangs zwischengespeichert und anschließend übertragen werden können.

Bei einer Erweiterung hinsichtlich der Möglichkeit zur direkten Mikromanipulation mit Hilfe eines Joysticks mit Kraftrückkopplung ergibt sich jedoch die Forderung nach einer zeitnahen Übertragung von bestimmten Messwerten zum PC sowie dem Empfang bestimmter Parametern vom PC. Eine Echtzeitfähige Lösung wäre hierfür wünschenswert. Bei einer Aktualisierungsrate kommerziell verfügbarer Kraftrückkopplungsgeräte von etwa 1 kHz ist es nicht realistisch, die sich hierbei ergebenden maximal erlaubten Übertragungszeiten mit dem vorliegenden System garantieren zu können.

6.5.2 Ansatz zur Erweiterung der bestehenden Lösung

Die für das gleichzeitige Signalisieren des Abtastens einer Zeile und des Ausführens der Scanbewegung notwendige Übertragung von Befehlen innerhalb einer vorgegebenen, sehr kleinen Zeitspanne kann von der bestehenden Lösung nicht garantiert werden. Im Folgenden wird hierfür eine Lösung vorgestellt, welche auf der Basis von zu bestimmten Zeitpunkten signalisierten Ereignissen arbeitet. Ein weiterer Ansatz beschäftigt sich damit, das Problem der zeitnahen Übertragung von Datenpaketen bei der direkten Mikromanipulation zu lösen. Bei der Lösung beider Teilprobleme steht ein möglichst geringer Aufwand bei der Anpassung der Anwendungssoftware auf dem PC im Vordergrund.

6.5.2.1 Berücksichtigung von Zeitanforderungen der Nutzdaten

Das Einhalten von Zeitbedingungen bei der Übertragung von Daten zwischen den Teilsystemen des Steuerrechners und dem PC ist für die zukünftige Integration eines Geräts zur direkten Mikromanipulation interessant. Aufgrund der Tatsache, dass die Anwendungssoftware QuickScan als Benutzerprogramm unter Windows keine Echtzeitfähigkeiten besitzt ist die Einhaltung der angesprochenen, engen zeitlichen Bedingungen bei der Übertragung von Daten schwierig. Bei einer Erweiterung der Anwendung z. B. durch spezielle Hardwaretreiber für den Parallelport, die zeitkritische Teile des Protokolls implementieren ist zu berücksichtigen, dass für den beabsichtigten Einsatzzweck die Daten auch weiterverarbeitet werden müssen. Spätestens hier wird man auf dem Windows-PC erneut an eine Grenze stoßen. Es ist allerdings möglich, für die Kraftausgabe keine direkte Kopplung mit der Mikroskophardware zu verwenden, sondern hierfür ein Modell der Probenoberfläche an der aktuellen Position zu verwenden. Dieses Modell würde dann regelmäßig durch Messwerte vom Mikroskop aktualisiert werden. Dieser Vorgang wäre deutlich weniger zeitkritisch. Auch hierfür wäre eine Übertragung von Messwerten mit niedriger Latenzzeit und hoher Übertragungsrate wünschenswert. Sie ist aber nicht unbedingt erforderlich.

Es erscheint aus diesen Gründen ein Best-Effort-Ansatz als die am besten geeignete Lösung. Es werden bei dem in Abbildung 6.5 dargestellten Kommunikationsschema auf Seiten der DSPs mehrere Sendepuffer für Daten mit unterschiedlicher Priorität bereitgehalten. Beim Senden von Daten wird diesen nun ein Prioritätswert mitgegeben. Entsprechend des Prioritätswertes werden sie daraufhin in den entsprechenden Ringpuffer geschrieben. Von dort aus gelangen sie in den eigentlichen Sendepuffer. Aus diesem können sie per DMA Transfer zur Kommunikationshardware und von dort zum PC übermittelt werden. Dabei werden erst dann Daten aus einem Puffer mit niedrigerer

Priorität gesendet, wenn die Daten aus allen Puffern mit höherer Priorität gesendet wurden. Bei dem Ansatz wird sich also bemüht, die Daten mit der höchsten Priorität so früh wie möglich zu übertragen. Es wurden drei verschiedene Prioritätsstufen eingeführt. Neben der normalen Stufe für alle bisherigen Daten existieren noch eine hohe Stufe für Daten der Mikromanipulation und eine niedrige Stufe, z. B. zur Übertragung von Laufzeitstatistiken des Supertasks.

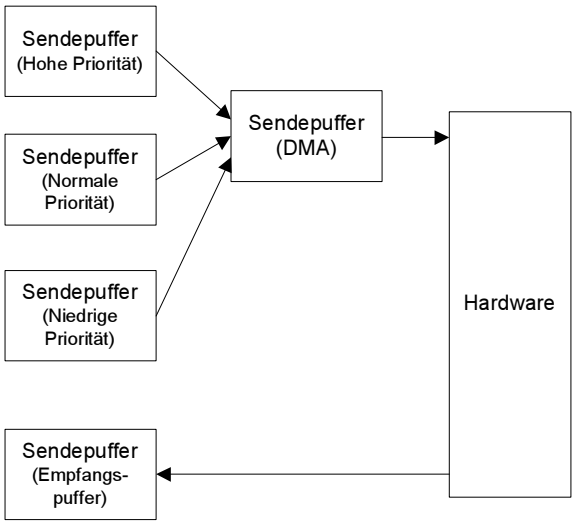


Abb. 6.5: Übertragung von Daten mit unterschiedlichen Prioritäten

Im Vordergrund steht bei diesem Ansatz die Übertragung von Daten vom DSP zum PC. Die umgekehrte Richtung bleibt unverändert. Dies hat den Grund darin, dass auf dem DSP wesentlich größere Datenmengen in Form von Messergebnissen anfallen können, die übertragen werden müssen als umgekehrt. Vom PC kommen nur einzelne Befehle in Form von einzelnen 32 Bit Wörtern. Da nach einzelnen Kommandos auch Bestätigungen abgewartet werden, ist auf der PC Seite das Füllen eines Sendepuffers mit Daten niedriger Priorität momentan nicht zu erwarten. Bei der Übertragung vom DSP zum PC ist das anders. Bei der zukünftigen Erweiterung für einen Force-Feedback Joystick fallen auch hier mehr Daten an. Es bietet sich dann an auch die Übertragungsfunktionen in der Anwendungssoftware auf dieselbe Art und Weise zu erweitern.

6.5.2.2 Synchrone Ereignisse

Eine gewisse Differenz ist für den Fall des gleichzeitigen Signalisierens eines Ereignisses auf beiden Telsystemen des Steuerrechners akzeptabel. Im vorliegenden

Anwendungsfall ist eine solche Synchronisation für das gleichzeitige Starten des Abtastens einer Zeile und des gleichzeitigen Ausführens der Scanbewegung erforderlich. Der Betrag einer akzeptablen Zeitdifferenz hierbei hängt von der gewählten Scangeschwindigkeit ab. Eine Differenz von 2-3 ms kann im Allgemeinen als akzeptabel betrachtet werden. Diese Differenz wird auch von der bestehenden Lösung in den meisten Fällen eingehalten. Durch das Verdrängen der nicht echtzeitfähigen Windows-Anwendung zwischen dem Senden der jeweiligen Startbefehle kann es jedoch zu deutlich größeren Differenzen kommen. Eine Lösung wäre die zeitnahe Übertragung der beiden Startbefehle durch geeignete Maßnahmen, z. B. den Einsatz eines speziellen Gerätetreibers auf dem Windows-PC, zu gewährleisten. Hierfür wären jedoch voraussichtlich umfangreiche Modifikationen an der Anwendungssoftware notwendig. Ein Ansatz bei dem dies nicht notwendig ist und der hier verfolgt wird, ist das Signalisieren eines gemeinsamen Ereignisses auf beiden Teilsystemen des Steuerrechners zu einem festgelegten Zeitpunkt. Hierfür ist eine synchronisierte, globale Zeit notwendig. Der Ansatz verlagert das Problem also weg von Garantie der Übertragung von zwei individuellen Startkommandos innerhalb eines bestimmten Zeitrahmens hin auf einem Mechanismus zur Zeitsynchronisation. Um die Zeit synchron zu halten ist zwar wiederum eine Kommunikationsverbindung mit möglichst deterministischen Übertragungszeiten vorteilhaft. Verfügbare Zeitsynchronisationsprotokolle wie z. B. NTP [Mills 2004] minimieren jedoch die Abweichungen, die durch schwankende Übertragungszeiten hervorgerufen werden und bieten so eine bessere Basis als die Synchronisation als eine einmalige Übertragung, bei welcher eine größere Abweichung auftreten kann.

Abbildung 6.6 zeigt den in diesem Szenario vorgesehenen Ablauf bei der Signalisierung eines gemeinsamen Ereignisses. Da dieser Mechanismus nur beim Signalisieren des Abtastens einer Zeile und des gleichzeitigen Ausführens der Scanbewegung notwendig ist, ist es hinsichtlich einer Implementierung ausreichend in jedem Teilsystem ein einzelnes Synchronisationsobjekt z. B. in Form einer speziellen globalen Semaphore vorzusehen.

Das synchrone Startkommando, welches in Abbildung 5.7 dargestellt ist würde dann nicht den Scanvorgang direkt starten, sondern den in der vorangegangenen Abbildung dargestellten Ablauf anstoßen. Es ist jetzt unerheblich, wie groß die zeitliche Differenz zwischen dem Empfang der Kommandos auf den beiden Teilsystemen ist. Beide Teilsysteme nehmen in einem nächsten Schritt untereinander, auf eine hier nicht näher speifizierte Art und Weise, eine Zeitsynchronisation vor. Nach erfolgter Synchronisation signalisieren beide Teilsysteme unabhängig voneinander nach einer definierten Zeitspanne jeweils ihre entsprechenden Semaphore.

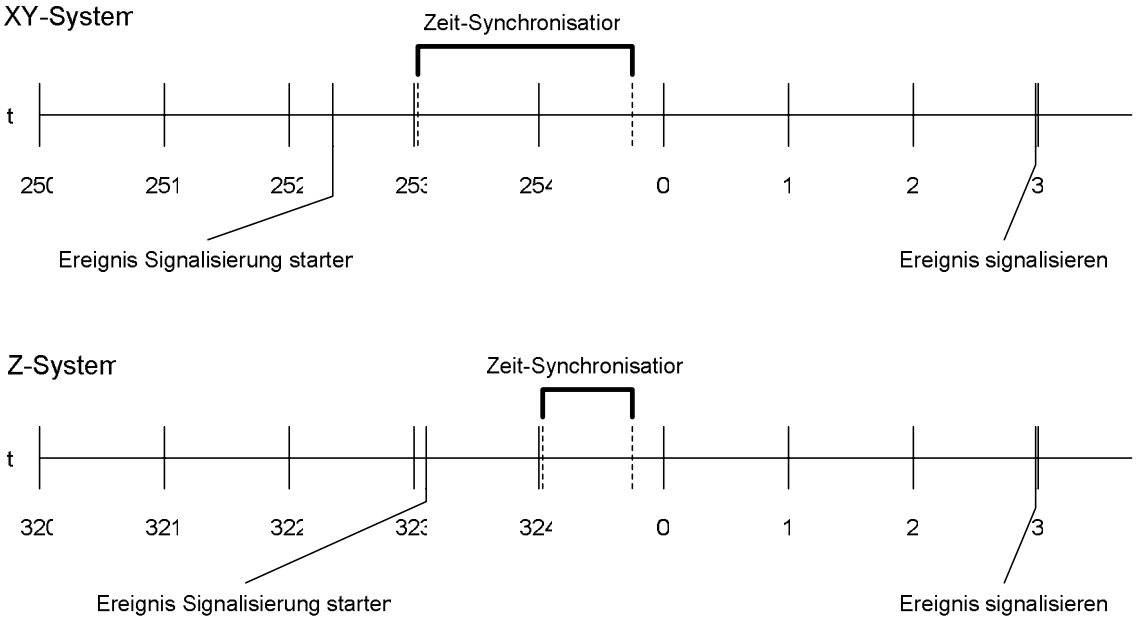


Abb. 6.6: Ablauf der Signalisierung eines verteilten Ereignisses

Hinsichtlich der Zeitsynchronisation kam für Testzwecke kein aufwendiges Protokoll zum Einsatz, welches über den EPP arbeitet. Der Einfachheit halber wurde die Synchronisation mittels einer zusätzlichen, direkten Verbindung zwischen beiden Teilsystemen gewählt. Durch diese zusätzliche Verbindung ist es möglich durch das Setzen eines vorher unbenutzten Digitalausgangs des Z-Systems einen vorher unbenutzten Interrupt im XY-System auszulösen und so eine Zeitsynchronisation vorzunehmen. Durch die Austauschbarkeit des Zeitsynchronisationsmechanismus in dem vorgestellten Schema wären allerdings auch andere Varianten denkbar.

6.5.2.3 Integration des Kommunikationskonzepts in die Software

Bei der Betrachtung der Integration des Kommunikationskonzepts ist zum einen die Einbindung in die Steuerungssoftware der DSPs, zum anderen die in das Anwendungsprogramm QuickScan zu betrachten. Aufgrund der Zielsetzung, die Anwendungssoftware möglichst ohne große Veränderungen weiter einsetzen zu können wurden nur Veränderungen durchgeführt die für die Anwendung transparent sind. Eine Erweiterung des Anwendungsprotokolls für die Übertragung von Laufzeitstatistiken kann dabei ohne großen Aufwand vorgenommen werden.

Um das in Abschnitt 6.5.2.1 vorgestellte Konzept für die Übertragung von Daten mit unterschiedlicher Priorität in die DSP-Software zu integrieren wurde im Gegensatz zur bestehenden Lösung ein schichtenbasierter Ansatz gewählt. Abbildung 6.7 zeigt die vier

verwendeten Schichten. Diese lassen sich grob mit Schichten des ISO/OSI-Referenzmodells[19] vergleichen.

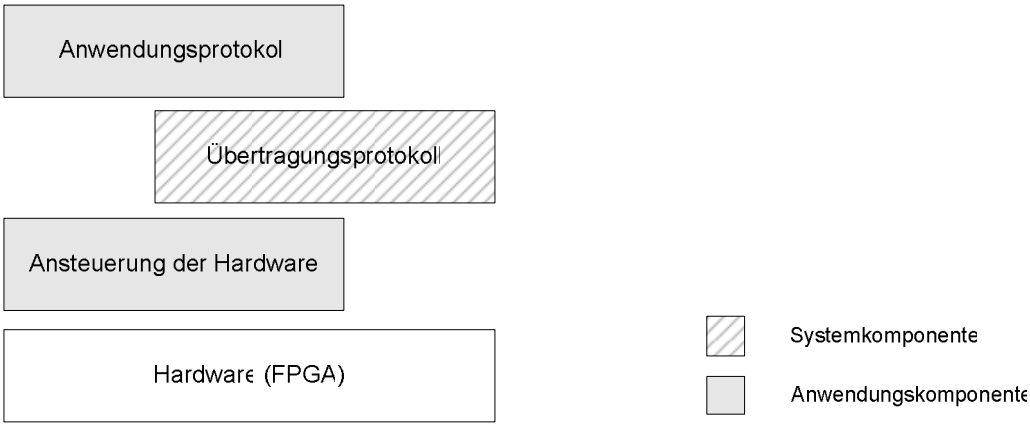


Abb. 6.7: Kommunikationskonzept als Schichtendarstellung

Die in der Abbildung dargestellte Hardware entspricht der Bitübertragungsschicht im ISO/OSI-Modell. Die Ansteuerung der Hardware stellt Funktionen zum Senden und Empfangen von einzelnen Datenpaketen (32-Bit Wörter) zur Verfügung. Darauf setzt die Schicht mit dem Übertragungsprotokoll auf, welche der Anwendung einen Kommunikationskanal anbietet und in etwa der Transportschicht entspricht. Das Anwendungsprotokoll bildet die oberste Schicht und entspricht auch in etwa den oberen Schichten des ISO/OSI-Modells.

Eine weitere Frage ist, ob und wenn ja welche Komponenten des Kommunikationskonzepts in die Systemsoftware aufgenommen werden sollen. Das Anwendungsprotokoll ist typischerweise Teil der Anwendung. Das Übertragungsprotokoll eignet sich gut für eine Implementierung als Teil der Systemsoftware. Hier können Funktionen zum Senden von Datenpaketen mit unterschiedlicher Priorität sowie zum (blockierenden) Empfang integriert werden. Ausschließlich diese werden dann von der Anwendung zur Kommunikation genutzt. Die Ansteuerung der Hardware kann sowohl der Anwendung als auch der Systemsoftware zugeordnet werden. Die Systemsoftware ist wie bereits beschrieben in recht großem Umfang an die Bedürfnisse der Anwendung anpassbar. Insofern ist es durchaus denkbar, dass Ansteuerung der Hardware als Modul mit in die Systemsoftware integriert wird. Die Ansteuerung der vorliegenden Hardware ist jedoch sehr speziell, da sie nur für diesen Anwendungsfall gebaut wurde. Es ist daher sinnvoller die

[19] siehe [ISO 1984]; eine Übersicht über die dort definierten Schichten findet sich außerdem in Anhang C

Ansteuerung der Hardware als Teil der Anwendung zu sehen. Diese Zuordnung ist ebenfalls in der vorangegangenen Abbildung dargestellt.

7 Implementierung

In diesem Abschnitt wird eine im Rahmen dieser Arbeit vorgenommene programmtechnische Umsetzung der Ansätze aus dem vorangegangenen Kapitel vorgestellt. Bei der Implementierung lag der Schwerpunkt auf der Erweiterung von MicroC/OS-II um das Supertask-Konzept. In dem Zusammenhang wurde auch eine Umsetzung des entwickelten Verfahrens zum Aussetzen von Task-Instanzen vorgenommen. Die Zielvorstellung hierbei war, die prinzipielle Eignung des Verfahrens für den Anwendungsfall praktisch zu überprüfen.

Für diesen Zweck ist es ausreichend, eine Steuerungssoftware für das Z-Teilsystem zu erarbeiten. Da das Anwendungsprotokoll der neuen Software weiterhin kompatibel zur bestehenden Lösung ist, ist ein gleichzeitiger Einsatz der neuen Software für das Z- sowie der bestehenden Software für das XY-System möglich. Aufgrund der genannten Schwerpunkte wurde auf eine Umsetzung der Kommunikationslösung verzichtet. Es soll dabei sowohl die Möglichkeit zum Übertragung von Daten mit unterschiedlichen Prioritäten als auch zum Signalisieren von verteilten Ereignissen nicht weiter betrachtet werden. Die implementierte Lösung ist so zwar nicht vollständig aber dennoch funktionsfähig, was im Rahmen dieser Arbeit für Testzwecke ausreichend erscheint.

Die Implementierung wurde in der Programmiersprache C vorgenommen. Dafür war eine Reihe von Entwicklungswerkzeugen vorhanden, die vom Hersteller der DSPs zur Verfügung gestellt wurden und die diese Sprache unterstützen. Es handelt sich dabei um eine auf dem GNU-C-Compiler basierende Lösung bestehend aus Präprozessor, Assembler, Compiler, Linker sowie einer C-Laufzeitbibliothek. Leider unterstützt die vorliegende Version nur eine bestimmte, feste Aufteilung des Arbeitsspeichers des DSPs. Dabei ist das nutzbare Code-Segment mit ca. 30 kByte sehr klein ausgefallen, so dass die durchgeführte Implementierung hier bereits an Grenzen gestoßen ist, obwohl noch genügend Speicher zu Verfügung stehen würde. Es wird vom Hersteller jedoch auch eine Vollversion angeboten die diesen Mangel nicht aufweist und bei Bedarf eingesetzt werden müsste. Weiterhin stand ein auf Windows basierender Simulator mit integriertem Debugger zur Verfügung. Die Einsatzmöglichkeiten dieses Werkzeugs sind jedoch begrenzt, da nur der Prozessorkern ohne weiterhin angeschlossene Hardware simuliert werden kann.

7.1 Anpassungen der Systemsoftware an die Hardware

Um das Betriebssystem MicroC/OS-II nutzen zu können, musste dieses an die vorliegende Hardware angepasst werden. Es ist eine offizielle Portierung des Systems

auf den ADSP-21065L in Form einer Reihe zusätzlicher bzw. modifizierte Quelltext-Dateien verfügbar. Diese ist aufgrund kleinerer Unterschiede in der Hardware auf dem eingesetzten ADSP-21061 nicht lauffähig. Das Problem konnte jedoch durch wenige Modifikationen behoben werden. Hierzu zählten die Anpassung der Initialisierung des Timers und dessen Interrupts sowie die Anpassung an die unterschiedliche Taktfrequenz des eingesetzten Prozessors.

Die Portierung unterstützt noch nicht das in der aktuellen Systemversion angebotene Speichermanagement. Dessen Verwendung ist jedoch nicht geplant. Somit stellt dies keinen Nachteil dar.

7.2 Umsetzung der Steuerungssoftware mit µC/OS-II Tasks

Für die Umsetzung der Steuerungssoftware mit Hilfe mehrerer Tasks wurden unabhängig oder sogar nebenläufig ausführbare Teile der bestehenden Lösung mit Hilfe einzelner Tasks implementiert. Die Funktionen der bestehenden Software, welche zur Funktionalität eines jeweiligen Tasks beitragen wurden dabei in einem Modul zusammengefasst.

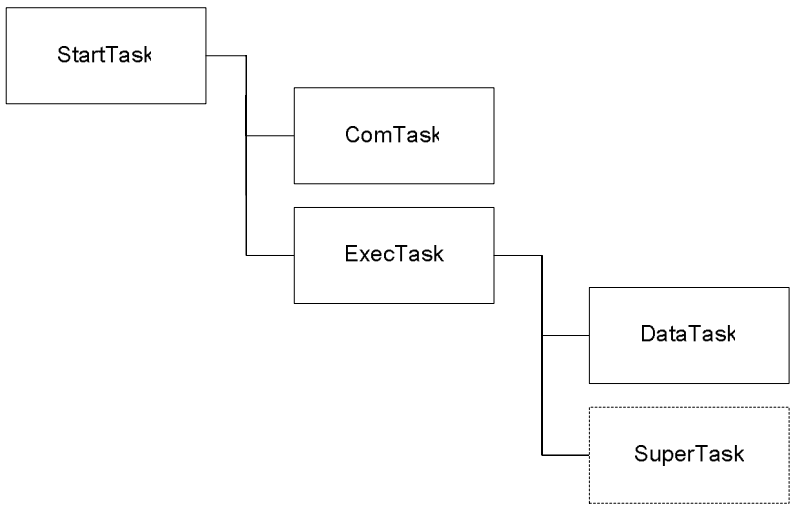


Abb. 7.1: Erzeugungsreihenfolge der Anwendungstasks

Nach dem Systemstart wird der erste Task gestartet. Dabei handelt es sich um den `StartTask`, der zuerst zentrale Teile der Hardware initialisiert und anschließend weitere Tasks startet, welche dann die eigentliche Anwendung implementieren. Wie in Abbildung 7.1 dargestellt, handelt es sich dabei um den `Com-` und den `ExecTask`. Nachdem beide gestartet wurden, beendet sich der `StartTask`. Der `ComTask`

übernimmt die Kommunikation mit dem PC. Seine Aufgabe ist es, nachdem die Kommunikationshardware initialisiert wurde, in regelmäßigen Intervallen von 1 ms das Vorliegen neuer Daten zu prüfen und bei Bedarf deren Empfang über das Signalisieren eines globalen Flags anzuzeigen. Ausserdem überprüft er ob Daten zu senden sind und bei Bereitschaft des FPGAs Daten zu senden, startet er entsprechend den DMA-Transfer. Der Task verwaltet für das Senden von Daten einen weiteren Puffer, in den zu sendende Daten zwischengespeichert werden können.

Der `ExecTask` übernimmt die Interpretation und Ausführung sämtlicher Befehle die vom PC kommen. Er wartet dafür auf die Signalisierung des Empfangs von Daten seitens des `ComTasks`, interpretiert anschließend den Befehl und führt die gewünschte Operation aus. Der `ExecTask` registriert bei seinem Start den `SuperTask` und erzeugt den `DataTask`, welcher regelmäßig die Notwendigkeit zum Übertragen der von der Oszilloskop-Funktion oder dem Abtasten einer Zeile erzeugten Daten prüft und die Datenübertragung ggf. vervollständigt. Eine periodische Aktivierung alle 20 ms ist für diesen Vorgang ausreichend.

Der `SuperTask` wird periodisch vom System aufgerufen und übernimmt mehrere Teilaufgaben, die Zeitbedingungen unterliegen. Dabei handelt es sich je nach Betriebszustand um das Auslesen der ADCs, die Durchführung des Tip Approach sowie das Aufnehmen von Messwerten bei der Oszilloskop-Funktion und beim Abtasten einer Zeile. Durch das Vorhandensein eines ausreichend großen Sendepuffers, der vom `ComTask` verwaltet wird ist es möglich die aufgenommenen Werte direkt vom `SuperTask` aus zu senden. Für den existiert eine alternative Version, die aufgerufen wird, wenn Instanz ausgesetzt werden soll. Diese implementiert als Basisfunktionalität das Auslesen der ADCs und das Aufnehmen der gewonnenen Messwerte.

Der SuperTask hat wie bereits beschrieben die höchste Priorität. Die weiteren, periodisch auftretenden Tasks für die Kommunikation und für das Vervollständigen von Datenübertragungen bekommen entsprechend ihrer Frequenz Prioritäten zugewiesen. Der Task mit höherer Frequenz, also der `ComTask` bekommt dabei die höhere Priorität. Die Priorität des sporadisch auftretenden `ExecTasks` ist die niedrigste im System.

7.3 Erweiterung der Systemsoftware für den Supertask

Im Folgenden soll beschrieben werden, in welcher Form der Supertask und das damit verbundene Verfahren zum Aussetzen von Instanzen implementiert und in die Systemsoftware eingefügt wurde.

Der Ablauf der Verwendung eines Supertasks ist in Abbildung 7.2 dargestellt. Der Supertask wird nicht als normaler Task erstellt, sondern es wird eine Funktion implementiert, die die Funktionalität des Supertasks realisiert (Supertask Handler) und welche vom System bei Bedarf aufgerufen wird.

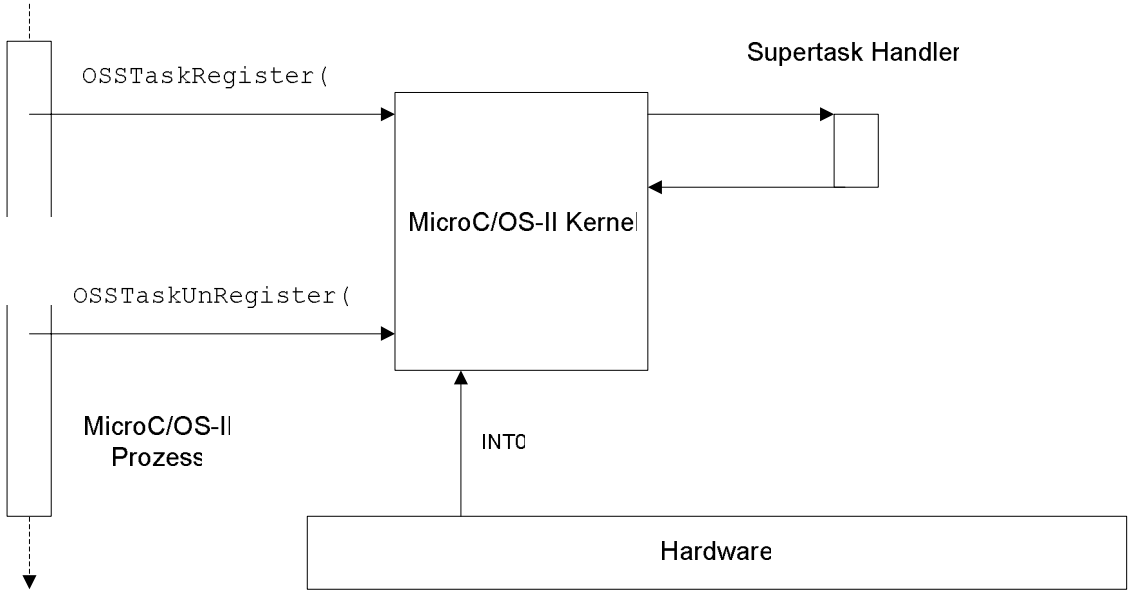


Abb. 7.2: Integration des Supertasks in µC/OS-II

Die Registrierung des Supertasks erfolgt über die neue Systemfunktion `OSSTaskRegister()`. Hier wird die Adresse des Supertask-Handlers angegeben sowie die maximal zu benutzende Prozessor-Zeit. Außerdem ist es möglich, eine alternative Funktion anzugeben, die aufgerufen wird, falls eine Instanz ausgesetzt werden soll. In der vorliegenden Implementierung wird der Supertask durch Interrupt 0 ausgelöst. Durch das Definieren einer entsprechender Konstanten ist es nach einer Neuübersetzung der Systemsoftware auch möglich, einen anderen Interrupt zu verwenden. Analog zum Aktivieren kann mit der Funktion `OSSTaskUnRegister()` der Supertask wieder deaktiviert werden.

Beim Auslösen des Supertasks werden vom Interrupthandler folgende Aktionen ausgeführt: Deaktivieren von Interrupts, Messen des Startzeitpunkts, Sichern der Register über Registersatzwechsel, Überprüfen, ob die aktuelle Instanz ausgesetzt werden soll, Aufrufen der angegebenen Handler-Funktion oder der alternativen Funktion, Zurücksichern der Register, messen des Endzeitpunkts, Aktivieren von Interrupts. Für das Deaktivieren von Interrupts während der Ausführung des Supertasks sowie für die Zeitmessung war ein tieferes Studium der Hardware notwendig.

Die Messung der Ausführungszeit muss sehr genau erfolgen. Die vom Betriebssystem vorgenommene Zeitmessung ist dafür nicht ausreichend, da sie nur mit einer Auflösung von in diesem Fall einer Millisekunde arbeitet. Für die Zeitmessung wurde die Art der Implementierung des internen Timers ausgenutzt. Dieser benutzt ein spezielles Register, in dem er einen vorgegebenen Wert pro Taktzyklus um eins erniedrigt. Sobald Null errreicht ist wird der Timer-Interrupt ausgelöst und die Zählung beginnt automatisch von vorn. Durch auslesen dieses Registers ist somit eine sehr genaue Zeitmessung möglich.

Das Sperren von Interrupts während der Ausführung des Supertasks ist ebenfalls problematisch, da in der Situation das Verlorengehen von Interrupts zu erwarten ist. Eine mögliche Lösung hierfür wäre die Verwendung eines Prolog/Epilog Schemas für die Behandlung von Interrupts. Der Interrupthandler stellt den Prolog dar, der lediglich das auftreten des Interrupts feststellt und der über einen bestimmten Mechanismus den Epilog anstößt. Die Ausführung des Supertasks würde dann im Epilog stattfinden, wobei Interrupts wieder erlaubt wären. Diese Möglichkeit bringt allerdings zusätzlichen Overhead mit sich, der für den vorliegenden Fall nicht akzeptabel ist. Im vorliegenden System gehen Interrupts dennoch nicht verloren, da diese nach globalem Reaktivieren von Interrupts signalisiert werden. Lediglich wenn die Ausführung des Supertasks so lange dauert, dass er währenddessen bereits zwei mal neu hätte ausgelöst werden sollen, geht der zweite Interrupt verloren. Gegenüber dem zusätzlichen Overhead, der für eine Behandlung solcher Situationen zu erwarten wäre, kann dieser Nachteil aber in Kauf genommen werden.

7.4 Laufzeitstatistiken

Das Aufzeichnen von Laufzeitstatistiken ist ein wichtiger Aspekt beim Einsatz des Verfahrens mit dem Aussetzen einzelner Instanzen. Dadurch ist es möglich während der Entwicklungszeit zu erkennen, ob und wenn ja wie oft der Task ausgesetzt wurde und entsprechend darauf zu reagieren. Es werden folgende Werte ermittelt und der Anwendungssoftware auf dem PC auf Abruf bereitgestellt: Die Anzahl der im letzten Intervall von einer Millisekunde ausgesetzten Instanzen des Supertasks, die Anzahl der insgesamt bereits ausgesetzten Instanzen, sowie den Fehlerwert am Ende des letzten Intervalls. Das Anwendungsprotokoll wurde entsprechend um Befehle erweitert, diese drei Werte von der Steuerungssoftware abrufen zu können. Das Anwendungsprogramm wurde um die Möglichkeit erweitert, diese Werte in Intervallen abzurufen und auf einer Konsole oder in einem Dialogfenster auszugeben.

8 Ergebnisse

8.1 Einsatz einer Simulation

Der in Abschnitt 6.5 vorgestellte Algorithmus wurde zu Testzwecken außerhalb des für seinen späteren Einsatz vorgesehenen Steuerrechners implementiert. Damit ist es möglich, auf einfache Art und Weise genauere Untersuchungen seines Verhaltens durchzuführen.

Die Implementierung geschah mit Hilfe der Interactive Data Language (IDL). Es wurden verschiedene Szenarien simuliert und Kenngrößen der von dem Algorithmus gelieferten Ergebnisse aufgezeichnet. Die Simulation wurde mit dem Ziel durchgeführt, eine Bewertung der von dem Algorithmus unter verschiedenen Bedingungen erzielten Ergebnisse durchführen zu können.

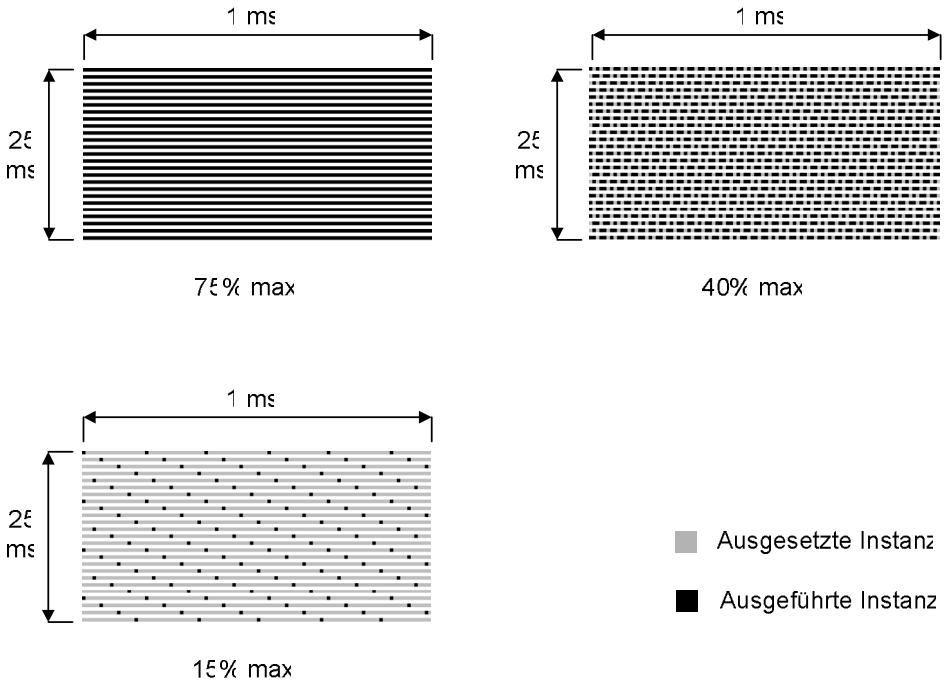


Abb. 8.1: Schema des Aussetzens des Supertasks bei konstanten Ausführungszeiten der Instanzen

In den Abbildungen 8.1 und 8.2 sind die Ausgaben des Algorithmus unter verschiedenen Bedingungen dargestellt. Die Simulationsparameter orientieren sich weitestgehend an den Gegebenheiten die auch die Steuerungssoftware des AFM vorfindet. Es wurde in beiden Fällen eine Intervalllänge von einer Millisekunde

verwendet. Die Periodenlänge des Supertasks beträgt 10 µs. Im ersten Bild haben alle Instanzen eine konstante Ausführungszeit von 5,625 µs. Im zweiten Bild sind die Ausführungszeiten normalverteilt mit einem Erwartungswert von 5,625 µs, einer Standardabweichung von 1,1 µs und einer unteren bzw. oberen Schranke von 1,25 µs respektive 10 µs. Die Simulation lief jeweils über einen Zeitraum von 25 ms, also über 25 Intervalle. Dabei wurde die vorgegebene maximale Auslastung variiert. Der Durchlauf mit konstanten Ausführungszeiten stellt allgemein das Problem eines Tasks mit unbekannter Laufzeit dar. Der Durchlauf mit normalverteilten Ausführungszeiten entspricht zusätzlich etwa dem Verhalten eines Tasks mit nichtkonstanter Laufzeit.

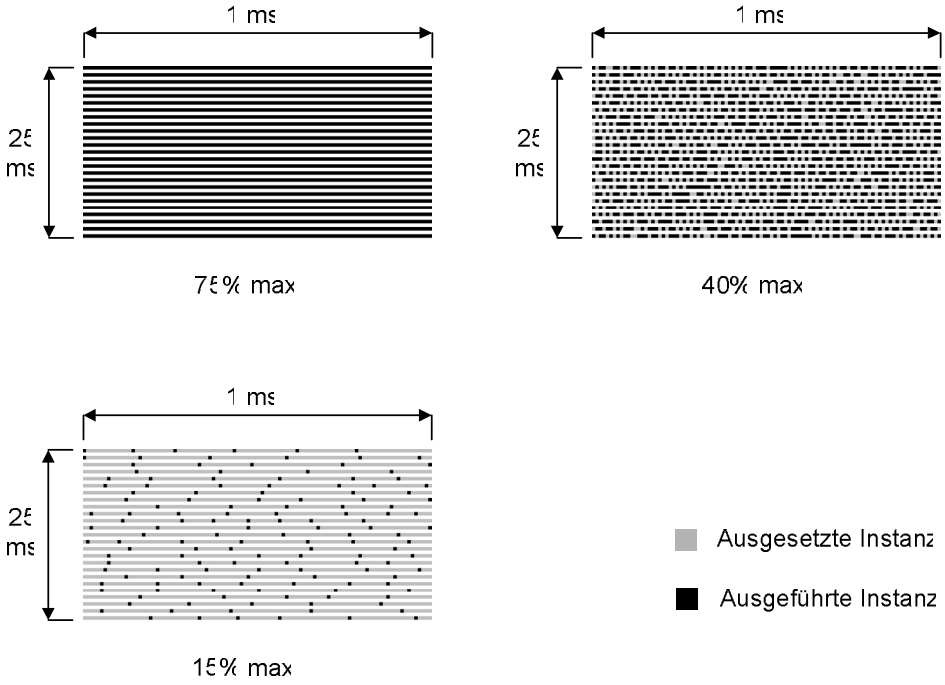


Abb. 8.2: Schema des Aussetzens des Supertasks bei normalverteilten Ausführungszeiten der Instanzen

Es ist zu erkennen, dass bei einer vorgegebenen maximalen Auslastung von 75% noch in jedem Intervall ausnahmslos alle Instanzen ausgeführt werden. Dies ist zu erkennen an durchgehenden schwarzen Linien. Bei einer Beschränkung auf 40% müssen bereits oft Instanzen ausgelassen werden. Dies ist zu erkennen an Unterbrechungen der Linien durch hellgraue Segmente. Bei einer Begrenzung der Auslastung auf nur 15% werden die meisten Instanzen nicht mehr ausgeführt. Sowohl bei 40% als auch bei 15% kann man von einer guten zeitlichen Verteilung der ausgesetzten Instanzen über jeweils ein Intervall sprechen. Besonders in den Fällen mit konstanten Ausführungszeiten ist

deutlich zu erkennen, dass es aufgrund des teilweisen Übertrags des Fehlerwertes in das nachfolgende Intervall zu keinen Diskontinuitäten im Muster kommt.

In den folgenden zwei Abbildungen werden Teilaspekte der erzielten Ergebnisse noch einmal hervorgehoben. Dabei handelt es sich, wie in Abbildung 8.3 zu sehen, um die Anzahl der ausgelassenen Task-Instanzen pro Intervall. Es ist auch hier zu erkennen, dass bei einem Limit von 75% noch keine Instanzen ausgelassen werden. Bei einer Beschränkung auf 40% werden in den meisten der 25 simulierten Intervalle ca. 37 Instanzen pro Intervall ausgelassen. Dabei ist die durchschnittliche Abweichung von diesem Wert bei der Versuchsreihe mit normalverteilten Ausführungszeiten erwartungsgemäß größer. Die Ergebnisse bei einem Limit von 15% sind entsprechend.

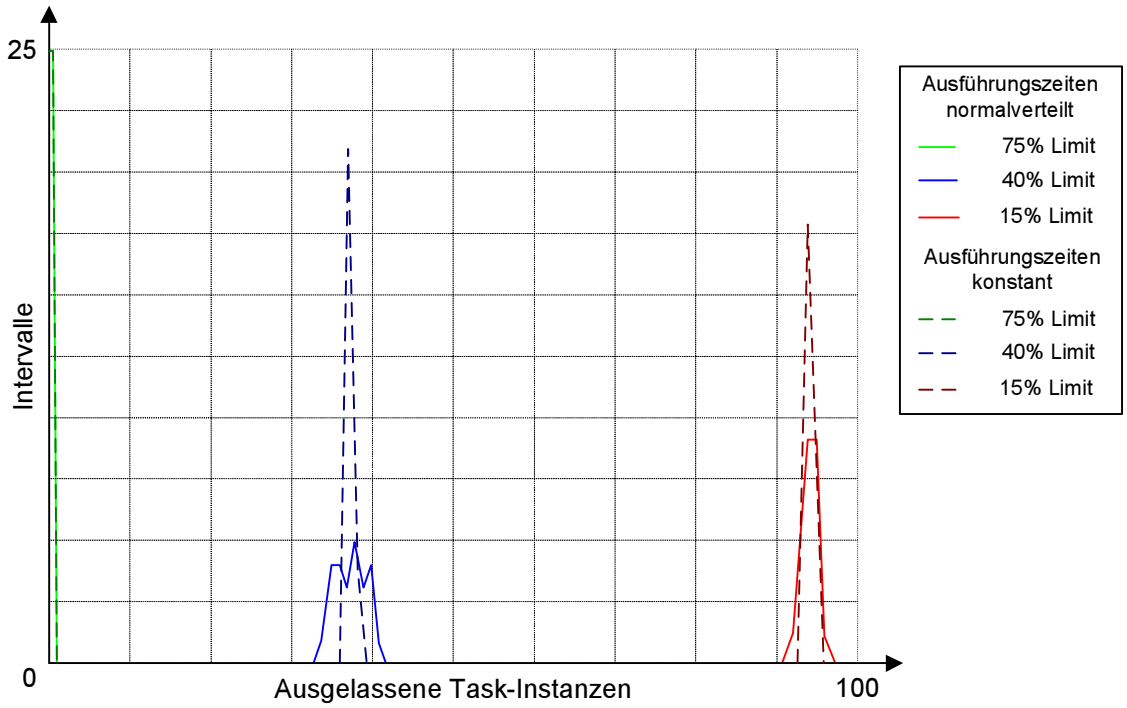


Abb. 8.3: Ausgelassene Instanzen des Supertasks pro Intervall

Die Abbildung 8.4 stellt für den Fall, dass Instanzen ausgelassen werden, die Verteilung der Größe der dadurch entstehenden Lücken dar. Dies ist insbesondere unter dem Gesichtspunkt interessant, dass bei Anwendung des Verfahrens auf den Regelungstask es zu Beschädigungen der Hardware durch zu langes Aussetzen des Tasks kommen kann. Es ist zu sehen, dass für den Fall mit einem Limit von 40% ein bis zwei Instanzen hintereinander nicht ausgeführt werden. In dem Extremfall mit einer Beschränkung auf 15% Prozessorzeit kommt es zu deutlich größeren Lücken. Deren Größe bleibt allerdings immer noch unter 30 aufeinander folgenden Instanzen. Wie in Abschnitt

6.4.4 nachgewiesen wurde, ist eine Beschädigung der Hardware erst bei einem Aussetzen des Tasks für ca. 1350 Instanzen zu erwarten.

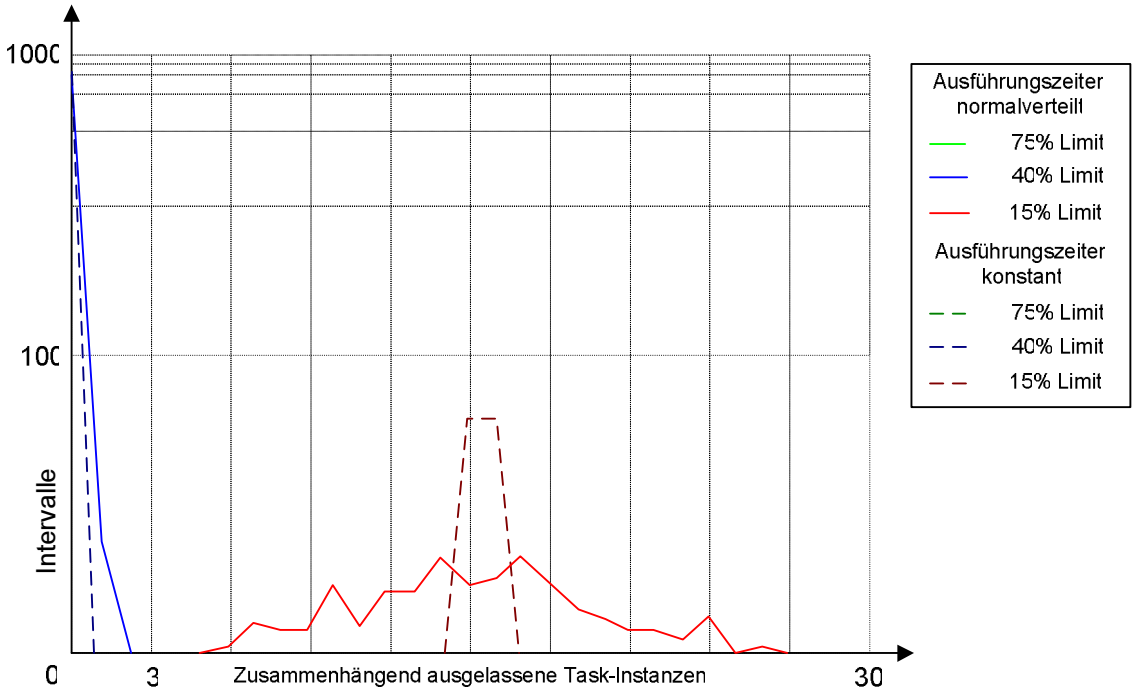


Abb. 8.4: Zusammenhängend ausgelassene Instanzen des Supertasks

8.2 Anwendung im realen System

Wie bereits im Kapitel 7 beschrieben worden ist, wurde eine Implementierung des Supertask-Konzepts sowie in diesem Zusammenhang eine Umsetzung des Verfahrens zum Aussetzen von Task-Instanzen vorgenommen. Dies erfolge mit der Zielsetzung, die prinzipielle Eignung des erarbeiteten Konzepts praktisch zu überprüfen.

Die Abbildung 8.5 zeigt zwei Feedback-Bilder, die einmal mit der bestehenden Lösung (links) und der neu implementierten Lösung (rechts) aufgenommen wurden. Im zweiten Fall wurde durch anpassen der Parameter für den Supertask ein konstantes Aussetzen der Regelung von ca. 60% der Instanzen erzwungen. Bei der Implementation der neuen Lösung wurde auf eine Anpassung des Regelalgorithmus verzichtet. Es wurde außerdem in beiden Fällen eine unüblich hohe Scangeschwindigkeit von 20 µm/s verwendet, welche der Regelung zeitlich schnellere Reaktionen als üblich abverlangt.

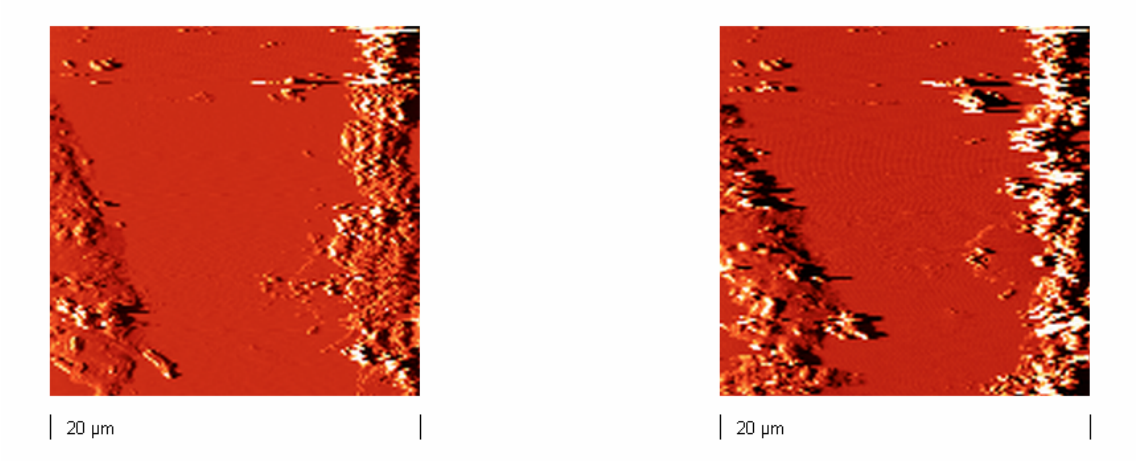


Abb. 8.5: Vergleich der Bildqualität mit und ohne Aussetzen des Regelungstasks

Die Feedback-Bilder zeigen nicht die gemessene Topografie, sondern die Verteilung der Regelabweichung bei der durchgeführten Messung. Eine möglichst niedrige Regelabweichung über das gesamte Bild deutet auf eine optimale Messung der Topografie hin. In beiden Bildern sind allerdings bereits starke Regelabweichungen (schwarz und weiß) vom Idealwert (rot) zu erkennen. Die Qualität der Aufnahme nimmt jedoch beim Aussetzen von Instanzen, wie zu erwarten war, deutlich erkennbar ab. Zu berücksichtigen ist dabei, dass in diesem konstruierten Fall die Regelung kontinuierlich bereits zu mehr als der Hälfte ausgesetzt wurde. Dies darf in einer regulären Version der Steuerungssoftware natürlich in der Form nicht vorkommen, da dann der Entwickler eindeutig einen zu rechenaufwendigen Regelalgorithmus implementiert hätte. Um diese Aufnahmen anzufertigen wurde als Probe ein leerer Objektträger, welcher allerdings starke Verunreinigungen zeigt, benutzt.

9 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein Konzept zur Gliederung der vorhandenen Steuerungssoftware in mehrere Tasks erarbeitet. Zur Verwaltung dieser Tasks wurde der Software ein geeignetes Echtzeit-Betriebssystem zur Seite gestellt. Hieraus hat sich die angestrebte, flexiblere Erweiterbarkeit der Steuerungssoftware ergeben. Das Betriebssystem wurde um die Möglichkeit zur Bereitstellung eines Tasks, der periodisch mit hoher Frequenz ausgeführt werden soll, erweitert. Der Einsatz dieses Supertasks ist für Realisierung der in dem Anwendungsfall geforderten Funktionalität notwendig.

Um Überlastsituationen aufgrund einer unbekannten oder zu niedrig angenommenen WCET des Supertasks zu vermeiden, wurde ein dynamisches Scheduling-Verfahren vorgestellt, welches mit einem schnellen Algorithmus zur Realisierung eines Akzeptanztests für einzelne Task-Instanzen kombiniert wurde. Es wurde rechnerisch gezeigt, dass dieses Verfahren für diese konkrete Anwendung bei der Abstandsregelung anwendbar ist. Weiterhin wurde diese Feststellung durch die Ergebnisse einer Simulation des Algorithmus sowie einer Implementierung auf der konkreten Hardware untermauert.

Das Aussetzen der Regelung sollte jedoch nicht der Normalfall sein, da hierdurch die Qualität der gewonnenen Aufnahmen stark negativ beeinflusst wird. Durch die Einführung einer Monitoring-Komponente wird das Führen einer Statistik über auftretende Überlastsituationen ermöglicht. Falls diese Laufzeitstatistik während der Testphase der Software zeigt, dass der implementierte Regelalgorithmus zu Zeitüberschreitungen führt und dementsprechend ausgesetzt werden musste, kann der Entwickler entsprechend reagieren und z. B. die Software modifizieren. Zur Anwendungszeit wird andererseits durch das Verfahren sichergestellt, dass, falls es z. B. durch einen Regelalgorithmus mit nicht konstanter Laufzeit zu Zeitüberschreitungen kommt, diese zu keinem Versagen der Steuerungssoftware führen. Damit konnte die Betriebssicherheit entscheidend erhöht werden.

Ein Nachteil der vorgestellten Lösung ist es, dass es durch eine fest vorgegebene Grenze der CPU-Auslastung durch den Supertask zu Situationen kommen kann, in denen der Task ausgesetzt wird, es aber gleichzeitig zu keiner vollständigen Auslastung der CPU kommt. In dieser Richtung müssten weitere Überlegungen hinsichtlich einer dynamischen Anpasuung dieser Grenze angestellt werden. Dabei wäre zu untersuchen, inwieweit eine solche dynamische Anpassung überhaupt möglich und sinnvoll ist.

Betreffs der Kommunikationlösung wurde ein Mechanismus zur Signalisierung von synchronen Ereignissen auf den beiden Teilsystemen des Steuerrechners vorgeschlagen, der zu einer für den Anwendungsfall brauchbaren Synchronisation der Teilsysteme führt. Hinsichtlich der Übertragung zeitkritischer Daten wurde festgestellt, dass unter Berücksichtigung aller Hard- und Softwarekomponenten für den anvisierten Anwendungsfall ein Best-Effort-Ansatz mit Berücksichtigung von unterschiedlichen Prioritäten der zu übertragenen Daten als am aussichtsreichsten erscheint. Vor allem vor dem Hintergrund der notwendigen Weiterverarbeitung von zeitkritischen Daten in einer nicht echtzeitfähigen Umgebung auf dem Anwendungs-PC erscheinen Versuche des Garantierens der Einhaltung harter Zeitbedingungen als nicht gerechtfertigt.

Referenzen / Literatur

[ADI 2003] Analog Devices, Inc. (Hrsg.) (2003): VisualDSP++ Development Software; http://www.analog.com/Analog_Root/productPage/productHome/0,,VISUALDSP%252B%252B,00.html; 20.12.2003

[BAM 2003] Bundesanstalt für Materialforschung und –prüfung (Hrsg.) (2003): Mikromechanik von Polymeren und Faserverbundwerkstoffen; http://www.bam.de/php/loadframe.php?a=http://www.bam.de/kompetenzen/arbeitsgebiete/abteilung_6/fachgruppe_62/laboratorium_621_i.htm; 16.12.2003.

[BDTI 2000] Berkeley Design Technology, Inc. (Hrsg.) (2003): Choosing a DSP Processor; http://www.bdti.com/articles/choose_2000.pdf; 20.12.2003

[BDTI 2003] Berkeley Design Technology, Inc. (Hrsg.) (2003): Smart Processor Picks for Consumer Media Applications; http://www.bdti.com/articles/030423ESC_processor_picks.pdf; 20.12.2003

[Becker, Gergeleit 2001] B. Becker, M. Gergeleit: Execution Environment for Dynamically Scheduling Real-Time Tasks, 22nd IEEE Real-Time Systems Symposium, WIP-Session, London, 2001

[Binnig et al. 1986] G. Binnig, C. F. Quate, C. Gerber: Atomic Force Microscope; In: Physical Review Letters 56, pp. 930-933; 1986

[Binnig et al. 1982] G. Binnig, H. Rohrer, C. Gerber, E. Weibel: Surface studies by scanning tunneling microscope; In: Physical Review Letters 49, pp. 57-61; 1982

[Campbell, 1998] Marc E. Campbell: Evaluating ASIC, DSP, and RISC Architectures for Embedded Applications. In: Languages, Compilers, and Tools for Embedded Systems. ACM SIGPLAN Workshop LCTES ’98; Montreal, Canada, 1998; pp. 261

[Chung, Liu, Lin 1990] J.-Y. Chung, J.W.S. Liu, K.-J. Lin: Scheduling Periodic Jobs that Allow Imprecise Results. In: IEEE Transactions on Computers 9, pp. 1156-1174, 1990

[Cravotta 2002] R. Cravotta: 2002 DSP Directory; http://www.module.ru/files/papers-edn040402.pdf; 4.12.2003

[ENEA 2003] Enea Embedded Technology (Hrsg.) (2003): Enea Embedded Technology : Products : OSEck; http://www.ose.com/products/product.php?product_id=130; 20.12.2003

[Eyre, Bier 2000] J. Eyre, J. Bier (2000): The Evolution of DSP Processors; http://www.bdti.com/articles/evolution.pdf; 20.12.2003

[Feindt 1994] E. G. Feindt: Regeln mit dem Rechner; München, 1994

[Gallmeister 1995] B. O. Gallmeister: POSIX.4. Programming for the Real World; Sebastopol, CA, USA, 1995

[Gergeleit et al. 1999] M. Gergeleit, E. Nett, J. Fitzner: On-line Prediction of Execution Times - A Basis for Adaptive Scheduling. In: Fourth International Workshop on Object-oriented Real-time Dependable Systems; Santa Barbara, California, 1999; pp. 186-194

[Gergeleit 2001] M. Gergeleit: A Monitoring-based Approach to Object-Oriented Real-Time Computing. Dissertation; Otto-von-Guericke Universität, Magdeburg, Deutschland, 2001

[Gorinevsky et al. 1997] Gorinevsky, Formalsky, Schneider: Force Control of Robotics Systems; CRC Press New York, 1997

[Guthold et al. 1999] M. Guthold, G. Matthews, A. Negishi, R. M. Taylor II, D. Erie, F. P. Brooks Jr., R. Superfine: Quantitative Manipulation of DNA and Viruses with the nanoManipulator Scanning Force Microscope; In: Surface Interface Analysis 27, pp. 437-443, 1999

[Hamamatsu 2003] Hamamatsu Corp. (Hrsg.): Photodiode Technical Information; http://usa.hamamatsu.com/hcpdf/techinfo/photodiode_technical_information.pdf; 20.12.2003

[Hildebrand 1992] D. Hildebrand: An Architectural Overview of QNX; In: Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, Seattle, WA, 1992

[IEEE 1998] IEEE (Hrsg.): IEEE Standard for Information Technology Standardized Application Environment Profile POSIX. Realtime Application Support; 1998

[ISO 1984] International Standards Organization (Hrsg.): Basic Reference Model for Open Systems Interconnection. ISO 7498, 1984

[JDC 2004] JDC Electronic SA (Hrsg.): SharcOS, enhanced RTOS for SHARC DSPs; http://www.jdc.ch/dsp/sharcos.htm; 8.2.2004

[Kopetz 1997] H. Kopetz: Real-Time Systems. Design Principles for Distributed Embedded Applications; Boston, Dordrecht, London, 1997

[Labrosse 2002] J. J. Labrosse: MicroC OS II. The Real Time Kernel; Lawrence, Kansas, USA, 2002

[Lapsley et al. 1996] P. Lapsley, J. Bier, A. Shoham, E. A. Lee: DSP Processor Fundamentals. Architectures and Features; IEEE Press Series on Signal Processing, 1996

[Leupers 1997] R. Leupers: Retargierbare Codeerzeugung für digitale Signalprozessoren. In: H. Fiedler, P. Gorny, W. Grass, S. Hölldobler, G. Hotz, I. Kerner, R. Reischuk (Hrsg.): Ausgezeichnete Informatikdissertationen 1997; Stuttgart, 1998

[Li, Malik 1999] Yau-Tsun Steven Li, Sharad Malik: Performance Analysis of Real-Time Embedded Software; Boston, Dordrecht, London, 1999

[Liu, Layland 1973] C. L. Liu, J. W. Layland: Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment; Journal of the Association of Computer Machinery (ACM), Vol. 20, No. 1, 1973; pp. 46-61

[Lynuxworks 2003] Lynuxworks (Hrsg.) (2003): RTOS: LynxOS embedded real-time operating system; http://www.lynuxworks.com/rtos/lynxos.php3; 25.12.2003

[Marti, Plettl 2003] O. Marti, A. Plettl (2003): Digitale Signalprozessoren; http://wwwex.physik.uni-ulm.de/lehre/PhysikalischeElektronik/Phys_Elektr/node79.html; 23.12.2003

[MathWorks 2003] The MathWorks, Inc. (Hrsg.) (2003): The MathWorks - Embedded Target for TI C6000™ DSP; http://www.mathworks.com/products/tic6000/description1.jsp; 20.12.2003

[Mentor Graphics 2004] Mentor Graphics (Hrsg.) (2004): Nucleus Real-Time Operating System; http://www.acceleratedtechnology.com/embedded/nucleus.html; 2.2.2004

[Mills 2004] D. Mills: Network Time Synchronization Project; http://www.eecis.udel.edu/%7emills/ntp.html; 10.2.2004

[Moses 1995] J. Moses: Is POSIX appropriate for embedded systems; In: Embedded Systems Programming, July 1995, p. 90

[Müller 2003] G. Müller (2003): Digitale Signalverarbeitung; http://www.geophysik.uni-frankfurt.de/geotheo/lecture/node1.html; 23.12.2003

[Navet, Migge 2003] N. Navet and J. Migge: Fine Tuning the Scheduling of Tasks through a Genetic Algorithm. Application to Posix1003.1b Compliant Systems; In: IEE Proceedings Software, IEE, vol. 150, pp13-24, 2003

[Nawab et al. 1997] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, J. T. Ludwig: Approximate Signal Processing. In: VLSI Signal Processing vol. 15, no. 1-2, 1997

[Nett et al. 1998] E. Nett, M. Gergeleit, M. Mock: An Adaptive Approach to Object-Oriented Real-Time Computing. In: Proceedings of ISORC 1998, Japan, pp. 20-24.

[NT-MDT 2002] NT-MDT Co. (Hrsg.) (2002): "GOLDEN" Silicon Probes with rectangular cantilevers (SG10, SG01, SG11 series); http://www.ntmdt.ru/Products/SPM_Accessories/AFM_Probes/SFM_Probes/product36.html; 16.12.2003.

[PI 2003] Physik Instrumente GmbH & Co. (Hrsg.) (2003): Product: Fundamentals; http://www.physikinstrumente.de/products/prdetail.php?secid=4-28; 10.12.2003

[Pohl et al. 1984] D.W. Pohl, W. Denk and M. Lanz, "Optical stethoscopy: image recording with a resolution of Lambda/20", In: Applied Physics Letters 4, 651-653 (1984).

[Rademacher 1997] M. Radmacher: Measuring the Elastic Properties of Biological Samples with the AFM; In: IEEE Engineering in Medicine and Biology; März/April 1997

[Sha et al. 1990] L. Sha, R. Rajkumar, J. P. Lehoczky: Priority Inheritance Protocols. An Approach to Real-Time Synchronization; In: IEEE Transactions on Computers, vol. 39, 1990, pp. 1175-1185

[Shannon 1949] C. E. Shannon: Communications in the presence of noise; In: Proceedings of the IRE (37), pp. 10-21, Januar 1949

[Stankovic et al. 2000] J. A. Stankovic, C. Lu, G. Tao, S. H. Son: Design and Evaluation of a Feedback Control EDF Scheduling Algorithm; http://www.artes.uu.se/events/gsconf00/papers/; 11.02.2004

[Streich 1994] H. Streich: TaskPair-Scheduling: An Approach for Dynamic Real-Time Systems, In: Proc. 2nd Workshop on Parallel and Distributed Real-Time Systems, Cancun, Mexico, 1994

[Thalhammer et al. 1997] S. Thalhammer, R. W. Stark, S. Müller, J. Wienberg, W. M. Heckl: The Atomic Force Microscope as a New Microdissecting Tool for the Generation of Genetic Probes; In: Journal of Structural Biology, 119, pp. 232-237, 1997

[vrije 2004] Vrije Universiteit Amsterdam (Hrsg.): Amoeba WWW Homepage; http://www.cs.vu.nl/pub/amoeba/; 18.1.2004

[Wang et al. 2002] Z. Wang, Y. Song, E.-M. Poggi, Y Sun: Survey of Weakly-Hard Real Time Schedule Theory and Its Application. In: International Symposium on Distributed Computing and Applications to Business, Engineering and Science - DCABES'2002; Wuxi, China, Dezember 2002

[Ward, Mellor 1993] P. T. Ward, S. J. Mellor: Strukturierte Systemanalyse von Echtzeit-Systemen; München, London, 1993

[Windriver 2003] Wind River (Hrsg.): Wind River - VxWorks 5.x; http://www.windriver.com/products/vxworks5/index.html; 25.12.2003

[Yodaiken 1997] V. Yodaiken (6.10.1997): The RT-Linux approach to hard real-time; http://rtlinux.lzu.edu.cn/documents/papers/whitepaper.html; 25.12.2003

[Zöbel, Albrecht 1995] D. Zöbel, W. Albrecht: Echtzeitsysteme. Grundlagen und Techniken; Bonn, 1995

Anhang

A Systemfunktionen von MicroC/OS-II

Tab. A.1: Task Management

Name	Funktion
OSTaskCreate()	Task erzeugen und starten
OSTaskCreateExt()	siehe OSTaskCreate()
OSTaskStkChk()	ruft Statistik über Stacknutzung eines Tasks ab
OSTaskDel()	Task löschen
OSTaskDelReq()	verzögertes Löschen eines Tasks
OSTaskChangePrio()	Priorität eines Tasks zur Laufzeit ändern
OSTaskSuspend()	Task-Ausführung anhalten
OSTaskResume()	Task-Ausführung fortsetzen
OSTaskQuery()	Informationen über einen Task abrufen

Tab. A.2: Time Management

Name	Funktion
OSTimeDly()	Ausführung für spezifizierte Anzahl Clock-Ticks aussetzen
OSTimeDlyHMSM()	Ausführung für spezifiziertes Zeitintervall aussetzen
OSTimeDlyResume()	Ausführung vorzeitig fortsetzen
OSTimeGet()	Wert des 32 Bit Clock-Tick Zählers lesen
OSTimeSet()	Wert des 32 Bit Clock-Tick Zählers schreiben

Tab. A.3: Semaphoren

Name	Funktion
OSSemCreate()	Semaphore anlegen
OSSemDel()	Semaphore löschen
OSSemPend()	auf Semaphore warten (blockierend)
OSSemPost()	Semaphore signalisieren
OSSemAccept()	siehe OSSemPend() (nicht blockierend)
OSSemQuery()	Status der Semaphore abfragen

Tab. A.4: Mutexes

Name	Funktion
OSMutexCreate()	Mutex anlegen
OSMutexDel()	Mutex löschen
OSMutexPend()	auf Mutex warten (blockierend)
OSMutexPost()	Mutex signalisieren
OSMutexAccept()	siehe OSMutexPend () (nicht blockierend)
OSMutexQuery()	Status des Mutex abfragen

Tab. A.5: Event Flags

Name	Funktion
OSFlagCreate()	Flag-Gruppe anlegen
OSFlagDel()	Flag-Gruppe löschen
OSFlagPend()	auf Signalisierungsmuster der Flag-Gruppe warten (blockierend)
OSFlagPost()	Flags der Gruppe signalisieren
OSFlagAccept()	siehe OSFlagPend() (nicht blockierend)

OSFlagQuery()	Status der Flag-Gruppe abfragen

Tab. A.6: Mailbox Management

Name	Funktion
OSMboxCreate()	Mailbox anlegen
OSMboxDel()	Mailbox löschen
OSMboxPend()	auf eingehende Nachricht warten (blockierend)
OSMboxPost()	Nachricht in Mailbox platzieren
OSMboxPostOpt()	erweiterte Version von OSMboxPost()
OSMboxAccept()	siehe OSMboxPend () (nicht blockierend)
OSMboxQuery()	Status der Mailbox abfragen

Tab. A.7: Message Queues

Name	Funktion
OSQCreate()	Message Queue anlegen
OSQDel()	Message Queue löschen
OSQPend()	auf Nachricht in der Message Queue warten
OSQPost()	Nachricht am Ende der Message Queue einfügen
OSQPostFront()	Nachricht am Anfang der Message Queue einfügen
OSQPostOpt()	erweiterte Version von OSQPost() und OSQPostOpt()
OSQAccept ()	siehe OSQPend() (nicht blockierend)
OSQFlush()	Inhalt der Queue löschen
OSQQuery()	Status der Message Queue abfragen

Tab. A.8: Memory Management

Name	Funktion
OSMemCreate()	Speicher-Partition mit angegebener Blockgröße anlegen
OSMemGet()	Speicherblock aus angegebener Partition anfordern
OSMemPut()	Speicherblock freigeben
OSMemQuery()	Informationen über Partition anfordern

B Enhanced Parallel Port Hardware

Tab. B.1: Signalbelegung am EPP

Pin	Bezeichung	Richtung	Funktion
1	Write	Out	Lese-/Schreib Selektierung
2-9	Data 0-7	In/Out	Datenleitungen
10	Interrupt	In	Unterbrechungsanforderung, Signalisieren von vorliegenden Daten
11	Wait	In	Handshake mit Peripheriegerät
12	Reserved	-	
13	Reserved	-	
14	Data Strobe	Out	Signalisiert Datentransfer
15	Reserved	-	
16	Reset	Out	Hardware Reset
17	Address Strobe	Out	Signalisiert Adresstransfer
18-25	GND	-	Masse

Tab. B.2: EPP Controller - Register

Adresse (Offset)	Funktion	Read/Write
+ 0	Daten (SPP Modus)	W
+ 1	Status {Bit 0 - Timeout; 1, 2 - reserved; 3, 4, 5 - User Define; 6 - Interrupt; 8 - Wait}	R
+ 2	Control {Bit 0 - Strobe; 1 - Data Strobe; 2 - Reset; 3 - Address Strobe; 4 - IRQE; 5 - PCD; 6, 7 - Reserved}	W
+ 3	Address Register (EPP)	R/W
+ 4	Data 1 (EPP)	R/W
+ 5	Data 2 (EPP)	R/W
+ 6	Data 3 (EPP)	R/W
+ 7	Data 4 (EPP)	R/W

C ISO/OSI-Referenzmodell

Tab. C.1: Schichten des ISO/OSI-Referenzmodells

Schicht	Bezeichnung
1	Bitübertragungsschicht (physical layer)
2	Sicherungsschicht (data link layer)
3	Netzwerkschicht (network layer)
4	Transportschicht (transport layer)
5	Sitzungsschicht (session layer)
6	Darstellungsschicht (presentation layer)
7	Anwendungsschicht (application layer)

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit selbständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

_____