Technische Universität Chemnitz-Zwickau

Fakultät für Elektrotechnik und Informationstechnik
Professur Meß- und Sensortechnik
Prof. Dr.-Ing. habil. W. Manthey

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Diplomingenieur

Thema

Programmierung einer Software zur Ansteuerung von
Richtcharakteristikmeßplätzen für Luft- bzw. Wasserultraschallwandler

eingereicht von Henrik Haftmann
geboren am 11.09.1969

Betreuer: Dipl.-Ing. E. Ahl

Hochschullehrer: Prof. Dr.-Ing. habil. W. Manthey

Chemnitz, den 27.09.1996

Ausführliche Aufgabenstellung

1. Die Überarbeitung und Korrektur der vorhandenen Software eines Luftultraschallwandlermeßplatzes. Die Programmoberfläche soll dabei mit „Turbo Vision“ unter Pascal oder C erstellt werden und mindestens den gleichen Funktionsumfang wie die vorhandene Software besitzen. Eine Anpassung des zeitkritischen Verhaltens der Software an verschiedene PC-Typen wäre wünschenswert.
2. Die Programmierung eines Windows-Treibers für die Schrittmotoransteuerung eines Meßplatzes für Wasserultraschallwandler.
3. Die Erstellung eines Programms für einen Wasserultraschallmeßplatz unter Windows. Die Software ist als DDE-Server-Anwendung zur Einbindung in die Signalverarbeitung mit Matlab zu realisieren

Sämtliche Programmlistings der zu entwickelnden bzw. zu korrigierenden Software sind in gut kommentierter Form der Arbeit als Anlage in Form eines fehlerfreien Datenträgers (z.B. Disketten 3,5") beizulegen. Eine zusätzliche Bereitstellung der gesamten Arbeit im PostScript-, WordPerfect- oder Word-für-Windows-Format ist wünschenswert.

Bibliographische Angaben

Haftmann, Henrik:
Programmierung einer Software zur Ansteuerung von Richtcharakteristikmeßplätzen für Luft- bzw. Wasserultraschallwandler.

Diplomarbeit
TU Chemnitz-Zwickau, Fakultät für Elektrotechnik und Informationstechnik
Professur Meß- und Sensortechnik, 1996
89 Seiten, 39 Abbildungen, 11 Tabellen, 4 Anlagen, 1 Diskette

Kurzreferat

Im Rahmen der Arbeit werden Programme für die Ansteuerung von Richtcharakteristik-Meßplätzen unter Microsoft® Windows™ erstellt. Die Programmentwicklung erfolgt für Luft- und Wassermeßplatz aufgrund der verschiedenen Programmanforderungen getrennt. Echtzeitforderungen werden durch spezielle Programme erfüllt.

Für den Wassermeßplatz wird ausgehend von einer existierenden Basis, dem vorhandenen Meßplatz, einem DOS-Motortreiber und einer DOS-Ansteuersoftware, ein dreiteiliges Programm-Set erstellt, welches aus den Teilen Ansteuerung des Ultraschallsenders, Virtueller Gerätetreiber für die Motoransteuerung sowie Ansteuerung des Positioniermotors über diesen Treiber besteht.

Am Luftmeßplatz wird die Hardware den neuen Erfordernissen angepaßt sowie die vorhandene DOS-Software nach Windows™ portiert. Es entsteht ein Komplettprogramm zur Ansteuerung des gesamten Meßplatzes unter Verwendung eines Virtuellen Gerätetreibers zur exakten Festlegung programmierter Wartezeiten.

Für alle 4 Programme wird je eine Hilfedatei angefertigt, die kontextsensitiv an das jeweilige Programm gebunden wird und sowohl jede Funktion einzeln erklärt als auch einen Programmüberblick und Hinweise zur Fehlerbehebung und zur Installation/Deinstallation beinhaltet.

Diese Arbeit dokumentiert den vorhandenen Stand, gibt einen Einblick in zwei spezielle Aspekte der Windows-Programmierung und dokumentiert Lösungswege und Endergebnisse für die oben genannten Programme. Die gewählte Programmiersprache ist Borland Pascal 7.0 mit Objekten sowie Turbo Assembler.

Selbständigkeitserklärung

Hiermit versichere ich, daß ich die von mir eingereichte Diplomarbeit selbständig und unter Verwendung der angegebenen Hilfsmittel sowie der Hinweise meines Betreuers angefertigt habe.

Chemnitz, den 21. September 1996

Inhaltsverzeichnis

Abkürzungsverzeichnis 7

1 Einleitung 9

2 Ausgangspunkte und Problemstellung 10
2.1 Der Wassermeßplatz 10
2.1.1 Positioniermotor 11
2.1.2 Ansteuerung des Wandlers 12
2.1.3 Meßwertaufnahme 13
2.1.4 Vorhandene Treibersoftware 13
2.1.5 Vorhandene Ansteuer-Oberfläche 15
2.2 Der Luftmeßplatz 15
2.2.1 Positioniermotor 15
2.2.2 Analog-Digital-Wandlerkarte 16
2.2.3 Der Spitzenwertgleichrichter und der zeitliche Ablauf der Ultraschallmessung 18
2.2.4 Vorhandene Software 19
2.3 Einführung zu VxD's 19
2.3.1 Was sind VxD's? 20
2.3.2 Erstellung und Programmierung 20
2.3.3 Grundlegende Arbeitsweise von VxD's im Kernel 21
2.3.4 Erweiterte Möglichkeiten bei der VxD-Programmierung 26
2.3.5 VxD's unter Windows 95 und Ausblick auf andere Systeme 31
2.4 Einführung zu DDE 33
2.4.1 Zwischenablage-Formate als Basis für den Datenaustausch 33
2.4.2 Datenaustausch per DDE 33
2.4.3 Nachrichtenbasierter Datenaustausch per DDE 34
2.4.4 Datenaustausch über die DDE-Management-Bibliothek DDEML 38

3 Umsetzung der Programmieraufgaben 41
3.1 Treiber für Positioniermotor MPK3D.386 41
3.1.1 Mögliche Ansteuertechniken 41
3.1.2 Kurvenform 42
3.1.3 Realisierung des Virtuellen Gerätetreibers 42
3.1.4 Funktioneller Aufbau 43
3.1.5 Bereitgestellte Schnittstellenfunktionen und Installation 46
3.2 DDE-Server für Positioniermotor 46
3.2.1 Ansteuerung des Gerätetreibers 48
3.2.2 Oberfläche 48
3.2.3 Dialoge 49
3.2.4 DDE-Kommunikation 51
3.3 DDE-Server für Wandler-Ansteuereinheit 53
3.3.1 Ansteuerung des Gerätes 54
3.3.2 Oberfläche 58
3.3.3 Log-Anzeige 59
3.3.4 Dateneingabe in Dialogen 59
3.3.5 DDE-Kommunikation 60
3.4 Komplettpaket für den Luftmeßplatz 61
3.4.1 Oberfläche 64
3.4.2 Meßfunktion und Timing - das Fenster „ADC“ 65
3.4.3 Ansteuerung des Motors - das Fenster „Motor“ 66
3.4.4 Datenaufnahme in einzelne MDI-Fenster 69
3.4.5 Datendarstellung 69
3.4.6 Laden, Speichern und das Dateiformat 70
3.4.7 Dialoge 70

3.4.8 DDE-Konversation 71
3.5 Hilfe-Dateien 75
3.5.1 Erstellung 75

4 Ergebnisse 78
4.1 Anwendung der Software am Wassermeßplatz 78
4.1.1 Elektromagnetische Verträglichkeit 78
4.2 Anwendung der Software am Luftmeßplatz 78
4.3 Wiederverwendbarkeit der Software 79

Verzeichnisse 80
Literaturverzeichnis 80
Abbildungs- und Tabellenverzeichnis 81

Anhang 83
A1 Technische Daten von Sendeendstufe und Spitzenwertgleichrichter für Luftmeßplatz 83
A2 Technische Daten AD-12-Bit-Karte für Luftmeßplatz 84
A3 Formular zum Erhalt einer VxD-ID 85
A4 Verzeichnisstruktur und Dateiliste der Programmdiskette 86

Abkürzungsverzeichnis

ADC	Analogue-Digital-Converter	Analog-Digital-Wandler, -Umsetzer, ADU
API	Application Programmer's Interface	Schnittstelle für den Anwendungsprogrammierer
ASCII	American Standard Code of Information Interchange	Amerikanischer Standard-Kode zum Informationsaustausch
AT	Advanced Technology	PC mit 80286 und Festplatte
BIOS	Basic Input Output System	Grundlegendes Ein-/Ausgabe-System
DAC	Digital-Analogue-Converter	Digital-Analog-Wandler, -Umsetzer, DAU
DDB	Device Descriptor Block	Gerätetreiber-Beschreibungsblock
DDE	Dynamic Data Exchange	Dynamischer Datenaustausch
DDK	Driver Development Kit	Treiber-Entwicklungs-(Werkzeug-)Satz
DLL	Dynamic Link Library	Dynamische Link-Bibliothek
DMA	Direct Memory Access	Direkter Speicherzugriff
DOS	Disk Operating System	Disketten- (und Platten-) Betriebssystem
DPMI	DOS Protected Mode Interface	Schnittstelle zu DOS für Protected-Mode-Programme
DRV	Driver	Treiber
DSO	Digital Sampling Oscilloscope	Digitales Sampling-Oszilloskop
DTP	DeskTop Publishing	DTP, „Publizieren vom Schreibtisch“ (mit Computereinsatz)
DTR	Data Terminal Ready	Datenendgerät bereit (serielle Schnittstellenleitung)
EMS	Expanded Memory Specification	Expansionsspeicher-Festlegung
EXE	Executable	Ausführbares (Programm in Maschinencode)
FSR	Fault Service Routine	Fehler-Behandlungsprogramm
GDT	Global Descriptor Table	Globale Deskriptortabelle
GPF	General Protection Fault	Allgemeiner Schutzfehler (oft: …Schutzverletzung)
GPIB	General Purpose Interface Bus	Allzweck-Busschnittstelle
HC	Help Compiler	Hilfe-Compiler
HWE		Hardwareendschalter, Referenztaster
ICE	In-Circuit Emulator	hier: Hardware-Debugger
ID	(unique) Identifier	Kennzahl, eineindeutiger Bezeichner
Int	Interrupt	Interrupt, Unterbrechung
IOPM	In/Out Permission Map	Ein-/Ausgabe-Erlaubnis-Liste
ISR	Interrupt Service Routine	Interrupt-Behandlungsprogramm
LAN	Local Area Network	Lokales Netzwerk
LSB	Least Significant Bit	Niederwertigstes Bit (am ADC: kleinste Spannungsstufe)
LDT	Local Descriptor Table	Lokale Deskriptortabelle
KB	kilobyte	Kilobyte, 1024 Byte
MDI	Multi-Document Interface	Mehrdokumenten-Schnittstelle
MMGR	Memory Manager	Speicher-Manager
MSB	Most Significant Bit	Höchstwertiges Bit
MSDN	MicroSoft Developer Network	Microsoft Entwickler-Netzwerk
NC	Numeric Controlled	numerisch gesteuert (i.a. rechnergesteuert)
nil	Not In List	Nicht In Liste (Symbolischer Zeiger, der nirgendwohin zeigt)
OEM	Original Equipment of Manufacturer	Originalausrüstung des Herstellers (hier: IBM-Zeichensatz)
OLE	Object Linking and Embedding	Objektverknüpfung und -einbettung
PC	Personal Computer	PC, Persönlicher Computer
PIA	Programmable Interface Adaptor	Programmierbare Ein-/Ausgabeeinheit
PIC	Programmable Interrupt Controller	Programmierbarer Interrupt-Controller
PIF	Program Information File	Programm-Informations-Datei (für DOS-Programme)
RIP	“Rest In Peace”, Fatal Exit (SDK)	„Ruhe in Frieden“, Schwerer Fehler

RTF	Rich Text Format	Rich-Text-Format, ein Dokumenten-Austauschformat
RTS	Request To Send	Sende-Anforderung (serielle Schnittstellenleitung)
S&H	Sample And Hold	Abnehmen-und-Halten, Sample&Hold
SAA	Standard Application Architecture	Standard-Anwendungsaussehen und -bedienung
SDI	Single-Document Interface	Einzeldokument-Schnittstelle
SDK	Software Development Kit	Software-Entwicklungs-(Werkzeug-)Satz
SWG		Spitzenwertgleichrichter
TSR	Terminate and Stay Resident	Beenden und resident (im Speicher) verbleiben
TSR	Trap Service Routine	Fallen-Behandlungsprogramm
TSS	Task State Segment	Task-Statussegment
TTL	Transistor Transistor Logic	Transistor-Transistor-Logik (Pegelnorm)
UAE	Unrecoverable Application Error	Nicht behebbarer Fehler im Anwendungsprogramm
VCO	Voltage Controlled Oscillator	Spannungsgesteuerter Oszillator
VM	Virtual Machine	Virtuelle Maschine (Prozeß)
VMM	Virtual Machine Manager	Verwalter für Virtuelle Maschinen
V86	Virtual-8086	Virtueller 8086 (Betriebsmodus)
VxD	Virtual *(something)* Device	Virtueller Gerätetreiber (für Windows)
WAN	Wide (or World) Area Network	Globales Netzwerk
WOA	Windows Old Application	DOS-Programm (unter Windows)
WOW	Windows-On-Win32	Emulator für Windows-16bit-Programme unter Windows NT
XT	Extended Technology	PC mit 8086 und Festplatte
XMS	Extended Memory Specification	Erweiterungsspeicher-Festlegung

1 Einleitung

Ultraschall-Meßverfahren haben in Wissenschaft und Technik eine weite Verbreitung gefunden. Dabei werden bestimmte Vorzüge ausgenutzt, wie die Fähigkeit, undurchsichtige Stoffe zu durchdringen, sowie die leichte Bündelfähigkeit von Ultraschall zu Strahlen.

Bisher waren zur Erfassung linearer, flächiger oder räumlicher Gebilde Geräte mit mechanisch bewegten Teilen erforderlich, um die Schallausbreitungsrichtung zu variieren. Durch den Einsatz von Sensor-Arrays können bewegte Teile entfallen und die Meßgenauigkeit erhöht werden.

Zur Erfassung von Richtcharakteristiken solcher Arrays wurden im Vorfeld automatisierte Meßplätze aufgebaut. Konkret handelt es sich dabei um einen Wasser- und einen Luftultraschallmeßplatz. Im Rahmen dieser Arbeit wird die erforderliche Ansteuer-Software für Windows erstellt.

Zunächst erfolgt eine technische Beschreibung der beiden vorhandenen Meßplätze. Danach werden zwei spezielle Aspekte der Windows-Programmierung, die Erstellung und Funktion Virtueller Gerätetreiber sowie der Dynamische Datenaustausch, erläutert. Diese Abschnitte führen in vorhandene Sachverhalte ein und bilden den Ausgangspunkt der Programmierarbeit. Insbesondere mit den Virtuellen Gerätetreibern kann das Einsatzfeld von Windows in Bereiche ausgedehnt werden, die bisher eine Domäne von Single-Task-Betriebssystemen waren.

Danach wird die Umsetzung jeder einzelnen Programmieraufgabe erläutert. Dabei führt die Software-Entwicklung für den Wassermeßplatz zu einem dreiteiligen Programm-Set. Spezielle Belange der Echtzeitverarbeitung unter Windows werden dabei besonders berücksichtigt und ausführlich dokumentiert. Bei der Realisierung des Schrittmotor-Treibers wird ein neuartiges Programmierkonzept verwendet und erläutert. Für den Luftmeßplatz wird ein Komplettprogramm entwickelt. Auch hierbei wird besonderes Augenmerk auf die Realisierung zeitkritischer Programmteile gelegt. Ein weiterer Abschnitt erläutert die Erstellung von Hilfe-Dateien, die für jedes der vier Programme angefertigt wurden.

Das vierte Kapitel dokumentiert die Arbeitsergebnisse, wie sie im gegebenen Zeitrahmen feststellbar sind. Verbleibende Probleme werden aufgezeigt und Lösungsvorschläge gegeben.

Im folgenden genannte Produktnamen werden ohne Copyrightvermerk genannt. Im übrigen ist unter „Windows“ immer „Microsoft Windows“™ oder „MS-Windows“ zu verstehen, falls nicht anders genannt. In der Regel bezieht sich diese Arbeit ausschließlich auf die Windows-Versionen 3.1 sowie 3.11 für Workgroups. Verweise auf andere Versionen von Windows werden gesondert vermerkt.

2 Ausgangspunkte und Problemstellung

Entsprechend der Aufgabenstellung unterteilt sich die Diplomarbeit in vier relativ unabhängige Projekte. Für das Verständnis der Zusammenhänge und Programmierambitionen werden im folgenden die beiden Ultraschallmeßplätze für Luft und Wasser genauer vorgestellt. Danach werden zwei für diese Arbeit grundlegende Windows-Programmierkonzepte, die Arbeit mit Dynamischem Datenaustausch und mit Virtuellen Gerätetreibern, aufgezeigt. Für das Verständnis der Programme sei vorausgesetzt, daß die normale Windows-Programmierung, z. B. mit Borland Pascal, bekannt ist.

2.1 Der Wassermeßplatz

Der Meßplatz hat den im Bild 2.1 gezeigten Aufbau. Der Laborrechner spielt eine zentrale Rolle beim Meßvorgang.

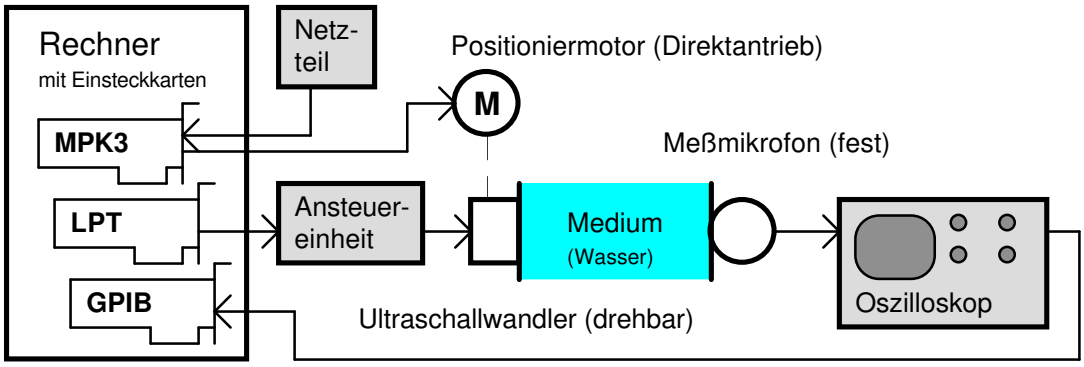


Bild 2.1: Gerätetechnischer Aufbau des Wassermeßplatzes

Hierzu sollen verschiedene Softwaremodule entwickelt werden, die reibungslos miteinander zusammenarbeiten und komfortabel zu bedienen sind.

Dabei soll die Softwarelösung in der Lage sein, die Meßdaten zum Softwarepaket Matlab zu transferieren. Unter DOS benötigt man ein Programm, das alle drei Komponenten des Meßplatzes (Positionierung des Motors, Aussenden von Impulsen, Aufnahme von Meßwerten) vereinigt und die Meßwerte in eine von Matlab lesbare Datei schreibt. Die mächtige Skriptsprache von Matlab und das DDE von Windows können unter DOS nicht genutzt werden. Daher steht das Ziel, die Meßwertaufnahme und die zugehörigen Prozesse gänzlich nach Windows zu verlagern.

2.1.1 Positioniermotor

Für die rotatorische Positionierung wird ein Motor von der Firma *isel-automation* zusammen mit einer zugehörigen Mikroschritt-Steuerkarte eingesetzt. Der Motor ist ein permanenterregter Schrittmotor mit 2 Strängen, hat intern 50 Pole („Raststellungen“ eines erregten Motors) und demnach 200 Vollschritte. Die maximale Mikroschrittzahl ergibt sich, wenn der Spulenstrom in der Nähe des Nulldurchgangs pro Mikroschritt um die kleinste erreichbare Stufe (1 LSB) springt. In Verbindung mit der Steuerkarte, die den Spulenstrom in 128 äquidistanten Stufen einzustellen vermag, sind bei Sinusansteuerung theoretisch $64\pi \approx 200$ Mikroschritte pro Vollschritt möglich (Tangente mit oben besagtem Anstieg im Nulldurchgang angenommen), die jedoch aufgrund unvermeidlicher Diskretisierungsfehler nicht äquidistant sind. Das ergibt eine Vollwinkelauflösung von 40000 Mikroschritten, d.h. der minimale Positionierwinkel beträgt rechnerisch 0,009°. Der Originaltreiber MPK3DRV teilt einen Vollschritt in maximal 80 Mikroschritte ein, das ergibt dann 16000 Mikroschritte pro Umlauf oder eine Positionierfeinheit von rechnerisch 0,0225°. Die absolute Genauigkeit, bezogen auf die Summe mehrerer aufeinanderfolgender Mikroschritte innerhalb eines Vollschritts, liegt offenbar noch erheblich darunter, da die Ansteuerung mit Sinusfunktionen nicht auf den Motor abgestimmt ist, siehe hierzu auch Abschnitt 3.1.2. Dabei handelt es sich um einen systematischen, also kompensierbaren Fehler. Hinzu kommen systematische Fehler durch Reibung und Achsenbelastung sowie zufällige durch Reibungsvariation und Diskretisierung.

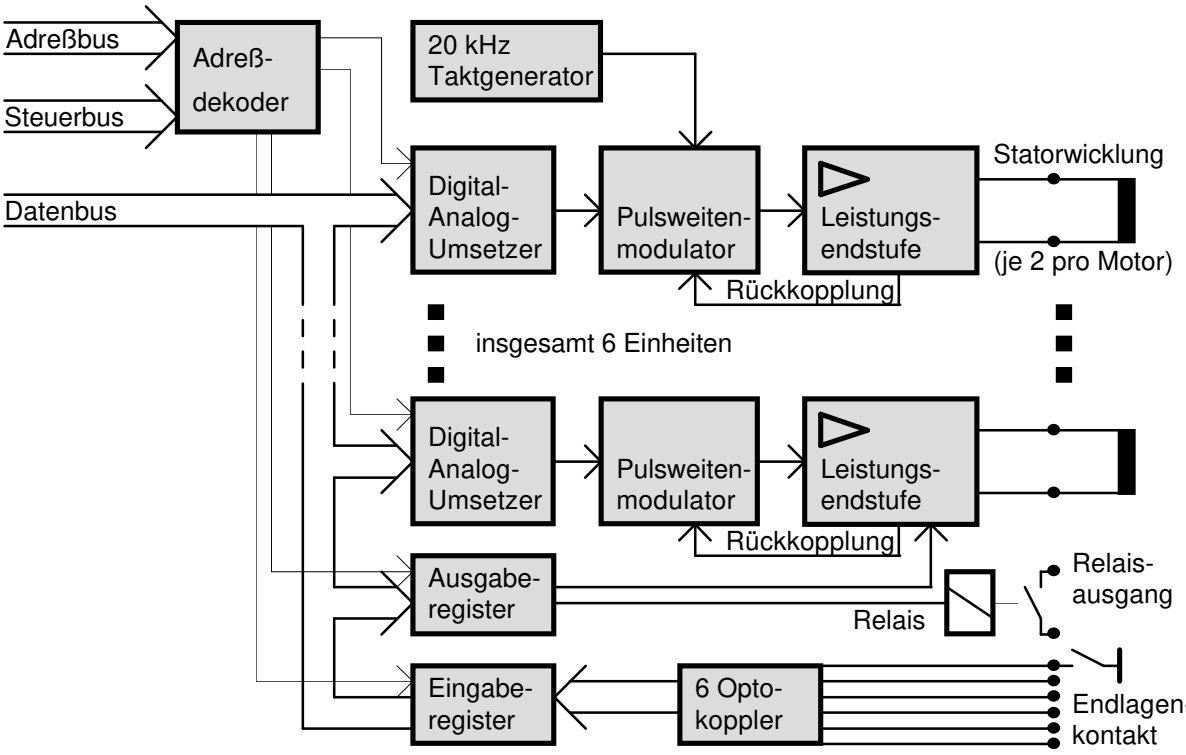


Bild 2.2: Etwaiger interner Aufbau der isel-Mikroschritt-Steuerkarte

Die zugehörige *isel*-Microstep-Steuerkarte MPK3 ist eine Einsteckkarte für einen PC und besitzt alle Komponenten zum Betreiben von drei Zwei-Strang-Schrittmotoren im Mikroschrittbetrieb inklusive Leistungsendstufen mit einer Stromergiebigkeit von 2A (siehe Bild 2.2). Dabei arbeitet die Leistungsendstufe mit einem Stromfühler, der über die Rückkopplung die Einschaltzeit des pulsweitenmodulierten Signals beeinflußt. Das ergibt eine Konstantstromspeisung, die hohe Drehmomente auch bei hohen Drehzahlen erlaubt. Außerdem besitzt die Karte sechs opto-entkoppelte Endschalter-Eingänge. Je zwei pro Motor dienen zum Anschluß von Lagegebern. Zusätzlich ist ein universeller Niederspannungs-Relaisausgang vorhanden . Gedacht ist diese Karte zur Ansteuerung einer NC-Maschine mit 3 linearen Achsen (3 Freiheitsgrade) und 1 rotierenden Werkzeug (Bohrer oder Fräse), die ebenfalls von *isel* angeboten wird. Die Speisung der Leistungsendstufen auf der Karte erfolgt durch ein externes Netzteil.

Der Mikroschrittbetrieb, in der Literatur [Fisch] auch als Minischrittbetrieb bezeichnet, zeichnet sich durch viele Vorteile in mechanischer Hinsicht aus:

- Vermeiden ruckartiger Bewegungen bei kleinen Geschwindigkeiten, wie sie insbesondere bei synchroner Bewegung mehrerer Schrittmotoren bei kleinen Steigungen auftreten
- Verringerung der Geräuschentwicklung
- Verringerung von Resonanzerscheinungen
- Sanfterer Anlauf und Stop

Erkauft werden diese Vorteile durch erhöhten elektronischen Ansteueraufwand. So sind für jeden Motor 2 Digital-Analog-Umsetzer notwendig. Auf der *isel*-MPK3-Karte wurde die Konstantstrom-Impulsansteuerung eingesetzt, die einen optimalen energetischen Wirkungsgrad erreicht, jedoch bezüglich elektromagnetischer Verträglichkeit an einem Meßplatz unangenehme Störungen erzeugt und über die Motor-Anschlußleitungen verbreitet. Die Pulsfrequenz ist kartenintern auf ca. 20kHz festgelegt.

2.1.2 Ansteuerung des Wandlers

Die Ansteuerung des Ultraschallwandlers erfolgt über eine von [Kräm] entwickelte externe Schaltung, die am Druckerport des Rechners anschließbar ist. Die Schaltung ist Ergebnis einer vorhergehenden Studienarbeit. Sie wurde dabei derart konzipiert, daß sie sich wie ein normaler Drucker verhält, d.h. das Signalspiel einer Centronics-Schnittstelle einhält. Somit können Standard-Systemfunktionen zur Ansteuerung des Wandlers benutzt werden, wie z.B. die DOS-Schnittstelle Int 21h, die BIOS-Schnittstelle Int 17h oder die Windows-DLL-Schnittstelle in der USER.EXE. Als weiterer Vorteil ist zu nennen, daß diese Schaltung auch an nahezu allen anderen Computersystemen Verwendung finden kann.

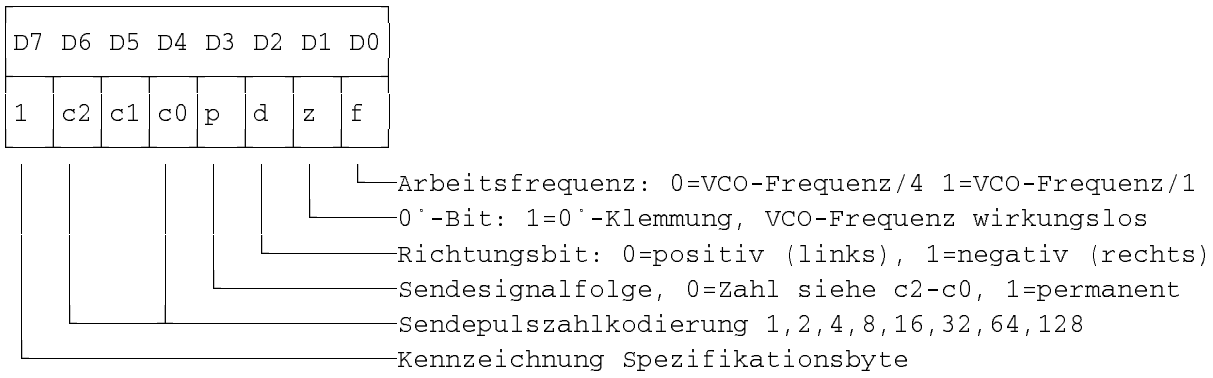
Das Druckerport ist eine unidirektionale 8-bit-parallele Schnittstelle (D1..D8) mit Handshake-Leitungen (/STROBE; /ACK, BUSY) zur Synchronisation der Datenübertragung sowie sechs weiteren Meldeleitungen (/INIT, /AF, /SELIN; PE, /ERROR, /SEL). Die Schaltung verarbeitet nur 2 verschiedene Datenbytes. Die Unterscheidung zwischen diesen beiden erfolgt anhand des höchstwertigen Bits (MSB) des jeweils übertragenen Bytes. Somit verbleiben für die eigentliche Informationsübertragung je 7 bit, insgesamt 14 bit.

Das erste Byte ist das *Frequenzdatenbyte*, eine vorzeichenlose 7-bit-Zahl, die direkt an den Eingang des Digital-Analog-Umsetzer (DAC) auf der Schaltung gelegt wird. Kennzeichen für dieses Byte ist eine Null als höchstwertiges Bit:

D7	D6	D5	D4	D3	D2	D1	D0
0	a6	a5	a4	a3	a2	a1	a0

Der DAC steuert mit seinem analogen Ausgang einen spannungsgesteuerten Oszillator (VCO), der die gewünschte Frequenz für ein Schieberegister erzeugt. Dieses wiederum bewirkt in Verbindung mit den daran angeschlossenen Endstufen eine zeitversetzte Ansteuerung der Einzelwandler der Wandler-Anordnung . Der Zusammenhang zwischen dem Digitalwert und der VCO-Generatorfrequenz ist nichtlinear. Die Kennlinie wurde in einer vorhergehenden Diplomarbeit [Kräm] aufgezeichnet und in das ursprüngliche Programm fest eingebunden.

Das zweite Byte ist das *Spezifikationsbyte* mit einer 1 als Kennzeichen im MSB, und es enthält Einzelbits in folgender Bedeutung:



Das Richtungsbit legt das Vorzeichen des Abstrahlwinkels fest. Frontal zum Wandler liegt der Winkel 0, links davon (von oben betrachtet) liegen positive, rechts davon negative Winkelgrade, wie es dem mathematisch positiven Drehsinn entspricht. Zu beachten ist, daß sich die Motorachse genau entgegengesetzt diesem Verhalten bewegt, d.h. die Achse dreht sich bei Zunahme des Winkels rechtsherum, also im Uhrzeigersinn.

2.1.3 Meßwertaufnahme

Die Meßwertaufnahme erfolgt durch ein Oszilloskop, entweder manuell durch Ablesen und Dateneingabe oder automatisch mit einem fernsteuerbaren Digital-Oszilloskop. Solche Oszilloskope weisen fast immer eine GPIB (General Purpose Interface Bus, Vielzweck-Busschnittstelle)-Schnittstelle auf, die für hohe Datenübertragungsraten in beide Richtungen geeignet ist. Im Rechner ist hierzu eine GPIB-Karte erforderlich. Die zugehörige Treibersoftware für DOS und Windows wurde vom Kartenhersteller mitgeliefert.

Zur komfortablen Ansteuerung des Oszilloskops unter Windows wurde im Vorfeld das Programm GPIB-DSO geschrieben, welches auch als Server (Diener) im Dynamischen Datenaustausch (DDE) fungieren kann [Ahl]. Damit kann ein Matlab-Skript als DDE-Client (Kunde) die Datenaufnahme steuern.

2.1.4 Vorhandene Treibersoftware

Ausgangspunkt war die Software MPK3DRV.EXE Version 1.1 vom 21.Februar 1995, die von der Firma *isel* zur Mikroschritt-Steuerkarte mitgeliefert wurde. Es handelt sich dabei um ein reines DOS-Programm, welches als TSR (Terminate and Stay Resident) konzipiert und programmiert wurde. Es belegt ca. 50KB im Hauptspeicher und verbraucht laut Dokumentation soviel Rechenleistung, daß mindestens ein 386/40MHz erforderlich ist, um den Treiber absturzfrei zu installieren [isel1]. Der Treiber kapselt die komplizierte Ansteuerung der 3 Schrittmotoren für Geraden-, Kreis- und Helixbewegungen und stellt seine Funktionen über einen dedizierten Softwareinterrupt, namentlich Int78h, zur Verfügung. Für die Benutzung am Meßplatz ist nur die Bewegung einer Achse nötig.

Dieser Treiber wurde relativ fest für die im Abschnitt Dabei soll die Softwarelösung in der Lage sein, die Meßdaten zum Softwarepaket Matlab zu transferieren. Unter DOS benötigt man ein Programm, das alle drei Komponenten des Meßplatzes (Positionierung des Motors, Aussenden von Impulsen, Aufnahme von Meßwerten) vereinigt und die Meßwerte in eine von Matlab lesbare Datei schreibt. Die mächtige Skriptsprache von Matlab und das DDE von Windows können unter DOS nicht genutzt werden. Daher steht das Ziel, die Meßwertaufnahme und die zugehörigen Prozesse gänzlich nach Windows zu verlagern.

2.1.1 erwähnte Werkzeugmaschine konzipiert. Da der Treiber seine normale Arbeit erst dann aufnimmt, wenn alle drei Achsen eine Referenzfahrt zum Endlagenschalter durchgeführt haben, mußte eine Emulationsschaltung entwickelt werden, die in zeitlich richtiger Reihenfolge entsprechende Signale generiert. Alternativ hätte man den Treiber gezielt modifizieren können – die Stellen zu finden ist jedoch bei einer 50KB großen, in Hochsprache geschriebenen Datei sehr kompliziert.

Außerdem darf im normalen Betrieb keiner der Endlagenschalter angefahren werden, da sich der Treiber dann deaktiviert, und eine neue Referenzfahrt erzwungen wird. Diese für translatorische Antriebe sinnvolle Maßnahme ist für die rotatorische Positionierung mit über 360° Drehwinkel hinderlich. Daher wurde der freie Relaisausgang der Karte dazu verwendet, den Endlagenkontakt ein- und auszuschalten.

Erst die zum Zeitraum dieser Diplomarbeit erhältliche Version 1.2 des Schrittmotortreibers vom 9.Mai 1996 ermöglicht die softwaremäßige Deaktivierung einzelner Achsen, womit die Emulationsschaltung künftig nicht mehr benötigt wird. Nur der Ein-/Ausschalter für den Endlagenschalter ist bei Verwendung des neuen Treibers weiterhin vonnöten.

Beide Programmversionen bereiten jedoch Probleme, wenn sie unter Windows eingesetzt werden sollen. Laut Dokumentation ist der Treiber weder unter Windows (was bedeutet, den Treiber vor dem Start von Windows zu installieren) noch in einer DOS-Box einsetzbar.

Dennoch war es mit mäßigem Erfolg möglich, den Treiber in einer DOS-Box zu betreiben. Um eine flüssige Verarbeitung zu realisieren, mußte in der .PIF-Datei (PIF= Programm-Informations-Datei; Hilfsdatei für Windows, die Start-Einstellungen einer DOS-Box enthält) eine hohe Hintergrundpriorität eingestellt werden. Außerdem mußte der Schalter „Leerlaufzeit entdecken“ abgeschaltet werden. Trotzdem war das Ergebnis ein stotternder Motor, dessen Bewegungs-Unregelmäßigkeiten zunahmen, wenn Windows-Applikationen arbeiteten, sowie ein sehr „zäh“ arbeitendes Windows.

Den Treiber vor dem Start von Windows zu benutzen war nicht möglich. Offenbar ist ein Programmfehler daran schuld, daß es beim Versuch, danach Windows zu starten, auf vielen Rechnern zum Absturz kommt.

Damit entstand ein neues Problem: Wie eine Windows-Anwendung mit dem Treiber in der DOS-Box kommunizieren soll. Aufgrund der Trennung in zwei Virtuelle Maschinen (VM's) mit gegenseitig abgeschirmten Adreßräumen führt ein Software-Interruptaufruf seitens des Windows-Programms nicht zum Ziel (in die DOS-Box hinein). Erforderlich wurde eine Hilfs-Software, die in beiden VM's sichtbar ist und dazu dient, Anforderungen durchzureichen, um wenigstens eine Synchronisation zu erreichen. Da ein vor Windows geladenes TSR-Programm (ein speicherresidentes DOS-Programm) diese Forderung erfüllen kann, wurde ein solches TSR-Programm erstellt.

Insgesamt hatte man eine Lösung, die funktionierte, aber:

- die Lösung ist mit den vielen erforderlichen Installationsschritten schwer handhabbar
- die Pflege der vielen zusammengehörigen Software-Teile ist sehr kompliziert und keineswegs zukunftsträchtig
- der Motor stottert; man muß das Windows sehr vorsichtig handhaben, um Schrittfehler zu vermeiden. Insbesondere ist das Starten größerer Applikationen problematisch
- das Stottern des Motors führt zu einer stärkeren mechanischen Beanspruchung der an der Motorachse befestigten sensiblen Meßanordnung
- Windows wird stark in seiner Leistungsfähigkeit eingeschränkt, es läuft zäh
- die Funktionen, die der Treiber zur Verfügung stellt, sind sehr mächtig. Dieser Leistungsumfang wird hier gar nicht gebraucht
- für die korrekte Funktion der Software ist eine komplizierte Emulationsschaltung für die nicht benötigten Endschalter nötig (entfällt beim Einsatz der Version 1.2 von MPK3DRV)
- der Treiber arbeitet mit einem Kompromiß zwischen Präzision und Geschwindigkeit, was auf schlechte Programmierung hindeutet
- es ist nur eine Notlösung
- diese Art der Hardware-Ansteuerung ist unter Windows und jedem anderen Multitasking-System veraltet

Zudem setzt die Firma *isel* auf das DOS und erklärte nicht die Absicht, einen echtzeitfähigen Windows-Treiber zu entwickeln.

Aufgrund dieser vielen negativen Aspekte fiel die Entscheidung, die Hardware direkt anzusteuern und einen geeigneten neuen echtzeitfähigen Treiber zu entwickeln. Die erforderlichen Informationen zur Ansteuerung der Mikroschritt-Karte fanden sich im wesentlichen in der *isel*-Dokumentation [isel1]. Der Rest wurde durch Probieren ermittelt.

2.1.5 Vorhandene Ansteuer-Oberfläche

Auf der Basis des Treibers MPK3DRV.EXE und der oben erläuterten Ansteuerung des Ultraschallwandlers wurde ein DOS-Programm mit Borland Pascal entwickelt, welches alle für die Messung benötigten Funktionen unter einer Oberfläche vereinigt [Fri]. Die Positionierung des Motors und das Absenden von Meßimpulsen kann sowohl menügesteuert als auch durch vorprogrammierte Schleifen erfolgen. Die Meßwertaufnahme erfolgt durch Ablesen vom Oszilloskop und Eingabe der Daten. Diese Daten können in ein zum Programmsystem Matlab kompatibles Format umgewandelt werden. Die Weiterverarbeitung der Meßreihen erfolgt im Labor generell durch Matlab, da es sich hierbei um ein leistungsfähiges und auf verschiedenen Plattformen verfügbares Software-Werkzeug handelt.

Das vorhandene Programm erweist sich beim genaueren Hinsehen als undurchschaubar („Spaghetti-Code“), mit Sprüngen quer durch das ganze Programm, sogar zwischen verschiedene Prozeduren. Außerdem gibt es weder Übergabeparameter noch Ergebniswerte bei Funktionen, die Prozedurnamen sind nichtssagend, und auch die Kommentierung läßt zu wünschen übrig. Das Programm besteht aus einer einzigen Hierarchiestufe, d.h. eine Aufteilung z.B. in Hardware-Ansteuerung, Meßwert-Verarbeitung und Oberfläche ist nicht erkennbar.

2.2 Der Luftmeßplatz

Der Luftmeßplatz hat den im Bild 2.3 gezeigten Aufbau.

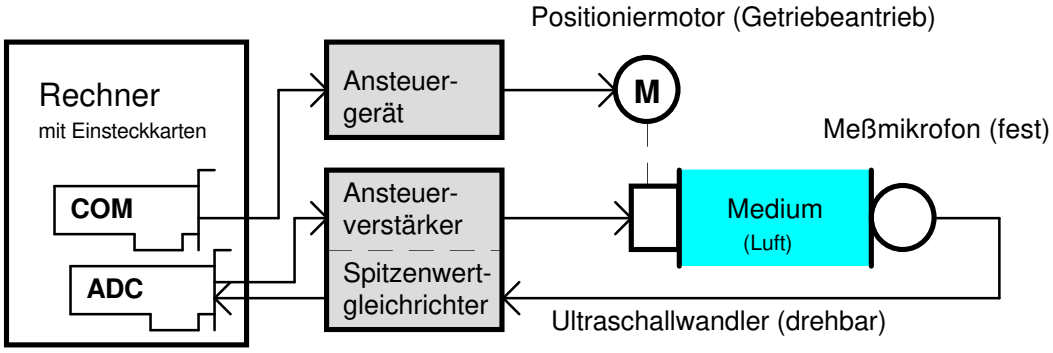


Bild 2.3: Gerätetechnischer Aufbau des Luftmeßplatzes

Aufgrund der ähnlich durchzuführenden Meßaufgaben hat er Ähnlichkeiten mit dem Wassermeßplatz des vorhergehenden Kapitels. Ein Oszilloskop (nicht im Bild) wird nicht unbedingt benötigt, wird aber in der Regel zur Beobachtung des Empfangssignals zusätzlich eingesetzt. Wesentlicher Unterschied ist, daß das Schrittmotor-Ansteuergerät extern ist und keine Echtzeit-Ansteuerung benötigt.

Hierzu sollte ein Programm entwickelt werden, das alle Komponenten für einen manuellen oder automatischen Meßvorgang ansprechen kann.

2.2.1 Positioniermotor

Auch hier wird ein gleichartiger Motor der Firma *isel* eingesetzt. Jedoch wird dieser nicht über eine PC-Einsteckkarte betrieben, sondern von einem externen Ansteuergerät, die über die serielle Schnittstelle programmierbar ist (Bild 2.4). Die Übergabe der Steuerbefehle erfolgt über je eine ASCII-Zeichenkette mit einem Zeilenvorschub-Zeichen (LF) am Ende. Die Quittung erfolgt durch Übertragung eines einzelnen ASCII-Zeichens vom Steuergerät zum Rechner. Das Zeichen '0' bedeutet fehlerfreie Übertragung und Ausführung. Das Datenübertragungsprotokoll (Baudrate, Datenbits, Stopbits) wird am Stuergerät mit einem DIP-Schalterfeld festgelegt. Für den Versuchsstand benötigte Funktionen sind die Referenzfahrt und die Positionierung.

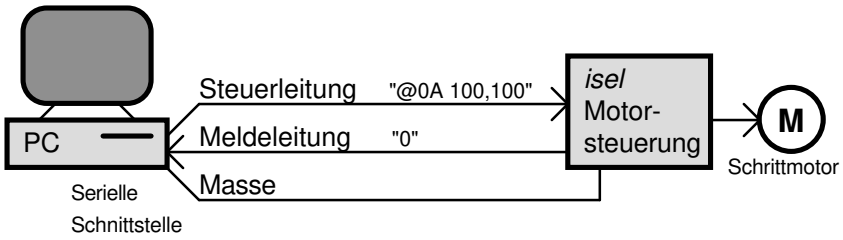


Bild 2.4: Anschluß der Motorsteuerung über eine 3-Draht serielle Schnittstelle

Auch diese Ansteuereinheit mit Motor ist für einen Linearantrieb gedacht. Somit existiert wieder das Problem, daß der Endschalter im Normalbetrieb überfahren werden muß, wenn Winkel ≥ 360° abgefahren werden. Möglich ist einerseits, daß der Anwender von Hand die Funktion dieses Kontakts ein- und ausschaltet, oder daß ein freies Ausgabeport der ohnehin erforderlichen Analog-Digital-Wandlerkarte im Rechner den Vorgang automatisiert. Andererseits ist es auch möglich, eine programmierbare Ausgabeleitung der seriellen Schnittstelle zum Schalten zu verwenden. Das vermeidet die sonst erforderliche Querverbindung und wurde im Rahmen dieser Diplomarbeit so gelöst.

Die externe Ansteuereinheit erspart die programmtechnischen Probleme, die mit der PC-Einsteckkarte vom Wassermeßplatz (siehe Abschnitt Dabei soll die Softwarelösung in der Lage sein, die Meßdaten zum Softwarepaket Matlab zu transferieren. Unter DOS benötigt man ein Programm, das alle drei Komponenten des Meßplatzes (Positionierung des Motors, Aussenden von Impulsen, Aufnahme von Meßwerten) vereinigt und die Meßwerte in eine von Matlab lesbare Datei schreibt. Die mächtige Skriptsprache von Matlab und das DDE von Windows können unter DOS nicht genutzt werden. Daher steht das Ziel, die Meßwertaufnahme und die zugehörigen Prozesse gänzlich nach Windows zu verlagern.

2.1.1) auftreten. Es ist kein Echtzeitverhalten erforderlich. Da für die Rückmeldung nur 1 Zeichen übertragen wird, ist es nicht notwendig, die Zeichen von der seriellen Schnittstelle durch eine ISR (Interrupt-Behandlungsprogramm) abzuholen; eine einfache Abfrage über die entsprechende BIOS-Funktion genügt, da der im Rechner eingebaute Schnittstellenschaltkreis mindestens 1 Zeichen puffern kann.

Diese externe Lösung bietet noch weitere Vorteile. Beispielsweise ist die serielle Schnittstelle an allen gängigen Computertypen vorhanden, und entsprechende Treiber dafür sind generell im System integriert. Die mit energiereichen Impulsen belasteten Leitungen von der Leistungsendstufe zum Motor können kurz gehalten werden, und die Störabstrahlungen sind nicht so groß bzw. besser beherrschbar.

2.2.2 Analog-Digital-Wandlerkarte

Die Meßwertaufnahme erfolgt nicht wie beim Wassermeßplatz mit Hilfe eines computersteuerbaren Oszilloskops, sondern mit einer im Rechner eingebauten Analog-Digital-Wandler-Karte. Dabei kam ein industriell gefertigtes Exemplar, vertrieben durch die Firma *Kolter Electronic*, zum Einsatz.

Die eingesetzte Analog-Digital-Wandler-Karte ist eine universelle Meßkarte für den Laboreinsatz und hat 16 analoge 12-bit-Meßkanäle und 16 TTL-kompatible, frei verwendbare Ein-/Ausgänge (Datenrichtung je nach Programmierung). Technische Daten hierzu befinden sich im Anhang A2 sowie mit Schaltplan in der Dokumentation [AD12].

Die Karte ist über folgende I/O-Adressen ansprechbar:

```
const
 ADC_Base: Word=$DE0;
 ADC_StartConvert=0;  {ADC Umsetzung starten (Dummy-Lesezugriff)}
 ADC_HighByte    =2;  {Portadresse der höherwertigen Bits 11..4}
 ADC_LowByte     =3;  {Portadresse der Bits 3..0 in den Bits 7..4}
 PIA_PortA       =4;  {Kanal-Auswahl und S&H (PIA 8255 Kanal A}
 PIA_PortB       =5;  {Freies Ausgabe-Register (PIA 8255 Kanal B)}
 PIA_PortC       =6;  {Freies Ausgabe-Register (PIA 8255 Kanal C)}
 PIA_Control     =7;  {PIA 8255 Steuerbyte}
```

Mit dem Kanal-Auswahl-Register wird einer der 16 Analogeingänge ausgewählt. Die Nummer des Kanals liegt binärkodiert in den niederwertigen 4 Bits des Kanal-Auswahl-Registers vor. Das Bit 7 steuert die 4 Sample&Hold-Stufen der Eingänge 0..3. Unverständlicherweise haben die Entwickler nur eine Leitung für alle S&H-Stufen verwendet. Damit entgeht die bei Umsetzern dieser Art übliche Möglichkeit des wechselseitigen Betriebs zweier odere mehrerer Eingänge mit hohen Abtastraten. Die S&H-Stufen kommen am Luftmeßplatz nicht zum Einsatz; die erforderliche Signalkonditionierung wird hier vom Spitzenwertgleichrichter wahrgenommen. Die übrigen 3 Bits sind laut Schaltplan nicht belegt.

Ausgelöst wird ein Umsetzvorgang durch einen lesenden Portzugriff auf das Steuerport, wobei der eingelesene Wert ohne Bedeutung ist und verworfen werden kann. Danach wird eine bestimmte feste Zeit, die sich aus der Umsetzzeit des Wandlers ergibt, gewartet. Die Wartezeit liegt im µs-Bereich und variiert je nach eingesetztem Schaltkreis (hier: 25µs). Der Fertigmelde-Ausgang des verwendeten Wandler-Schaltkreises ist leider nicht softwaremäßig auswertbar, weder durch Polling noch durch Interrupt. Somit bleibt einem Programm nur die Möglichkeit, eine genügend lange Zeit zu warten. Ein Griff zum Lötkolben ist angebracht, um die Karte sinnvoller und einfacher einsetzbar zu machen. Wenn das tatsächliche Ende des Umsetzvorgangs softwaremäßig erkennbar ist, kann durch diese Maßnahme die Umsetzzeit minimiert werden. Die auf der Karte vorhandene Interruptmöglichkeit ist rudimentär und besteht letztlich aus einer herausgeführten Interruptleitung.

Nach Verstreichen der Wartezeit kann der digitalisierte Meßwert ausgelesen werden. Die Reihenfolge des Auslesens (erst Low, dann High oder umgekehrt) ist nicht von Belang.

Ein Auszug aus dem Programmteil ADCMESS.PAS zeigt, wie ein Meßwert aufgenommen werden kann:

```
function GetAdcValue(Kanal: Byte):Integer;
 begin
  Port[ADC_Base+PIA_PortA]:=Kanal;       {meist Kanal 7}
  Kanal:=Port[ADC_Base+ADC_StartConvert];{Kanal als Dummy mißbrauchen}
  ShortDelay(25000 div TICKNANO);        {Zeitverzögerung 25µs}
  GetAdcValue:=Word(Port[ADC_HighByte]) shl 4 + Port[ADC_LowByte] shr 4;
 end;
```

Der Analog-Digital-Umsetzer kann mittels Jumper (Kodierstecker) auswählbar sowohl vorzeichenbehaftet als auch vorzeichenlos betrieben werden. Am Meßplatz wurde die vorzeichenlose Jumperung vorgefunden. Das Ergebnis liegt als Binärzahl mit 12 Bit vor. Das High-Byte enthält die oberen (höherwertigen) 8 Bits des Meßergebnisses, das Low-Byte enthält in der höherwertigen Tetrade die unteren (niederwertigen) 4 Bits. Die niederwertige Tetrade des Low-Bytes ist undefiniert und dient wahrscheinlich als Platzhalter für eine höherauflösende Version derselben Karte bzw. Wandlerschaltkreises. (Im Gegensatz zu Bitbreiten im Rechner, die „nach oben“ Richtung MSB wachsen, wachsen Bitbreiten von Umsetzerschaltkreisen gewöhnlich „nach unten“ in Richtung LSB.)

Man beachte, daß die voreingestellten Ein-/Ausgabeadressen das Limit für den PC/XT von 3FFh übersteigt. In älteren Bussystemen kann sich daher eine Mehrdeutigkeit herausstellen, mit der Folge, daß die Karte ebenfalls auf den I/O-Adressen 1E0h..1E7h „sichtbar“ ist. Diese Adressen ergeben sich aus der bitweisen Verknüpfung `$DE0 and $3FF .. $DE7 and $3FF` (hier in Pascal-Schreibweise). Da auch dieser untere I/O-Bereich von keiner Standard-Ein-/Ausgabeeinheit benutzt wird, sollten sich keine Probleme daraus ergeben. Umgekehrt ist zu beachten, daß weitere Hardware, die nur 10 Adreßleitungen ausdekodiert und sich auf Portadresse 1E0h befindet, ebenfalls mit dieser Meßkarte adreßraummäßig kollidiert.

Von den 16 frei programmierbaren TTL-kompatiblen Anschlüssen sind 8 nach außen geführt. Von denen wurden bisher 2 Leitungen verwendet:

```
{Bit 0 von PIA_PortB: Spitzenwertgleichrichter-C aufladen/löschen}
const
 SWG_Loesch=0;
 SWG_Laden=1 shl 0;
{Bit 1 von PIA_PortB: Sendeleitung ein/aus}
 SL_ein=1 shl 1;
 SL_aus=0;
```

Aufgrund der relativ geringen erforderlichen Meß-Rate im Vergleich zur Umsetzgeschwindigkeit ist die Verwendung von Interrupts nicht besonders vorteilhaft gegenüber Polling, selbst wenn die Karte interruptfähig wäre. Über das Zeitverhalten der Messung klärt der folgende Abschnitt auf.

2.2.3 Der Spitzenwertgleichrichter und der zeitliche Ablauf der Ultraschallmessung

Für den Versuchsaufbau wurde im Vorfeld eine Sendeendstufe gebaut, die eine Flanke des Sendesignals in einen Hochspannungsimpuls definierter Spannung und Energie umwandelt, mit dem der Sendewandler direkt angesteuert werden kann [Hart]. Da die Meßimpulse nur sehr kurze Zeit verfügbar und Wechselsignale mit einer Zeitdauer im unteren µs-Bereich sind, ist außerdem ein Spitzenwertgleichrichter zur Erfassung der reflektierten Welle erforderlich. Diese beiden Komponenten wurden daher zusammen mit dem Netzteil in ein gemeinsames Gehäuse gebaut.

Die Technischen Daten dieses Gerätes befinden sich im Anhang A1.

Um Meßfehler durch Störungen möglichst auszuschließen, werden im Originalprogramm 2 Wege gegangen:

Erstens wird von einer mehrfachen Versuchsdurchführung („Meß-Salve“) ein Mittelwert gebildet. Pro Stellung des Motors werden z.B. 100 „Schuß“ abgegeben und die Abstrahlleistung gemessen. Hauptfehlerquelle sind elektrische Störungen, insbesondere durch die 20-kHz-Steuerfrequenz für den Schrittmotor.

Zweitens wird, um Störungen durch Umgebungslärm (z.B. an Laborgegenständen reflektierte Wellen oder einer zweiten Ultraschall-Meßanlage) auszuschließen, nur innerhalb eines „Zeit-Fensters“ gemessen. Aufgrund der Bekanntheit der Entfernung zwischen Wandler und Meßmikrofon sowie der Schallgeschwindigkeit in Luft wird der Spitzenwertgleichrichter erst kurz vor dem Eintreffen der Welle geöffnet, und unmittelbar danach der Meßwert in den Rechner übernommen. Danach kann in einer verhältnismäßig großen Zeitspanne der Meßwert von der oben genannten Analog-Digital-Wandlerkarte umgesetzt und rechentechnisch verarbeitet werden. Störungen durch vor- oder nacheilende Wellenfronten werden so wirksam bekämpft. Bild 2.5 verdeutlicht diesen Vorgang grafisch.

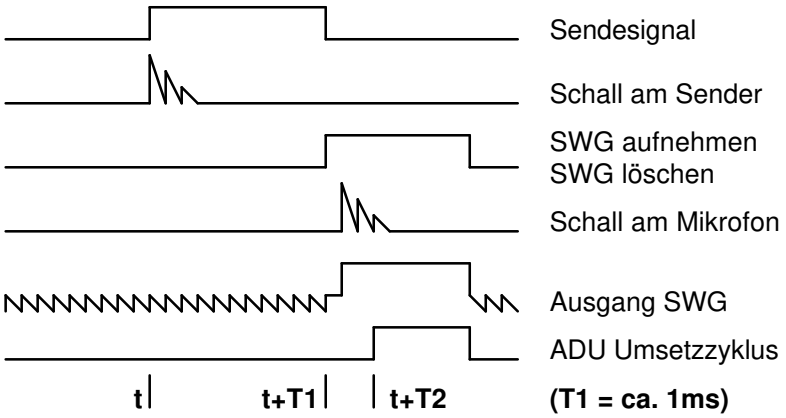


Bild 2.5: Zeitlicher Ablauf der Ultraschallmessung

Zur einfachen Kontrolle der Softwarefunktion mit dem Oszilloskop wird in dem neu zu erstellenden Programm während der Zeit der Umsetzung das digitale Ausgabebit B6 aktiviert.

Die beste Art der Störungsverminderung ist, die Störquellen selbst auszuschalten. Die starken elektrischen Impulse von der Motorsteuerung lassen sich wirksam bekämpfen, wenn die Motortreiberstufen für die Zeit der Messung abgeschaltet werden, da die Achse ohnehin stehen muß. Dadurch verliert jedoch der Schrittmotor sein Haltemoment und könnte von seiner eingenommenen Lage abweichen, was sich nicht ohne zusätzliche Mittel erfassen läßt. Da hier am Motor ein Getriebe mit Untersetzungsverhältnis 50:1 angeflanscht ist, ergibt sich im Gegensatz zum Wassermeßplatz die Situation, daß eine kräftige Selbsthemmung vorhanden ist. Wie Versuche zeigten, genügt sie, um die Achse auch unter Verwendung einer Vorspanneinrichtung zum Eliminieren des Getriebespiels sicher in Position zu halten.

Die Steuerkarte weist auf der Rückseite einen Low-aktiven TTL-Eingang zur Abschaltung der Endstufen auf. Zu seiner Aktivierung mußte noch eine Lötbrücke gesetzt werden. Angesteuert wird dieser einfach von einer weiteren der beiden vorhandenen Steuerleitungen der seriellen Schnittstelle. Somit sind nunmehr zwischen Rechner und Motor im Gegensatz zu Bild 2.4 noch 2 weitere Leitungen erforderlich.

2.2.4 Vorhandene Software

Ein Komplett-Ansteuerprogramm für den Luftmeßplatz liegt als ein Ergebnis einer Studienarbeit vor [Dietz]. Es arbeitet unter DOS mit einer Menüsteuerung. Geschrieben wurde das Programm in Borland C unter Verwendung des Speichermodells Small (bedeutet: ein Code- und ein Datensegment). Die Quellen sind bis auf "`video.c`", die Quelldatei für die Menüführung, alle vorhanden.

Das Programm wurde so geschrieben, daß es auf dem XT (8086-Prozessor mit 4,77 MHz Taktfrequenz) läuft. Wegen der Notwendigkeit sehr kurzer Zeiten wurden feste Schleifen eingebaut, womit das Programm an diesen Rechner gebunden ist. Aufgrund der geringen Taktfrequenz des Prozessors ist die Einhaltung des Ablaufschemas mit maximalen Abweichungen im Mikrosekundenbereich kaum anders möglich.

Bedingt durch das verwendete Speichermodell ist nur die lokale Speicherung von Datensätzen bis maximal 64 KB möglich. Eine nachträglich durchgeführte Modifikation, die Datenbereiche mittels `farmalloc()` anzufordern, um mehr Daten halten zu können, bringt jedoch eine Reihe weiterer Probleme mit sich, die es erforderlich machen würden, das gesamte Programm gründlich zu überarbeiten. Die nötige Festlegung auf ein Speichermodell in C bringt Probleme, wenn es bei der Weiterentwicklung des Programms nicht mehr für die Datenhaltung ausreicht. Das kann in Pascal nicht passieren, da Pascal von vornherein ein „großes" und auf die Besonderheiten des 8086-Systems abgestimmtes Speichermodell benutzt, welches keinem der C-Speichermodelle entspricht.

Das Programm verwendet die Menüsteuerung als eine Art Unterprogrammstarter. Beim Start des Programms wird der Menüsteuerung eine Struktur mit den einzelnen Namen, Programmadressen usw. übergeben. Das entspricht in etwa heutigen modernen Programmierumgebungen wie Turbo Vision oder Windows. Jedoch ist kaum erkennbar, wann welche Routine gerufen wird. Eine Hierarchie des Programms in verschiedene Ebenen (Hardware-Ansteuerung, Meßwertaufnahme, Menü-Dialoge und Grafikausgabe) ist erkennbar, jedoch nicht durchgehend. Analoges betrifft auch die Parameterübergabe an und von Funktionen. Innerhalb der Funktionsrümpfe wurde bisweilen erheblicher Aufwand zur Konvertierung einer Zahl in einen String und umgekehrt eingesetzt. Offenbar waren die effizienten Funktionen `atoi()`, `itoa()` sowie das leistungsstarke `sprintf()` mit den möglichen Formatanweisungen unbekannt.

Auch dieses recht komplexe Programm wurde im Rahmen dieser Arbeit über Bord geworfen und von Grund auf neu geschrieben.

2.3 Einführung zu VxD's

Windows als moderne Programmierumgebung erscheint durch seine Multitaskingfähigkeit geeignet, komplexe Aufgaben zu lösen. Häufig erweist es sich als notwendig, externe Hardware anzusprechen. Die Möglichkeit, über Dynamischen Datenaustausch (DDE) Daten zwischen Applikationen auszutauschen, macht es attraktiv, für jede Hardware einen speziellen Treiber als DDE-Server zu schreiben und die Datenverarbeitung von Standardsoftware (Tabellenkalkulation, Datenbank, CAD-Programm) durchführen zu lassen. Nahezu jede Standardsoftware unterstützt die DDE-Konversation.

Externe Hardware anzusteuern, die an einer Standardschnittstelle (COMx, LPTx) angeschlossen ist, ist problemlos über die COMM-Funktionen von Windows möglich. Auch das Ansteuern von Hardware auf einer Steckkarte mit Ein-/Ausgabeadressen (I/O-Ports) ist zumindest unter Windows 3.x problemlos, solange die Zugriffe nicht zeitkritisch sind. Der gängige Weg läuft über eine Dynamische Linkbibliothek (DLL), um zu garantieren, daß nur ein Programm auf das I/O-Port zugreifen kann. Eine DLL wird stets nur einmal geladen. Alternativ kann man das Anwendungsprogramm so schreiben, daß es sich nur in einer Instanz starten läßt.

Der nächste Schritt mit zunehmendem Schwierigkeitsgrad ist die Verarbeitung von Interrupts. Die Programmierung kann dank DPMI (DOS-Schnittstelle für Protected-Mode-Programme) und dem Virtuellen Gerätetreiber zur Emulation des Interrupt-Controllers (VPICD, dazu später mehr) genauso wie unter DOS durch „Direktzugriff" auf den Interruptcontroller und durch Setzen des Interruptvektors über die gewohnte DOS-Schnittstelle (Int 21h) erfolgen.

Des weiteren gibt es Geräte mit Speicher-Ein-/Ausgabe (memory mapped I/O). Da der Speicherbereich generell auf festen Adressen liegt und unter Windows Standard-Aliasdeskriptoren (vordefinierte Deskriptoren mit gleichen Adreß- und verschiedenen Zugriffs-Angaben) zur Verfügung stehen, gestaltet sich der Zugriff weitestgehend einfach.

Es gibt jedoch eine ganze Reihe Probleme, die unter Windows nicht oder nur umständlich durch .EXE- und .DLL-Programme gelöst werden können. Hier setzt der Bereich der VxD's ein.

VxD's galten und gelten als ein Thema "for wizards only" – nur für Zauberer. Die folgenden Ausführungen sollen die Geheimnisse um diese Programme lüften helfen.

2.3.1 Was sind VxD's?

VxD's sind Virtuelle Gerätetreiber, die beim Laden Teil des eigentlichen Windows-Kernels werden und mit Windows/386™ erstmalig eingeführt wurden. Die Namensgebung der Abkürzung VxD ergibt sich aus "Virtual *irgendein* Device", wobei das "x" für „*irgendein*" (something) steht. Entweder sind sie Anhängsel der Datei WIN386.EXE (diese Datei stellt u.a. eine Kollektion aus Standard-VxD's dar), oder sie werden als Einzeldateien mit der typischen Endung .386 zu verschiedenen Anwendungen mitgeliefert.

VxD's sind nur unter Windows/386 (eine Windows-Variante der Version 2.x), Windows Enhanced Mode (Windows 3.x) und Windows 95 aktiv. Sie sind nicht aktiv (werden nicht geladen) beim Betrieb von Windows im Real- oder Standard-Mode sowie bei WinOS2 (Windows unter OS/2) sowie nicht verwendbar unter Windows NT. Eine Unterstützung ist durch OS/2 4.0 vorgesehen.

War man es als Programmierer unter DOS gewöhnt, alles machen zu dürfen und von der gesamten Prozessorleistung Besitz zu ergreifen, so wird man unter Windows unter Verwendung „normaler" Programme (.EXE oder .DLL) mit den Möglichkeiten schon recht eingeengt. Sollte jemandem Windows als zu enges Korsett erscheinen, sodaß man bisher immer bei DOS blieb, ändert sich diese Sichtweise beim Einstieg in die VxD-Programmierung. Alles ist wie schon unter DOS erlaubt (mehr noch, wenn man sich die Beschränkungen unter EMM386 vor Augen hält), man bekommt einen leistungsfähigen, dokumentierten 32-bit-DOS-Extender als Experimentierplattform, hat die volle Konrolle über DOS- und Windows-Programme gleichermaßen und kann Dinge tun, die unter DOS undenkbar sind. Auch ganze Programme lassen sich als VxD schreiben, ein Beispiel dafür ist *NuMega*'s SoftICE-Debugger.

Über die VxD-Programmierung sind bisher weltweit 3 Bücher erschienen [Dav] [Haz] [Nor]. Sie sind in englischer Sprache verfaßt, behandeln keine Windows 95-Spezifika und sind verhältnismäßig schwer zu bekommen. Wenigstens eines davon zu lesen ist angeraten.

Weitere Beiträge zum Thema VxD befinden sich im Microsoft System Journal, welches es mit der Mitgliedschaft im MSDN gibt, sowie in den dort erhältlichen Dokumentationen auf CD. Diese Informationsquellen sind sehr kostenintensiv.

2.3.2 Erstellung und Programmierung

Pro und Kontra des Einsatzes von VxD's

Zu Beginn des Abschnitts 2.3 wurden typische Einsatzfälle „klassischer" Windows-Treiber-Programmierung beschrieben. VxD's sind vorzuziehen, wenn auch noch DMA (Direkter Speicherzugriff) ins Spiel kommt oder höherfrequente Interrupts (10..>1000Hz) zu verarbeiten sind. Ist Kompatibilität zum Standard- oder gar Real-Modus wichtig, können diese Problembereiche auch in DLL's gelöst werden. VxD's sind jedoch *unumgänglich* oder zumindestens sehr empfohlen in folgenden Fällen:

- wenn die volle Kontrolle über den Prozessor erforderlich ist, um z.B. Echtzeitverhalten im Mikrosekunden-Bereich zu erhalten (empfohlen, z.T. erforderlich bei zeitkritschen Zugriffen auf geschützte Ports)
- wenn die Interruptfrequenz 1 kHz übersteigt (empfohlen)
- wenn Hardware zu emulieren ist (erforderlich)
- wenn Gerätekonkurrenz zu behandeln ist (erforderlich)
- wenn Hardware vor ungewollten und gefährlichen Fremdzugriffen zu schützen ist, um das System stabiler zu machen (erforderlich)
- wenn man sehr viel Prozessorleistung benötigt

Anhand dieser Punkte kann man ersehen, wie mächtig VxD's in das Gesamtsystem eingreifen können. Genauso fatal kann sich aber auch der kleinste Fehler auf das Gesamtsystem auswirken, denn es gibt keine Schutzmechanismen innerhalb des Kernels und der darin eingebundenen VxD's. Bei einem zur Ausnahmebedingung (Exception) führenden Fehler innerhalb des Kernels beendet sich Windows bzw. die betreffende DOS-Box umgehend. Insofern ist nicht alles erlaubt, da beispielsweise unbekannte Befehle oder Divisionen durch Null genauso wie im Real Mode zum Abbruch führen (wobei es jedoch wiederum möglich ist, neue Befehle zu emulieren und die Division durch Null gezielt abzufangen). Neu hinzu kommen Seitenfehler (Zugriff auf weder im Speicher noch auf Festplatte vorhandene Speicherseiten), die ebenfalls zum Abbruch führen.

Erstellungsmöglichkeiten

Das Schreiben von VxD's erfolgt im allgemeinen in Assemblersprache. Erforderlich ist dazu das DDK (Device Driver Kit) von Microsoft, welches über das MSDN (Microsoft Developer Network) Level 2 erhältlich ist. Alternativ läßt sich DDK-Lite aus [Dav] in Verbindung mit einem Microsoft Assembler (MASM) einsetzen. Neuerdings setzt sich auch C mehr und mehr durch, um auch komplexere Funktionen (z.B. mathematische Berechnungen) innerhalb des VxD's ohne großen Aufwand durchführen zu können. Dabei kann VtoolsD von *Vireo Software* zur VxD-Entwicklung herangezogen werden, welches durch eine alle Assembler-Funktionen „einhüllende“ Bibliothek die Treiberentwicklung in C vereinfacht. Von den o.g. Möglichkeiten stand nur das DDK-Lite zur Verfügung, welches in Ermangelung eines MASM auf den vorhandenen TASM (Turbo Assembler 3.2 von Borland, mitgeliefert zu Borland Pascal 7.0) zugeschnitten wurde.

Das Debuggen kann mittels der Debug-Version des Windows-Kernels und dem Programm WDEBUG.EXE erfolgen, die auch dem DDK-Lite beiliegen. Erforderlich ist hierzu ein Terminal, das an eine serielle Schnittstelle des Rechners angesteckt wird. Alternativ kann auch ein zweiter (Monochrom-) Monitor am gleichen Rechner eingesetzt werden. Besonders komfortabel ist das Debuggen des VxD's mit SoftICE (Software-In-Circuit-Emulator) für Windows von *NuMega*, obwohl auch dieses Programm im wesentlichen per Kommandozeile zu bedienen ist.

Was schließlich für die Entwicklung der VxD's eingesetzt wurde, sind TASM.EXE, LINK386.EXE, ADDHDR.EXE und diverse Include-Dateien, allen voran die VMM.INC. Nützlich waren noch die Beispieldateien sowie das Buch [Dav] selbst.

Die nun folgende Einführung reißt in sehr kurzer Form verschiedene Aspekte der VxD-Programmierung an. Danach erfolgt ein Vergleich mit Gerätetreibern anderer moderner Betriebssysteme.

2.3.3 Grundlegende Arbeitsweise von VxD's im Kernel

Der VxD-Ladevorgang

Virtuelle Gerätetreiber werden nur beim Start von Windows geladen und mit dem Kern verbunden. Die Einbindung von VxD's erfolgt durch Eintrag in die SYSTEM.INI unter der Sektion [386Enh] mit der Zeile in der Form „`device=avxd.386`“. Die Reihenfolge spielt – im Gegensatz z.B. zu Gerätetreibern in der CONFIG.SYS – keine Rolle, da die VxD's eine Lade-Reihenfolge-Nummer besitzen, an der sich der VxD-Lader des Kernels orientiert. Die Groß- und Kleinschreibung ist unerheblich. Befindet sich der Gerätetreiber nicht im Standard-Suchpfad von Windows, muß ein Pfad angegeben werden. Der Standard-Suchpfad von Windows ist folgender:

1. Das aktuelle Verzeichnis,
2. das Windows-Verzeichnis (das Verzeichnis, das WIN.COM enthält); die Funktion `GetWindowsDirectory()` ermittelt den Pfadnamen dieses Verzeichnisses,
3. das Windows-Systemverzeichnis (das Verzeichnis, das Systemdateien wie GDI.EXE enthält); die Funktion `GetSystemDirectory()` ermittelt den Pfadnamen des Verzeichnisses,
4. das Verzeichnis, das die ausführbare Datei für die aktuelle Task enthält; die Funktion `GetModuleFileName()` liefert den Pfad dieses Verzeichnisses (entfällt für VxD's),
5. die Verzeichnisse, die in der Umgebungsvariablen PATH aufgeführt sind,
6. Die Verzeichnisse, die in einem Netzwerk (mit dem Befehl MAP) aufgeführt sind [BPOLH].

In der Datei WIN386.EXE enthaltene VxD's werden mit einem Sternchen („*“) vor dem Gerätetreibernamen kenntlich gemacht. Eine andere – weniger offensichtliche – Art der Einbindung Virtueller Gerätetreiber erfolgt beim Start von Windows durch geladene residente DOS-Programme über die Anzapfung des Int 2Fh. Davon machen beispielsweise DOS-basierte Netzwerkserver Gebrauch.

Alle VxD's werden beim Start von Windows geladen und mit dem Kern verbunden. Der Windows-Kern ist insofern modular aufgebaut, da ein VxD, die Standard-VxD's eingeschlossen, einen relativ kleinen Kern (VMM und den VxD-Lader) in seiner Funkionalität erweitern. Es liegt dasselbe Prinzip zugrunde, wie auch Windows selbst im wesentlichen ein Bündel aus .DLL's ist, die die Grundfunkionalität erweitern. Insofern ist der Windows-Kernel kein monolithischer wie der einfacher Unix-Systeme, noch kann man von Mikrokernel-Architektur sprechen – hierzu fehlt die Möglichkeit, Treiber zur Laufzeit von Windows zu laden und abzustoßen.

Der Kern selbst arbeitet im sog. FLAT-Speichermodell (flat = flach, eben). Das entspricht in etwa dem TINY-Modell (tiny = winzig), jedoch mit 32-bit-Offsets. Die Segmentregister sind fest und haben die Basisadresse 0 und ein Limit von FFFFFFFFh. Es gilt DS=ES=SS<>CS. Der Codesegment-Selektor zeigt auf einen Codesegment-Deskriptor (im Windows-Kernel hat der Selektor den konstanten Wert 28h, dies kann sich jedoch in künftigen Versionen von Windows ändern), die anderen Selektoren auf einen Datensegment-Deskriptor (im Windows-Kernel 30h). Der Kernel und die VxD's werden auf lineare Adressen >80000000h gelegt bzw. geladen (Bild 2.6). Die Paging-Einheit ist generell aktiviert (Bit31 in CR0 =1), um diese Adreßlage zu garantieren. Alle Virtuelle Maschinen (VM's) liegen dahinter. Somit ergibt sich ein Limit von knapp 2 Gigabyte für die Summe aller Anwendungen (inklusive ausgelagertem Speicher).

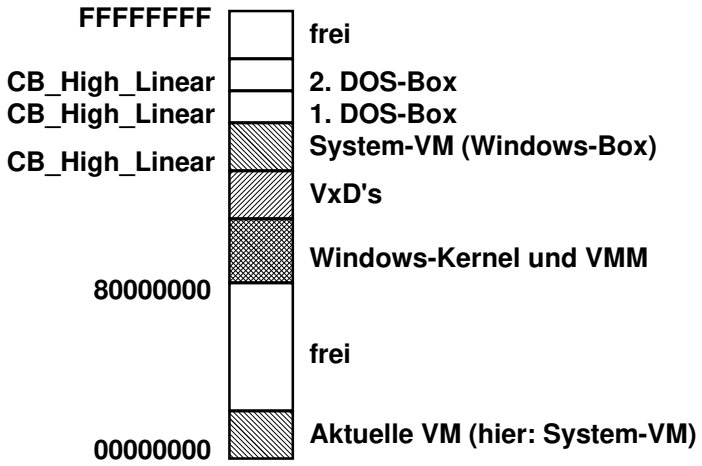


Bild 2.6: Speicherbelegung im linearen Adreßraum unter Windows (2 DOS-Boxen geöffnet)

Die jeweils aktive VM wird zusätzlich in den Bereich 0..7FFFFFFFh gemappt. Dies ist erforderlich, da nur so mehrere V86-VM's realisiert werden können. Auch die System-VM hat einen Real-Mode-Rumpf, der im wesentlichen aus dem Ergebnis des System-Schnappschusses und dem DPMI-Host besteht. Die Umschaltung erfolgt durch das Umladen des Registers CR3 auf eine andere Page-Tabelle bei der Taskumschaltung des Prozessors (CR3 ist im TSS enthalten).

Unter Windows 3.x beinhaltet jede VM genau eine Task, d.h., die Virtuellen Maschinen sind “single-threaded”, haben in sich genau einen Ausführungspfad. Eine VM ist hierbei das ganze Windows an sich (die sog. System-VM) sowie jede einzelne DOS-Box. Bei Windows für Workgroups wird noch pro Server-Prozeß eine VM geöffnet.

Der System-Schnappschuß (system snapshot)

Es ist auffallend, daß sämtliche TSR-Programme in jeder DOS-Box zu sehen sind, die *vor* dem Starten von Windows bereits im Speicher waren. Lädt man jedoch ein weiteres TSR-Programm in einer von 2 DOS-Boxen, ist das TSR für die andere Box vollkommen unsichtbar. Das TSR verschwindet zudem automatisch (jedoch nicht immer 100%ig rückwirkungsfrei für das System!), wenn die DOS-Box – gewaltsam oder nicht – geschlossen wird. Was weniger bekannt ist, ist die Tatsache, daß auch Windows selbst nur eine (salopp gesagt) DOS-Box ist, in der permanent ein Programm mit DPMI-DOS-Extender läuft, in der genauso alle vorher geladenen TSR's sichtbar sind. Die Verhältnisse unter Windows und Borland Pascal 7.0 mit DPMI-Zielplattform sind daher sehr ähnlich.

Verantwortlich für dieses Verhalten ist der System-Schnappschuß, ein fundamentales Ereignis beim Start von Windows. Der System-Schnappschuß erfolgt, nachdem sämtliche VxD's ihre Initialisierungsprozedur hinter sich haben (dazu später mehr). Der momentane Stand der DOS-Speicherbelegung sowie wichtiger I/O-Bausteine wie Interrupt- und DMA-Controller wird festgehalten als Grundlage für die Geburt aller VM's. Danach wird die System-VM im DOS-Modus (V86-Mode) hochgefahren. COMMAND.COM wird gestartet, und diese führt die Datei WINSTART.BAT (falls vorhanden) aus. TSR-Programme, die hier geladen werden, sind in anderen DOS-Boxen nicht sichtbar. Das schafft in den DOS-Boxen mehr Platz, falls ein TSR nur für Windows selbst gebraucht wird. Danach beginnt das eigentliche Windows mit dem Start von KRNL386.EXE.

Der Windows-Kernel (und das gilt prinzipiell für alle Kernel moderner Betriebssysteme) arbeitet in einer Art Kooperativen Multithreading. Ein Thread („Faden“) ist ein Ausführungspfad im gemeinsamen Ressourcenraum. Jede Task (im VxD-Sinne) bildet sich als genau ein Thread im Kernel ab. Dahinter verbirgt sich ein extra Stackbereich, der jeder Task im Ring 0 zugeordnet ist. Kooperativ bedeutet, daß die Hergabe des Prozessors auf freiwilliger Basis erfolgt. Daher dürfen langwierige Berechnungen nicht ununterbrochen im Kernel erfolgen, da dies das gesamte System lähmt. Der Speicherbereich ist für alle Threads derselbe – unter Windows gilt das für solchen über 80000000h. Einzelne Threads (nach außen: VM's) können auch eingeschläfert und aufgeweckt werden – davon wird unter Windows im Gegensatz zu Unix sehr selten Gebrauch gemacht. Im Prinzip ist Kooperatives Multithreading recht einfach realisierbar, wie es das Beispiel THREADS.PAS zeigt, jedoch stecken die Probleme in Detailfragen. Ebenfalls schwierig für den Neueinsteiger ist das „Denken in Threads“ sowie das Debugging, wenn kein geeigneter Debugger zur Verfügung steht und die ungewöhnlichen Funktionsweisen von Threads nicht klar sind.

Verhalten der VxD's im Kernel und Dynamisches Binden

Ein VxD ist im “Linear Executeable”-Format (Kennzeichen `"LE"` im Header) abgespeichert. Dieses Dateiformat ist ähnlich dem unter Windows verwendeten “New Executeable”-Format (Kennzeichen `"NE"` im Header) in Blöcken aufgebaut, ist aber speziell für 32-bit-Programme ausgelegt. Eine Variante von `"LE"` mit dem Kennzeichen `"LX"` wird unter OS/2 für 32-bit-Anwendungsprogramme eingesetzt [RBIL].

Wesentlicher Unterschied zwischen beiden Formaten ist, daß bei 32-bit-Programmen jeder verwendete Offset relozierbar sein muß; bei 16-bit-Programmen müssen nur Segmentreferenzen angeglichen werden. Häufig übersteigt die Größe der Relokationstabellen die des eigentlichen Codes. Obwohl 32-bit-Programme auch in einem Stück geladen werden könnten (wie das unter Unix der Fall ist), werden innerhalb der LE-Datei Blöcke angelegt, die verschieden verwerflichen Code (Initialisierungs-Code, gesperrter und auslagerbarer Code) enthalten und beim Laden des VxD's entsprechend im kostbaren Kernel-Speicher verteilt werden.

Der VxD-Lader orientiert sich an der einzigen von der LE-Datei exportierten Struktur (Pascal: Record), dessen Name *avxd*_DDB lautet, wobei *avxd* für den Namen des VxD's steht. In dieser Struktur sind u.a. folgende Angaben zusammengefaßt:

- Name des VxD's (8 Zeichen) und Versionsnummer (2 Bytes)
- die VxD-ID (16 bit, eine weltweit eindeutige VxD-Kennung, s.u.)
- Initialisierungs-Reihenfolge-Wert (je größer, desto später wird das VxD angesprungen)
- Eintrittspunkt für die Steuer-Prozedur
- Eintrittspunkte für die V86- und Protected-Mode-API
- Anfangsadresse und Länge einer Tabelle exportierter VxD-Funktionen

Die wichtigste Adresse ist der Eintrittspunkt für die Steuer-Prozedur. Sie wird bei jedem globalen Ereignis – z.T. mehrfach – gerufen, beispielsweise beim Starten und Beenden von Windows sowie beim Starten und Beenden einer Virtuellen Maschine. Bei jedem Ruf übergibt der Kernel im Register EAX einen Nachrichten-Code über das momentane Ereignis. Die wichtigsten Ereignisse sind in den Tabellen 2.1 und 2.2 aufgelistet. Bei sehr einfachen VxD's genügt es häufig, nur den Start von Windows abzufangen, um z.B. einen Interrupt-Handler zu installieren. Bei umfangreicheren VxD's sieht diese Prozedur eher wie eine Fensterprozedur mit einer langen CASE-Anweisung aus. In VMM.INC vordefinierte Makros helfen, den Schreibaufwand gering zu halten und effizienten Code zu erzeugen.

EAX=	**hex**	**Bedeutung**
Sys_Critical_Init	0	Erster Ruf der VxD-Steuerprozedur. Wird gerufen, bevor Interrupts freigegeben werden. Bestimmte Systemrufe, wie z.B. Allozieren von V86-Pages, sind nur hier erlaubt. Die VMM-Funktionen `Simulate_Int` und `Exec_Int` sind verboten. Eine Rückkehr mit gesetztem Carry-Flag bricht VxD-Ladevorgang ab.
Device_Init	1	Zweiter Ruf. Interrupts sind freigegeben, `Simulate_Int` und `Exec_Int` sind erlaubt. VxD's solten hier den Großteil ihrer Initialisierung vornehmen. Eine Rückkehr mit gesetztem Carry-Flag bricht VxD-Ladevorgang ab.
Init_Complete	2	Dritter und letzter Ruf während der Windows-Initialisierung. Die Freigabe von Initialisierungssegmenten der VxD's und der System-Schnappschuß stehen kurz bevor. VxD's, die V86-Seiten >=A0h suchen, sollten das hier tun. Eine Rückkehr mit gesetztem Carry-Flag bricht VxD-Ladevorgang ab.
System_Exit	5	Erster Ruf beim Beenden von Windows, sowohl beim normalen Ausstieg als auch bei Absturz. Interrupts sind freigegeben, der System-Schnappschuß ist restauriert. `Simulate_Int` und `Exec_Int` sind verboten. Carry bei Rückkehr wird ignoriert.
Sys_Critical_Exit	6	Zweiter und letzter Ruf beim Beenden von Windows. Wie System_Exit, jedoch sind Interrupts gesperrt. Carry bei Rückkehr wird ignoriert.

Tabelle 2.1: Steuerprozedur-Rufe beim Start und Beenden von Windows

EAX=	**hex**	**Bedeutung**
Create_VM	7	Erster Ruf beim Erzeugen einer neuen VM; EBX enthält das neue VM-Handle. Eine Rückkehr mit gesetztem Carry-Flag bricht VM-Kreation ab.
VM_Critical_Init	8	Zweiter Ruf beim Erzeugen einer neuen VM. `Simulate_Int` und `Exec_Int` in diese VM sind verboten. Eine Rückkehr mit gesetztem Carry-Flag überführt die VM zum `Not_Executeable`-Status (die VM wird zum Zombie), danach wird sie zerstört.
VM_Init	9	Dritter und letzter Ruf beim Erzeugen einer neuen VM (erfolgt nicht bei der System-VM). `Simulate_Int` und `Exec_Int` in diese VM sind erlaubt. Carry-Behandlung wie bei VM_Critical_Init.
Sys_VM_Init	3	Wie VM_Init, jedoch für System_VM.
VM_Terminate	A	Erster Ruf beim Zerstören einer VM. Carry wird ignoriert. `Simulate_Int` und `Exec_Int` in diese VM sind erlaubt.
Sys_VM_ Terminate	4	Wie VM_Terminate, jedoch für die System-VM. Bei Absturz wird dieser Ruf nicht ausgeführt.
VM_Not_ Executeable	B	Zweiter Ruf beim Zerstören einer VM. EDX enthält diverse Flags. Im Falle des Beendens einer laufenden VM (mit CTRL+ALT+DEL) ist dies der erste Ruf. `Simulate_Int` und `Exec_Int` in diese VM sind verboten. Zurückgegebenes Carry wird ignoriert.
Destroy_VM	C	Dritter und letzter Ruf beim Zerstören einer VM. `Simulate_Int` und `Exec_Int` in diese VM sind verboten. Zurückgegebenes Carry wird ignoriert.

Tabelle 2.2: Steuerprozedur-Rufe beim Start und Beenden einer VM

Der Aufruf von Systemfunktion des VMM (Virtual Machine Manager) oder anderer VxD's (die das System erweitern) erfolgt durch einen seltsamen wie einfallsreichen Mechanismus: Da der direkte Aufruf mit CALL nur über Relokationstabellen möglich wäre (wie das bei Windows-.EXE und .DLL realisiert wird), wird hier ein Dyna-Link-Interrupt verwendet (ein simples INT 20h), dem als je 1 WORD (2 Byte lange vorzeichenlose Ganzzahl) die VxD-ID *m* des Ziel-VxD's und die Nummer *n* der VxD-Funktion folgen. Das sind insgesamt 6 Bytes. Der Interrupthandler für den Int 20h wertet nun die nachfolgenden 2 WORDs aus und sucht daraufhin die wirkliche Adresse der Routine (bzw. gibt eine Fehlermeldung aus und beendet Windows bzw. die DOS-Box). Nun wäre das nicht sonderlich elegant, weil langsam. Der Int 20h-Handler ändert nunmehr diese 6 Bytes zu einem indirekten Aufruf um. Er ist fast so schnell wie der direkte Aufruf, ist ebenfalls genau 6 Bytes lang, und hat gegenüber dem direkten Aufruf den Vorteil, daß weiterhin ein Einhängen (Hook) in den Aufruf und damit die Änderung der Funktionalität der aufgerufenen Funktion durch andere VxD's jederzeit möglich ist. Die „Einhängbarkeit“ und Modifizierbarkeit von solchen Systemrufen war bereits unter DOS ein vorteilhaftes Leistungsmerkmal, welches aus Sicherheitsgründen bei Multiusersystemen keine Verbreitung finden konnte. Die Adresse hinter dem Opcode zeigt auf den *n*ten Eintrag der Tabelle der Funktions-Einsprungadressen des VxD's *m*. Durch diese Rück-Änderung wird das VxD nach und nach immer schneller. Außerdem kann man beim Debuggen sehr leicht sehen, welche Code-Abschnitte noch nicht abgearbeitet wurden.

Bleibt noch die Frage nach der Eineindeutigkeit des ID's *m*. Tatsächlich werden sämtliche VxD-ID's zentral bei Microsoft verwaltet und vergeben. Die Anforderung einer solchen ID gestaltet sich sehr einfach per email im Internet an `vxdid@microsoft.com`. Die ID kostet nichts, und man muß zur Zeit auch nicht Mitglied im MSDN sein. Nur das Formular (s. Anhang A3) ist auszufüllen, und in ein bis zwei Wochen hat man einen ID für sein VxD. Falls man keine Funktionen zu anderen VxD's hin exportieren möchte, ist der ID eventuell unnötig (es gibt noch weitere Verwendungen für diesen ID), und man kann die Konstante `Undefined_Device_ID` verwenden.

Arbeitsweise des Kernels und des Schedulers

Der Vitual Machine Manager (VMM), der Kern des Kernels, ist für grundlegende Systemaufgaben, wie Ereignisverteilung, Speicher- und Deskriptortabellenverwaltung zuständig. Er kreiert Virtuelle Maschinen (VM's), läßt sie laufen und beendet sie. Zu Beginn startet VMM die System-VM mit dem eigentlichen Windows. Standard-DLL's kommunizieren mit dem Standard-VxD namens SHELL, welcher das Starten und Beenden von weiteren VM's für DOS-Boxen initiiert.

Der VMM ist nicht reentrant. Er arbeitet mit Ereignislisten. Ereignislisten sind verkettete Listen mit Aufrufadressen für Ereignisbehandler. Ereignisse entstehen durch alle Arten von Ausnahmebedingungen (Exceptions), Interrupts sowie bestimmten VxD-Rufen. Es gibt eine globale Ereignisliste sowie je eine VM-spezifische zu jeder VM. Die Arbeit des VMM besteht darin, daß er solange Ereignisse aus zwei der Listen abarbeitet, bis sie leer sind. Die beiden Listen sind die globale Liste und die der aktiven VM. Erst wenn diese Listen leer sind, schaltet VMM in den Usermodus und läßt das Anwendungsprogramm in der VM laufen.

Eine Taskumschaltung erfolgt nur beim Übergang vom Kernel- in den Usermodus. Dazu ist jeder VM eine primäre Ausführungspriorität zugeordnet. Dabei handelt es sich nicht um die in den .PIF-Dateien einstellbare. Bevor VMM bei vorgefundenen leeren Listen die Steuerung an eine VM abgibt, durchsucht der primäre Scheduler alle VM's, welcher die höchste Priorität hat, und aktiviert genau jene. Diese Priorität kann durch bestimmte VxD-Rufe verändert werden, man bezeichnet dies als “Boost”. Dadurch können VxD's mit Vorrang bestimmen, welche VM als nächstes abgearbeitet wird oder nicht.

Solange eine Ausführungspriorität über allen anderen steht, wird nur Code in jener VM abgearbeitet.

Im VMM gibt es noch einen zweiten, sekundären oder Zeitscheiben-Scheduler. Dieser verändert nun die Ausführungsprioritäten von VM's um verhältnismäßig kleine Werte, verglichen mit denen, die VxD's benutzen. Zur zeitlichen Festlegung dieser Aktivitäten benutzt er die Einstellungen für Vorder- und Hintergrundpriorität der .PIF-Datei zur jeweiligen DOS-Box. Die Vordergrundpriorität findet Anwendung, wenn die VM fokussiert ist, also die Tastatur besitzt, sonst die Hintergrundpriorität. Somit steuert der sekundäre Scheduler mittelbar über die primäre Ausführungspriorität den primären Scheduler, der dann eventuell die Taskumschaltung vornimmt, falls nicht ein VxD einen anderen Ausführungspfad vorschreibt.

Zusätzlich zum Scheduling gibt es einen „kritischen Abschnitt“. Dieser wird benutzt, um nicht reentranzfähigen Code wie z.B. in DOS und BIOS aufzurufen. Eine VM kann den kritischen Abschnitt mittels `Begin_Critical_Section` beanspruchen und mit `End_Critical_Section` freigeben. Nichts besonderes passiert in dieser Klammer, bis eine andere VM, die inzwischen an die Reihe gekommen ist, mit `Begin_Critical_Section` den kritischen Abschnitt beansprucht. Dann erfolgt sofort eine Taskumschaltung zur VM, die momentan den kritischen Abschnitt besitzt, und der Code wird nun abgearbeitet bis zum `End_Critical_Section`. Daraufhin erfolgt wiederum sofort eine Taskumschaltung, damit die zweite VM fortfahren kann. Es kann nur einen Besitzer des kritischen Abschnitts geben.

Weiterhin bietet der VMM Routinen zum Einschläfern und Aufwecken von VM's an. Dies kann sowohl direkt als auch mittels einem Semaphor (Ampelzeichen) erfolgen. Solange ein Semaphor gesetzt ist (verglichen mit einer Verkehrsampel, die auf rot steht), blockiert eine VM beim Aufruf von `Wait_Semaphore` solange, bis eine andere VM oder eine Interruptprozedur mittels `Signal_Semaphore` den Semaphor löscht, die Ampel sozusagen auf grün schaltet.

2.3.4 Erweiterte Möglichkeiten bei der VxD-Programmierung

Aufgrund der Tatsache, daß VxD's für die korrekte Verwaltung von Windows und DOS-Boxen mitverantwortlich sind, ergibt sich eine wesentlich höhere Komplexität der von einem VxD zu lösenden Aufgaben als beispielsweise bei einem Unix-Kerneltreiber in einem monolithischen Kernel. Dazu werden vom VMM bzw. Standard-VxD's eine Reihe Funktionen angeboten. Die Lösung einiger Standardsituationen bei der VxD-Programmierung wird in den folgenden Abschnitten verdeutlicht.

Arbeit mit Speicher

Der VMM benutzt zwei Speicher-Manager. Der V86MMGR (Virtual-8086-Mode Memory Manager) verwaltet solchen für V86-Anwendungen, wie EMS (Expansionsspeicher) und XMS (Erweiterungsspeicher) sowie Pufferübersetzung zwischen Real- und Protected Mode. Der andere Speichermanager, MMGR, bietet Dienste wie Arbeit mit den Deskriptortabellen LDT und GDT, einem globalen Heap für universellen Gebrauch, die Verwaltung von physikalischem und linearem Speicher, Adreßübersetzungen und Seitenanforderungen.

Der 80386-Prozessor benutzt Deskriptortabellen, um die in der 80x86-Prozessorgeneration typischen Segmente in linearen Speicher umzusetzen. Die Segmentübersetzung ist immer aktiv und kann nur dadurch zurückgedrängt werden, indem Segmente definiert werden, die den gesamten Adressierungsumfang des linearen Speichers (4 Gigabyte) abdecken. Die lineare Adresse wird durch die beim 386-Prozessor eingeführte Paging-Einheit in eine physikalische Adresse umgewandelt. Die Seiten haben eine feste Größe von 4 Kilobyte und sind für die Auslagerung auf Massenspeicher geeigneter als Segmente. Die Adreßübersetzung erfolgt mittels zweistufig hierarchischer Seitentabellen, die sich im Hauptspeicher befinden. Die Seitenübersetzung kann deaktiviert werden.

Speicher kann in Seiten mit der Funktion `_PageAllocate` angefordert werden. Durch die Angabe bestimmter Flags kann erreicht werden, daß der zu allokierende Speicher DMA-fähig ist, oder daß eine Seitenfehler-Prozedur installiert wird (Hook), die beim ungültigen Zugriffsversuch aufgerufen wird. Auf diese Weise kann Speicher simuliert werden, der in dieser Form nicht vorhanden ist. Davon machen Bildschirmtreiber Gebrauch, die die ansonsten recht umständliche programmtechnische Handhabung der VGA-Karte vereinfachen. Über die Allozierung von V86-Seiten mit „Haken“ (`_Assign_Device_V86_Pages` und `_Hook_V86_Pages`) realisiert das Standard-VxD VDD die textuelle Darstellung von DOS-Boxen im Fenster.

Kleinere Speicherportionen können mit der Funktion `_HeapAllocate` angefordert werden. Zur Verwaltung von verketteten Listen stehen ebenfalls Funktionen zur Verfügung, die auch asynchron gerufen werden dürfen. Damit stehen leistungsfähige Unterprogramme zur Verfügung, die das Programmieren in Assembler erleichtern.

Allozierter linearer Speicher muß nicht auf den Hauptspeicher des Rechners verweisen. Die Seiten können gerade ausgelagert sein. Der VMM kümmert sich beim Zugriff auf nicht-präsenten Speicher automatisch um das Nachladen. Da hierzu nicht-reentrante BIOS- oder VxD-Funktionen bemüht werden, darf dieser Vorgang nicht während einer Interruptausführung durchgeführt werden. Das hat die Konsequenz, daß jedweder Speicher, der in Interruptserviceroutinen angesprochen wird, gegen Auslagern gesperrt werden muß. ISR-Code und Daten müssen sich im "Locked"-Segment des VxD's befinden. Sonstige Speicherbereiche, auf die die ISR zugreift, können mit `_LinPageLock` gesperrt werden. Ein großer Teil des Codes für den zur Arbeit erstellten MPK3D.386-Schrittmotortreiber ist ISR-Code und gezwungenermaßen gesperrt. Verletzt man diese Regel, landet man früher oder später auf dem DOS-Prompt, da sich Windows bei einem Seitenfehler sofort beendet. Ein solcher Fehler ist sehr schwer aufzudecken, zumal er nur dann auftritt, wenn viele Seiten ausgelagert sind, d.h. Windows stark belastet ist.

Häufig ist es notwendig, daß ein VxD ein Stück Speicher für jede virtuelle Maschine zugewiesen bekommt. Das ist vergleichbar mit dem Extra-Fensterspeicher, der bei der Vereinbarung einer Windows-Fensterklasse in seiner Größe festgelegt werden kann (`TWindowClass.cbWndExtra`) und für jedes Fenster verfügbar ist. Die dazu gehörige Funktion lautet `_Allocate_Device_CB_Area` und darf nur zur Initialisierung aufgerufen werden.

Bisweilen ist es auch für VxD's erforderlich, mit Selektoren zu arbeiten. Das im Rahmen dieser Arbeit angefertigte VCALL0.386 zum Beispiel macht von solchen Funktionen Gebrauch, um 16-bit-Code im privilegierten Kernel-Modus laufenzulassen. Ein globaler Selektor wird durch `_Allocate_GDT_Selector` belegt. Seine Attribute, wie Basisadresse und Limit, können auch nachträglich verändert werden. Dabei hilft die Funktion `_BuildDescriptorDWords`, die Bits für den Deskriptortabelleneintrag entsprechend zurechtzuschieben.

Das VxD-API – das VxD wird gerufen

In dem DDB des VxD's sind zwei weitere Adressen angegeben: die V86- und die PM-API-Prozedur-Adresse. Meistens werden beide Zeiger auf eine Prozedur gerichtet, um den gleichen Funktionsumfang zu realisieren. Das Anwenderprogramm kann sich die Einsprungadresse durch Aufruf des Int 2Fh mit AX=1684h und BX=VxD-ID beschaffen. Der Interruptaufruf liefert den Zeiger in ES:DI; bei Nichtvorhandensein der API liefert diese Funktion NULL (nil). Es handelt sich dabei nicht wirklich um eine Einsprungadresse; das wäre zu einfach und würde sich nicht mit dem Privilegierungsschema des 80386-Prozessors vereinbaren. Denkbar wären Call-Gates im Protected Mode. Windows macht aber keinen Gebrauch davon.

Im Protected Mode von Windows 3.1 wird eine Adresse auf einen Int 30h geliefert. Der VMM, der die Interrupts bearbeitet, stellt den Verursacher fest und leitet den Ruf an das richtige VxD weiter. Im Real Mode dagegen wird ein z.B. im BIOS liegender illegaler Befehl angesprungen. Es handelt sich dabei um einen sogenannten System-ROM-Breakpoint. Das Ausführen illegaler Befehle führt zu einer Exception (Ausnahmebedingung Int 06h), die über die Interruptdeskriptortabelle in den VMM führt. Dort wird ebenfalls die Ursache ermittelt und danach das richtige VxD angesprungen. Es ist mit „klassischen" User-Level-Debuggern, wie dem Turbo Debugger von Borland oder CodeView von Microsoft, nicht möglich, solche Kernel-Rufe zu verfolgen.

Beim Ansprung der API werden die Register des Anwenderprogramms nicht direkt übergeben, sondern es wird im Register EBP ein Zeiger auf eine Client-Register-Struktur mitgeliefert, die dann die Anwender-Register enthält. Ein Zugriff erfolgt daher z.B. mit

```
mov   ax,[ebp.Client_AX]
```

Um übergebene Adressen in Abhängigkeit des momentanen Betriebsmodus (V86, 16bit PM, 32bit PM) in Flat-Adressen umzurechnen, steht das Makro `Client_Ptr_Flat` zur Verfügung:

```
Client_Ptr_Flat   esi, es, bx
```

ESI (der erste Parameter) ist das Zielregister, ES:BX die Quell-Client-Register. Man beachte, daß sich die Quelle immer auf die Client-Register bezieht, man muß (und darf im Falle der Segmentregister) also keinesfalls ES und BX vorher mit dem Inhalt der Client-Register laden.

Als Aufrufkonvention für die VxD-API überwiegt die Register-Übergabe. Bei Rückkehr ist es üblich, mit dem Carry-Flag einen Fehler zu signalisieren. Das Setzen des Carry-Flags hat ebenfalls mittels der Client-Register zu erfolgen, wie hier:

```
or    [ebp.Client_Flags], CF_Mask          ;Carry setzen
and   [ebp.Client_Flags], not CF_Mask      ;Carry löschen
```

Mit dem Turbo Assembler TASM können die Makros `SETFLAG` und `MASKFLAG` verwendet werden, um den Code zu optimieren.

VxD-API-Funktionen mit C-Aufrufkonvention haben einen Unterstrich am Anfang ihres Namens zur besseren Kennzeichnung. Die Übergabe von Stackparametern kann unter Verwendung der Makros `VMMCall` bzw. `VxDCall` erfolgen. Das Aufräumen des Stacks wird von dem Makro ebenfalls erledigt. Es wird wie folgt verwendet:

```
VMMCall     _AddInstanceItem, <<OFFSET32 InstStruc>, 0>
```

Man beachte dabei die spitzen Klammern um die Argumente. Das bewirkt, daß bei der Makroexpansion die Parameter als genau ein Argument übergeben werden. Leider bieten Assemblermakros gefährliche Fallen an. Beispielsweise werden nicht nur Kommata, sondern auch Leerzeichen zur Trennung einzelner Parameter herangezogen. Bei der inneren Makroexpansion würde der erste Parameter des Beispiels in `OFFSET32` und `InstStruc` zerfallen, was zu „unerklärlichen“ Assembler-Fehlern führt. Um dies zu verhindern, muß dieser Parameter ebenfalls in spitze Klammern gesetzt werden.

VxD CallOuts – das VxD ruft Anwenderprogramme

Zur Lösung bestimmter Aufgaben ist es nötig, daß sich ein VxD zu einem bestimmten Zeitpunkt nach außen bemerkbar macht. So ist es beim Motortreiber wichtig, daß Windows eine Nachricht erhält, wenn das Bewegungssegment zu Ende ist.

Üblicherweise übergibt die Applikation dem VxD eine Callback-Adresse. Jedoch muß das VxD beachten, daß Adressen mehrfach auftreten können, da mehrere VM's existieren können. Das Callback ist in jedem Fall in der richtigen VM aufzurufen, da ansonsten ein Systemabsturz unvermeidlich ist. Die Callback-Routine muß reentranzfähig sein.

Häufig tritt die Callback-Bedingung asynchron ein, d.h. während eines Interrupts und im Kontext irgendeiner gerade laufenden VM. Dann ist der Scheduler aufzurufen, der zunächst eine Taskumschaltung zur gewünschten VM realisiert. Außerdem muß der Kernel „synchronisiert“, d.h. in einen reentranzfähigen Zustand gebracht werden. Beides nimmt die VMM-Funktion `Call_VM_Event` ab. Außerdem kann `Call_Priority_VM_Event` die Ziel-VM in eine höhere Ausführungspriorität heben. Nach erfolgter Kernelsynchronisierung und Taskumschaltung wird die im Register ESI übergebene Adresse angesprungen.

Danach ist man immer noch im VxD. Jetzt erst kann User-Code abgearbeitet werden. Unter der Annahme, daß die User-Level-Callback-Routine nicht alle Register sichert, tut man dies üblicherweise im VxD unter Benutzung der Makros `Push_Client_State` und `Pop_Client_State`. Die Aufruf-Reihenfolge

```
Push_Client_State
VMMCall  Begin_Nest_Exec
movzx    edx,word ptr [CallbackAddr]
mov      cx,word ptr [CallbackAddr+2]
VMMCall  Simulate_Far_Call
VMMCall  Resume_Exec
VMMCall  End_Nest_Exec
Pop_Client_State
```

führt zur Abarbeitung des Callbacks; die Callback-Routine muß in jedem Falle mit einem Far-Return enden.

Der Aufruf von Hochsprachen-Funktionen mit Parameterübergabe auf dem Stack gestaltet sich mit ein oder mehreren

```
mov      ax,ValueToPush
VMMCall  Simulate_Push
```

vor dem Aufruf von `Simulate_Far_Call` recht einfach. Auf diese Weise kann z.B. die Windows-Funktion `PostMessage()` direkt aufgerufen werden. Sie ist eine der wenigen bei Windows standardmäßig vorhandenen reentranzfähigen Funktionen. Das Anwenderprogramm muß die Adresse von `PostMessage` dem VxD übergeben.

Trap-Behandlung und das Virtualisieren von Ports

Der VMM bietet komfortable Routinen zum Behandeln von Ein-/Ausgabe-Zugriffsverletzungen an. Ein Funktionsverteiler ermittelt beim Sammel-Fehlerinterrupt Int 0Dh (GPF, Allgemeine Schutzverletzung) die Ursache und springt den passenden Handler an. Außerdem übergeht VMM den fehlerauslösenden Befehl. Das ist nötig, um Programme in DOS-Boxen, die auf systemkritische Ports zugreifen, korrekt laufenzulassen, und um andererseits Portzugriffe auf andere (unkritische) Portadressen nicht zu stören. Kann schließlich kein geeigneter Handler gefunden werden, wird ein Standard-Handler gerufen, der eine Fehlermeldung auf dem Bildschirm ausschreibt und zum Schließen der fehlerverursachenden Applikation auffordert.

Installiert wird eine Ein-/Ausgabe-Trapserviceroutine mit der Funktion

```
movzx    edx,[PortAdr]       ;Portadresse, hier wie alles 32bit
lea      esi,IOHandler       ;Ein/Ausgabe-Trapserviceroutine
VMMCall  Install_IO_Handler  ;installieren!
```

Die Routine wird aufgerufen, bevor die Ein/ Ausgabe erfolgt. Bei der Rückkehr übergeht der VMM den fehlerauslösenden Ein-/Ausgabebefehl. Das bedeutet, daß der Ein-/Ausgabebefehl im Anwenderprogramm auf keinen Fall abgearbeitet wird. Innerhalb der Routine kann man alles mögliche ausführen. Der Routine wird u.a. im Register ECX die Art des Portbefehls übergeben. Da es recht aufwendig ist, alle denkbaren Portzugriff-Typen auseinanderzunehmen, gibt es ein Makro, das diese Arbeit abnimmt:

```
Dispatch_Byte_IO  inproc,outproc
```

wobei inproc und outproc Marken (Labels) im Assemblerprogramm sind. Der Portbefehl wird zerlegt und die entsprechende Routine inproc (bei Port-Eingabe) bzw. outproc (bei Port-Ausgabe) ein- oder mehrfach aufgerufen, als ob ein oder eine Reihe der Befehle `in al,dx` bzw. `out dx,al` im Programm gestanden hätte. Manchmal ist es aus Performancegründen nötig, auf dieses Makro zu verzichten, und man muß selbst anhand gesetzter Bits in ECX die Emulation durchführen. Es ist freigestellt, was man mit dem Wert in AL tut (bei Ausgabe) bzw. was man dem Programm in AL liefert (bei Eingabe). Im Falle des Programms MPK3D.386 werden die Nur-Ausgabeports rücklesbar gemacht. Das bedeutet, daß Ausgaben tatsächlich durchgereicht werden und das Ausgabebyte gemerkt wird. Bei Eingabe wird einfach das zuletzt geschriebene Byte zurückgeliefert.

Man beachte, daß der Port-Zugriffsschutz nur für User-Level-Programme gilt. Innerhalb eines VxD's kann man ungehindert Portzugriffe ausführen. Außerdem kann der Zugriffsschutz auf VM-Basis aufgelöst oder wieder eingerichtet werden, da die Ein-/Ausgabe-Berechtigungs-Liste (IOPM) Teil des Task-Status-Segments (TSS) ist, von der jede VM eine besitzt.

Das Virtualisieren von real existierenden Portadressen wird häufig verwendet, um Gerätekonkurrenz aufzulösen. Ohne Virtualisierung könnten z.B. zwei DOS-Programme eine serielle Schnittstelle gleichzeitig programmieren. Was dabei herauskommt, ist weitestgehend Zufall. Die meisten VxDs gehen nach folgendem Schema vor, um das Problem zu lösen:

Zunächst wird in allen VM's der Portzugriff abgefangen. Der erste Zugriff, der erfolgt, bewirkt, daß das VxD die Ports nunmehr für die betreffende VM freigibt (`Disable_Local_Trapping`) und künftig für die anderen VM's so emuliert, als wären die Ports gar nicht vorhanden. Beim ersten Zugriffsversuch einer anderen VM auf ein solcherart gesperrtes Port bringt das VxD häufig eine Meldung aus (Wortlaut in etwa „Das Gerät ist gerade durch ein anderes Programm belegt...“). Es ist aber auch denkbar, daß die Portzugriffe gepuffert werden (z.B. bei einem Druckerspooler), oder die betreffende VM terminiert wird.

Bei diesem Vorgehen verbleibt das Problem, festzustellen, wann die Hardware wieder frei ist. Dies kann über ein spezielles API, eine Zeitüberschreitung (TimeOut) oder beim Schließen der DOS-Box erfolgen.

Eine andere Zielsetzung, Ports zu virtualisieren, kann die Simulierung physikalisch nicht vorhandener Hardware sein. Beispielsweise kann neue, inkompatible Hardware für ältere DOS-Programme kompatibel gemacht werden. Häufig müssen solche VxD's auch Interrupts und DMA virtualisieren. Als Beispiel hierzu sei LPTDAC.386 aufgeführt, das ein Widerstandsnetzwerk am Druckerport in eine SoundBlaster™ 2.0 verwandelt. Für die Entwicklung von Software mit Schutzstecker („Dongle“) könnte für die Erprobung ein „virtueller Dongle“ benutzt werden, solange der echte noch nicht vorhanden oder in Entwicklung ist. Hardware kann auch vollkommen transparent für ein vorhandenes Programm über ein Netzwerk zu einem Hardware-Server propagiert werden.

Interrupt-Verwaltung und VPICD

Das im WIN386.EXE enthaltene Standard-VxD VPICD (Virtual Programmable Interrupt Controller Device) beinhaltet Funktionen, den PIC für jede VM zu virtualisieren sowie das Interruptverhalten insbesondere in DOS-Boxen so zu simulieren, als ob Windows nicht vorhanden wäre. Für eine Verarbeitung von Interrupts in einem VxD werden jedoch nur 2 bis 3 Funktionen von VPICD gebraucht:

```
IrqDesc         VPICD_Irq_Descriptor     <8,0,OFFSET32 CmosIsr>

      lea       edi,IrqDesc              ;Adresse IRQ-Deskriptor-Struktur
      VxDCall   VPICD_Virtualize_IRQ     ;IRQ-Verwaltung an sich reißen
      VxDCall   VPICD_Physically_Unmask  ;IRQ-Leitung freischalten
```

Im IRQ-Deskriptor sind nur die Angaben für IRQ-Nummer (`8`) und die Adresse der Interruptserviceroutine (`CmosIsr`) nötig.

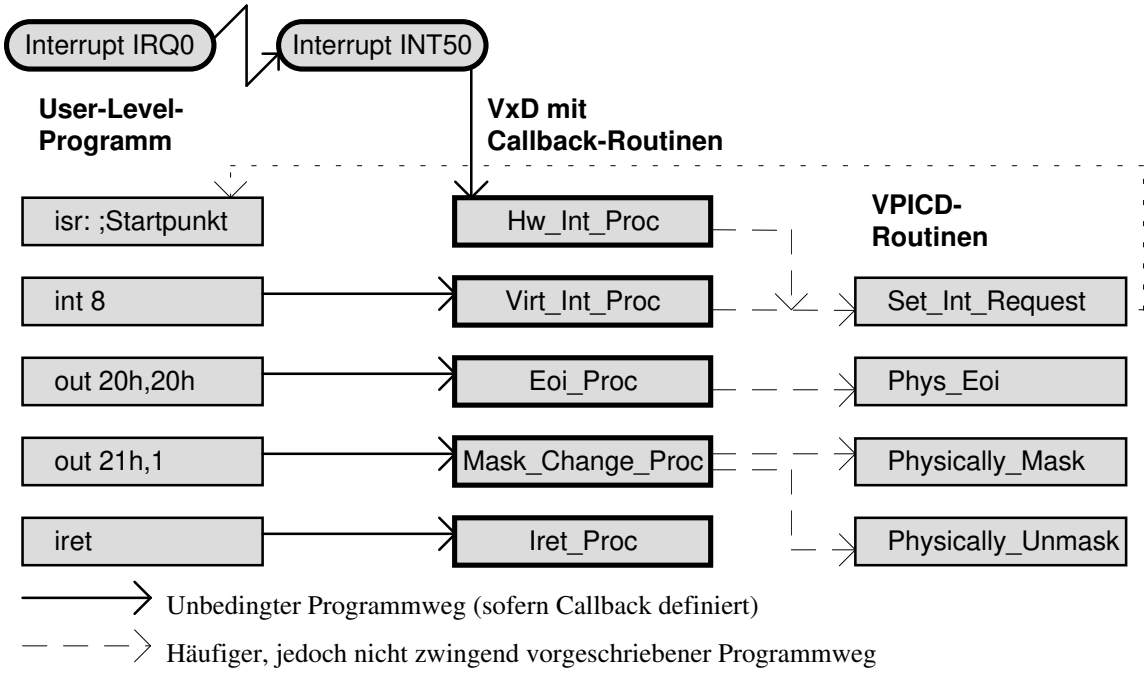


Bild 2.7: Programmfluß bei vollständiger Virtualisierung eines Interrupts mittels VPICD_Virtualize_IRQ

Die Interruptserviceroutine (Hw_Int_Proc im Bild 2.14) weist einige Unterschiede gegenüber solchen z.B. unter DOS auf. Sie wird nicht direkt, sondern von VPICD gerufen. Daher enthalten die Register definierte Werte, insbesondere die Segmentregister. EBX enthält z.B. das Handle der gerade laufenden VM und EAX das IRQ-Handle. Alle Register dürfen verändert werden, und die Routine ist mit einem RET NEAR (statt IRET) zu beenden. Beim Eintritt sind Interrups gesperrt; man kann sie innerhalb der ISR freigeben.

Genau genommen ist die Verwendung der VPICD-Funktionen nicht zwingend. Es ist genausogut möglich, den Interrupt-Controller selbst zu programmieren und in der Interruptdeskriptortabelle einen Zeiger auf seine Serviceroutine einzutragen, die dann mit IRET enden sowie alle benötigten Register, insbesondere Segmentregister retten und setzen muß. Jedoch ist aus Gründen der Kompatibilität die Verwendung der VPICD-Funktionen vorzuziehen. Nur wenn das z.B. aus Performancegründen bedenklich oder aus anderen Gründen nicht machbar ist, sollte man von der direkten Methode Gebrauch machen.

Zusätzlich unterstützt VPICD ab der Windows-Version 3.1 sogenannte „bimodale“ Interrupt-Handler. Dieser Begriff stammt aus der Zeit, als Windows auf dem 286er Prozessor im Standard-Modus lief. Auf diesen Prozessoren gibt es keinen V86-Mode, und für das quasiparallele Laufenlassen von Real-Mode- und Protected-Mode-Code ist eine ständige Hin- und Herschaltung zwischen diesen Betriebsweisen erforderlich. Um diese Belange kümmert sich der in Windows integrierte DPMI-Server. Die Umschaltung vom Protected- zum Real-Mode erfolgt beim 286er Prozessor über einen Reset und ist dementsprechend zeitraubend. DPMI kümmert sich unter anderem darum, daß eine für Protected Mode angemeldete ISR im Protected Mode gerufen wird und umgekehrt. Erfolgt ein Hardware-Interrupt im gerade falschen Betriebsmodus, muß DPMI zu Beginn und zum Ende der ISR umschalten. Die Lösung sind die oben erwähnten speziellen bimodalen Interrupthandler, die in beiden Betriebsmodi des Prozessors lauffähig sind. Die Reaktionszeiten solcher ISR's sind recht gut und mit solchen in VxD's vergleichbar – abgesehen von der geringeren Taktrate der 286er Prozessoren.

VPICD weist eine Schnittstelle via Int 2Fh auf, mit der solche Interruptserviceroutinen im privilegierten Ring 0 laufengelassen werden können, ohne die sonst erforderliche Privilegsumschaltung und ohne DPMI. Somit sind diese Routinen eigentlich „trimodal“, da sie auf 3 verschiedene Weisen gerufen werden können.

Virtualisieren von Direktem Speicherzugriff und VDMAD

Direkter Speicherzugriff (DMA) ist unter Windows ein besondes problematisches Thema. Nicht nur, daß die Abbildung segmentierten in den linearen und von diesem zum physikalischen Speicher zu beachten ist, denn der DMA-Controller arbeitet nur mit physikalischen Adressen. Der aus der Urzeit der Computer-Ära stammende DMA-Controller 8253 verarbeitet nur zusammenhängende Bereiche, und diese müssen sich innerhalb einer 64-KB-Seite (16-bit-Kanäle: 128-KB-Seite) befinden, und er kann nur die ersten 16 Megabyte des gesamten Adreßraumes adressieren.

Zudem ist DMA für DOS-Anwendungen so zu simulieren, als ob Windows gar nicht liefe. (Ein ähnliches Problem hat EMM386.EXE bei DMA-Transfers in den UMB oder EMS.)

VDMAD kann DMA-Anforderungen in Teilübertragungen zerlegen, um das seitenweise Durcheinander im Hauptspeicher zu verbergen. Allerdings können Auto-Init-DMA-Anforderungen nicht zerlegt werden; Anwendungen müssen in diesem Fall einen korrekt adressierbaren Puffer angeben.

In moderneren Computern (PS/2) gibt es auch verbesserte DMA-Schaltkreise. Insbesondere haben solche keine Adreßraum-Beschränkungen und besitzen die Fähigkeit zu Scatter-Gather (Zerstreuen-und-Aufsammeln). Mit letzterem ist gemeint, daß der DMA-Bereich nicht zusammenhängend sein muß, sondern seitenweise im Hauptspeicher verstreut sein kann. Dann kann der DMA-Chip passend zur aktiven Seitentabelle des Prozessors programmiert werden, um irgendeinen zusammenhängenden linearen Bereich ohne Umschweife bearbeiten zu können.

VDMAD virtualisiert sämtliche DMA-Portadressen des PC's. Wenn eine Anwendung die DMA programmiert, programmiert sie in Wirklichkeit nur den virtuellen Status innerhalb von VDMAD. Erst wenn die Anwendung die DMA freigibt, überträgt VDMAD den virtuellen Status zum DMA-Schaltkreis.

Ein VxD hat die Möglichkeit, sich in diesen Mechanismus mittels `VDMAD_Virtualize_Channel` einzuhängen. Er kann die momentane DMA-Programmierung veränden oder abfangen und an andere, verbesserte Hardware weiterleiten.

2.3.5 VxD's unter Windows 95 und Ausblick auf andere Systeme

Gerätetreiber sind systemspezifisch und daher von Natur aus nicht portabel. Das erklärt auch den Sinn, sie in Assembler zu schreiben. Dennoch haben Gerätetreiber verschiedener Systeme auch Gemeinsamkeiten. Die folgenden Abschnitte sollen diese Aspekte beleuchten.

Verwendbarkeit von Windows 3.1-VxD's

Momentan lassen sich Virtuelle Gerätetreiber, die für Windows 3.x geschrieben wurden, beinahe problemlos unter Windows 95 einsetzen. Jedoch plädiert Microsoft nicht auf den Fortbestand der Windows-Kernelarchitektur und setzt für künftige Entwicklungen auf den Windows-NT-Kernel, sodaß VxD's nicht als zukunftsträchtig einzustufen sind.

Windows NT verwendet eine gänzlich andere Kernelstruktur und kann mit VxD's nichts anfangen. Hier werden sog. Kernel-Mode-Treiber eingesetzt, die aufgrund des Micro-Kernel-Konzepts von Windows NT einen anderen Aufbau haben.

Die Kurzlebigkeit ist keinesfalls ein spezielles Problem von Windows-Gerätetreibern. Mit diesem Problem müssen auch andere Betriebssysteme kämpfen, damit das Betriebssystem mit der Entwicklung der Rechner-Hardware schritthalten kann.

Im allgemeinen werden VxD's von Windows-Emulatoren, wie WinOS2 für OS/2, WOW für Windows NT oder WINE für Linux nicht unterstützt. Eine Unterstützung wenigstens einer Untermenge der VxD's wäre ebenso nützlich, wie auch der Erfolg von Windows u.a. darauf setzt, daß es auf DOS und die dort vorhandenen DOS-Gerätetreiber aufsetzt, auch wenn sie noch so alt sind.

Zusätzliche Möglichkeiten bei Windows 95-VxD's

Unter Windows 95 wurde die Endung auf .VXD umgestellt. Damit werden speziell für Windows 95 gefertigte Virtuelle Gerätetreiber kenntlich gemacht, die einen erweiterten Funktionsumfang besitzen.

So können VxD's auch bei Bedarf geladen und auch wieder vom Kernel abgestoßen werden. Es handelt sich dabei um dynamische VxD's, die auf einige weitere Nachrichtencodes in ihrer Steuerprozedur reagieren müssen. Dennoch gibt es auch unter Windows 95 auch statische VxD's.

Unter Windows 95 können statische VxD's in der System-Registry unter `HKEY_LOCAL_MACHINE\System\CurrentControlSets\Services\VxD` eingetragen werden, jedoch besteht aus Kompatibilitätsgründen die SYSTEM.INI sowie die Int 2Fh-Schnittstelle [MSDN]. Zu beachten ist, daß unter Windows 95 jede Win32-Anwendung eine VM darstellt. Außerdem gibt es Threads, die im Sinne der Task-Verwaltung ebenfalls einzelne Tasks darstellen, jedoch innerhalb einer gemeinsamen VM auftreten. Damit verliert sich die bisher gewohnte Einheit von Task und VM. Jeder Thread benötigt ein TSS (Task State Segment).

Bezüglich der VxD-ID gibt es unter Windows 95 parallel noch einen weiteren Mechanismus zur „dynamischen“ Zuteilung einer ID, jedoch nur für VxD's mit Windows 95-Erweiterungen. Die Zuteilung erfolgt anhand von Zeichenketten ähnlich den System-Atomtabellen von Windows. Eine zentrale Anforderung einer ID ist somit nicht mehr nötig. Allerdings dürfen die Zeichenketten nicht doppelt im System vorhanden sein.

Vergleich mit Unix-Gerätetreibern

Gegenüber Gerätetreibern im Multitasking-Klassiker Unix sind Windows-Gerätetreiber als komplexer einzustufen. Einerseits sind die speziellen Belange von DOS-Anwendungen in Boxen zu beachten, und zweitens erledigen Windows-Gerätetreiber eine Menge Arbeit, um korrekt auf DOS aufzusetzen (in dem Sinne, daß Windows nur ein Betriebssystem-Aufsatz ist). Als genereller Unterschied ist noch die gänzlich gegensätzliche Arbeitsweise von Windows und Unix zu beachten. Während Windows immer ein ereignis-orientiertes System darstellt, bei dem das Einschläfern und Aufwecken von Prozessen eher die Ausnahme darstellt, werden unter Unix kaum Nachrichten (Signale) versendet; stattdessen sind eine ganze Reihe Prozesse vorhanden, von denen die meisten auf externe Ereignisse wartend schlafen.

Des weiteren ist die Schnittstelle zwischen Treiber und Anwendungsprogramm fast immer eine Zeichentreiber-Datei (in der Regel im `/dev`-Verzeichnis). Dieses Verfahren findet Anlehnung bei Windows NT. Dagegen werden bei Windows 3.x fast immer spezielle API's verwendet, wobei eine Callback-Funktion bestimmte Signale, wie z.B. die Fertigmeldung, entgegennimmt. Unter Unix dagegen schläfert der Treiber in der Regel den aufrufenden Prozeß solange ein, bis die Anforderung vollständig verarbeitet oder ein Zeichen verfügbar ist. Das hat den Vorteil, daß ein Programm nicht merkt, ob ein Treiber synchron (z.B. Polling unter voller CPU-Last mit gelegentlichem Aufruf von `schedule()`) oder asynchron (interruptgetrieben) arbeitet. Ein Programm, das nebenher noch andere Aufgaben erledigen möchte, muß einen neuen Prozeß abspalten (`fork()`) oder einen neuen Thread kreieren. Gegenüber Callbacks erfordern diese Verfahren mehr Rechenleistung, führen aber auch zu strukturell einfacheren und besser wartbaren Programmen ohne den für manche Windows-Programme typischen „Callback-Spaghetti“.

Eine Sonderrolle spielen rein monolithische Unix-Kernel. Obwohl der Kern durchaus Hierarchiestufen aufweist und in verschiedene Quelldateien aufgeteilt ist, besteht er letztlich aus genau einem Stück Code. Hierzu Gerätetreiber zu entwickeln bedeutet häufig, in globalen Datenstrukturen zu stöbern und durch geeignete Makros oder gar Patches Zugang zu verschaffen. Die Installation eines neuen Treibers erfolgt durch Veränderungen am `makefile`, der Neucompilierung und schließlich Booten des Systems.

Obwohl beide Systeme ähnliche Kernel-Funktionen zur Verfügung stellen (allerdings mit stark abweichenden Namensgebungen) und beide sowohl Signale als auch Einschläfern/Aufwecken unterstützen, ist bei der Treiberprogrammierung konzeptionell anders vorzugehen, um unter Unix einen „typisch Unix“ und unter Windows einen „typisch Windows“-Treiber zu erhalten.

2.4 Einführung zu DDE

Die folgenden Abschnitte beschäftigen sich mit den Möglichkeiten des dynamischen Datenaustausches zwischen Anwendungen unter Windows. So stehen in Borland Pascal alle Funktionen der Windows-API zur Verfügung, während beispielsweise für Matlab-Skripte angepaßte Funktionen angeboten werden. Deshalb werden in den folgenden Abschnitten die Aufrufe von Funktionen des DDE am Beispiel von Funktionen der Programmiersprache Borland Pascal für Windows beschrieben. Die (unter Beachtung der jeweiligen Sprachkonventionen) gleichen Funktionen stehen in den C-Compilern Microsoft C/C++ 7.0 und Borland C++ 3.1 zur Verfügung.

2.4.1 Zwischenablage-Formate als Basis für den Datenaustausch

Für den DDE-Datenaustausch wurden dieselben Austauschformate festgelegt wie für die Zwischenablage. Die in Tabelle 2.3 aufgeführten Konstanten finden daher auch unter DDE Verwendung. Am häufigsten wird das Zwischenablageformat `CF_TEXT` verwendet. Es wird von jedem Windows-Programm akzeptiert.

Formatbezeichner	**Erläuterung**
CF_TEXT	Textblock, abgeschlossen mit einem NUL-Zeichen, der aus Zeichen des ANSI-Codes besteht
CF_BITMAP	Bitmap im Format der Windows Version 2
CF_METAFILEPICT	Bild in Form einer Zwischendatei des Formats Metafilepict
CF_SYLK	Microsoft spezifisches Textformat (Symbolic Link) mit Längenangabe
CF_DIF	von Visicalc eingeführtes Data Interchange Format mit eigener Format und Längeninformation
CF_TIFF	Tag Image File Format zur Beschreibung von Bitmaps
CF_OEMTEXT	Textblock, abgeschlossen mit einem NUL-Zeichen, der aus Zeichen des Hersteller-Zeichensatzes (IBM, aktuelle Codeseite) besteht
CF_DIB	Bitmap im geräteunabhängigen Format der Windows Version 3
CF_PALETTE	Farbpalette, die meist verwendet wird, um di Palette für ein Bitmap des Format CF_DIB zu bestimmen

Tabelle 2.3: Standard-Zwischenablage-Formate

Daneben existiert die Möglichkeit, eigene Zwischenablageformate zu verwenden. Dazu müssen sie registriert werden. Der Vorteil liegt in einem möglichst schnellen Austausch großer Datenmengen ohne Konvertierungsaufwand. Nachteilig ist die fehlende Portabilität solcher Formate.

2.4.2 Datenaustausch per DDE

Unter Windows 2.x stand nur die Zwischenablage (Clipboard) zum standardisierten Datenaustausch zur Verfügung. Nachteil des Clipboards ist, daß keine Nachrichten auf Veränderungen im Clipboard verteilt werden, und das Clipboard eine Windows-globale Resource darstellt, die dem Anwender vorbehalten sein sollte.

Das brachte Microsoft dazu, den Datenaustausch unter Windows um den Dynamic Data Exchange DDE zu erweitern. Damit steht dem Programmierer ein nachrichtenbasiertes Protokoll zum Austausch von kurzen Informationen (1 Word `wParam` und 1 LongInt `lParam`) zur Verfügung. Die Erweiterung des übertragbaren Datenvolumens erfolgt in der Regel durch Allozierung gemeinsam nutzbaren (shareable) Speichers und Übergabe des zugehörigen Handles.

DDE unterstützt den Aufbau von Client-Server-Modellen. So kann jede Anwendung, die das DDE-Protokoll unterstützt, als Client bei einer anderen Anwendung Daten anfordern oder Befehle an sie weiterleiten. Die Server-Anwendung wird in die Lage versetzt, angeforderte Daten an die entsprechende Anwendung zu senden. Dabei kann jede Anwendung sowohl Client- als auch Serverfunktionen wahrnehmen.

Die Adressierung von Daten wird im DDE-Protokoll durch eine dreistufige Hierarchie gekennzeichnet. Bild 2.8 stellt diese Hierarchie am Beispiel der Tabellenkalkulation EXCEL und der Textverarbeitung WinWord dar.

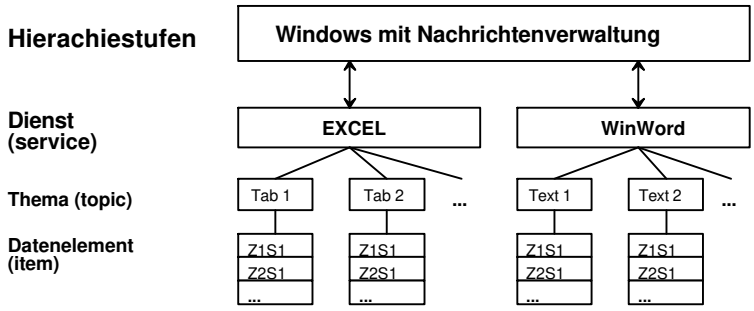


Bild 2.8: Dreistufige Hierarchie bei der Adressierung der Daten durch das DDE-Protokoll

Die oberste Stufe der Hierarchie bilden die Dienste (services). Ein Programm kann mehrere DDE-Dienste anbieten. Der geläufige Fall ist jedoch, daß eine Anwendung genau einen DDE-Dienst unter demselben Namen wie den Programmnamen anbietet. Jeder Dienst besitzt ein oder mehrere Themen (topics). Diese Themen können die einzelnen Dokumente einer MDI-Anwendung sein. Es sind aber auch andere Themen denkbar. So besitzen z.B. alle Microsoft-Anwendungen das Thema „System“. Dieses enthält Informationen über die jeweilige Anwendung. Die unterste Kategorie stellen die einzelnen Datenelemente (items) selbst dar. Bei der Tabellenkalkulation können das einzelne Zellen oder speziell vereinbarte Bereiche sein. Bei der Textverarbeitung sind es Textzeichen oder ebenfalls speziell vereinbarte Bereiche. Hat ein Programm nur ein Thema, wird zur Überwindung dieser Hierarchiestufe zumeist derselbe Name wie beim DDE-Dienst verwendet.

DDE gibt es in zwei Entwicklungsstufen. Aus Kompatibilitätsgründen sind in der aktuellen Windows Version 3.11 beide verfügbar. Die erste Form des DDE ist eine auf gewöhnliche Fensternachrichten basierende. Die zweite Form ist seit Windows 3.1 verfügbar und arbeitet mit einer DLL. Sie ist kompatibel zum DDE unter Windows NT.

2.4.3 Nachrichtenbasierter Datenaustausch per DDE

Daß unter Windows Anwendungen allgemein durch Nachrichten kommunizieren, macht sich der dynamische Datenaustausch DDE zunutze. Die erste, ältere Form des DDE besteht hauptsächlich aus den in Tabelle 2.4 aufgeführten Nachrichten und Microsofts Vorschrift, wie man diese anzuwenden hat. Im nachrichtenbasierten Modell heißen die DDE-Dienste “application” (Anwendung) und nicht “service”, was zu Verwirrungen führen kann. Daher sei im folgenden „Dienst“ bzw. „service“ als Name für die oberste Hierarchiestufe festgelegt.

Befehl (Nachricht)	**Send/Post**	**Wirkung**
wm_dde_ack	Post	Empfangsbestätigung einer DDE-Nachricht
wm_dde_advise	Post	Aufbau eines “warm links” oder “hot links”, Server muß Änderung von Daten bzw. geänderte Daten an Client senden
wm_dde_data	Post	Senden von Daten an Client (durch Übergabe eines Handles)
wm_dde_execute	Post	Ausführen eines Befehls

wm_dde_initiate	Send	Aufforderung zum Verbindungsaufbau
wm_dde_poke	Post	Senden von Daten an Server
wm_dde_request	Post	Anforderung von Daten
wm_dde_terminate	Post	Beenden einer DDE-Verbindung
wm_dde_unadvise	Post	Abbau eines “warm links” oder “hot links”

Tabelle 2.4: Die in Windows verfügbaren DDE-Botschaften

Mit Hilfe dieser DDE-Nachrichten können zwei Anwendungen untereinander eine Verbindung aufbauen. Dabei wird die aktive Anwendung dieser Verbindung als Client bezeichnet. Sie löst den Verbindungsaufbau aus und fragt bei einer vorhandenen Verbindung nach Daten an. Die andere Anwendung (der Server) spielt eine passive Rolle. Sie reagiert auf Anfragen zum Verbindungsaufbau und bei einer bestehenden Verbindung auf Anfragen nach Daten.

Gesendet werden diese Nachrichten mit der Funktion

```
SendMessage(Wnd: HWnd; Msg, wParam: Word; lParam: LongInt);
```

mit sofortiger Bearbeitung und Rückantwort oder mit

```
PostMessage(Wnd: HWnd; Msg, wParam: Word; lParam: LongInt);
```

unter Einfügung dieser Nachricht in die Nachrichtenwarteschlange des Zielfensters. Welche der beiden Funktionen zu verwenden ist, hängt von der verwendeten Nachricht ab und steht an Spalte 2 der Tabelle 2.4.

Die beiden Nachrichtenparameter werden dabei zur Übertragung von Zusatzinformationen wie z.B. einem Handle der sendenden Anwendung oder der Bestätigungsart einer `wm_dde_ack`-Nachricht (positiv, negativ) benutzt.

Verbindungsaufbau

Wie in Bild 2.9 zu sehen ist, beginnt der Client mit dem Senden eines `wm_dde_initiate` den Verbindungsaufbau. Da das Handle des Zielfensters unbekannt ist, wird dafür `HWND_BroadCast` (=$FFFF) eingesetzt. Diese Konstante bewirkt, daß alle sichtbaren Hauptfenster auf dem Bildschirm diese Nachricht erhalten.

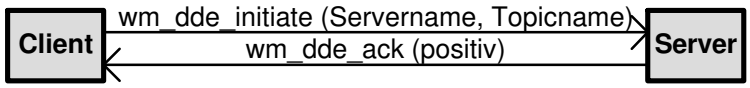


Bild 2.9: Erfolgreicher Verbindungsaufbau zwischen 2 DDE-Anwendungen

Daraufhin überprüft jede Server-Anwendung, ob der Servername mit dem eigenen übereinstimmt und ob sie das gesuchte Thema (topic) anbietet. Sollte dies der Fall sein, sendet die Server-Anwendung ein `wm_dde_ack` an den Client als positive Bestätigung. Das bei dieser Rückantwort übermittelte Fenster-Handle dient fortan als Ziel für eine gerichtete Kommunikation zwischen Client und Server. Sollte eine Server-Anwendung in einem Punkt keine Übereinstimmung feststellen, erfolgt keine Reaktion.

Anforderung von Daten vom Server

Nach einem erfolgreichen Verbindungsaufbau kann der Client nun beispielsweise beim Server nach Daten anfragen. Dazu dient die Nachricht wm_dde_request (DDE-Anforderung). Der Client erhält keinerlei Information über eine Änderung der von ihm abgefragten Daten. Bild 2.10 stellt einen erfolgreichen und einen mißglückten Datentransfer dar.

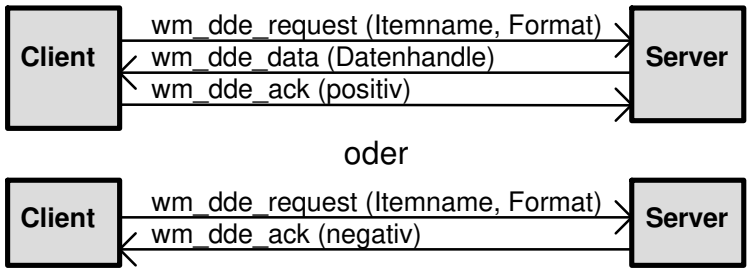


Bild 2.10: Datenanforderung

Hat der Server die angeforderten Daten nicht oder nicht im gewünschten Format, antwortet er auf die `wm_dde_request`-Nachricht mit einem negativen `wm_dde_ack`. Als Formate werden die schon in Tabelle 2.3 aufgeführten Formate der Zwischenablage verwendet. Die Definition eigener Formate ist auch möglich. Davon wird insbesondere bei Übertragung großer Mengen binärer Daten Gebrauch gemacht. Die Liste der verfügbaren Datenformate kann vom Server über das System-Thema vom Datenelement “Formats” bezogen werden, sofern das System-Thema unterstützt wird..

Hat der Client nun Daten angefordert, die der Server auch im entsprechenden Format besitzt, sendet er ein Handle auf diese Daten mittels `wm_dde_data`. Der Empfang dieses Handles wird durch den Client mit dem Senden eines positiven `wm_dde_ack` bestätigt.

Unter DDE existiert die für Clients die Möglichkeit, sich automatisch bei Datenveränderungen beim Server informieren zu lassen. Dabei gibt es zwei Verbindungsarten. Eine davon ist der “warm link”. Bei dieser Verbindungsart informiert der Server den Client über eine Änderung der zuvor gesendeten Daten. Bei einem “hot link” (Bild 2.11) werden statt der Information über die Änderung der Daten die geänderten Daten selbst gesendet.

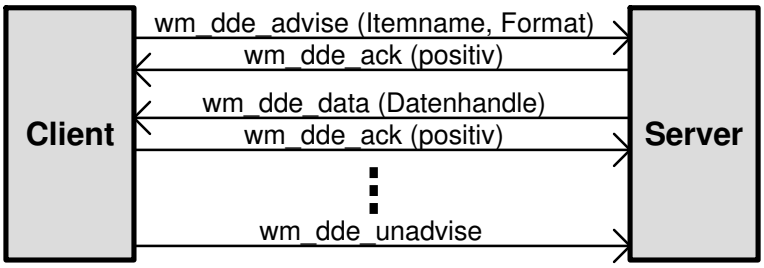


Bild 2.11: Erfolgreicher Datentransfer mittels “hot link”

Sowohl der “warm link” als auch der “hot link” werden durch das Senden einer `wm_dde_advise` Nachricht vom Client aufgebaut. Wie bei einer Datenanforderung überprüft der Server daraufhin, ob er über die gewünschten Daten im geforderten Format verfügt, und antwortet mit einem entsprechenden `wm_dde_ack`. Bei einer Datenveränderung sendet der Server bei “hot link” mittels einer `wm_dde_data`-Nachricht ein Handle auf die Daten zum Client. Diese Nachricht wird mittels `wm_dde_ack` bestätigt. Bei einem “warm link” sendet der Server nur `wm_dde_ack` zum Client. Der Client muß sich in diesem Falle die Daten mit einer `wm_dde_request`-Nachricht beschaffen. Sowohl “warm link” als auch “hot link” werden durch den Client beendet. In diesem Fall sendet er eine `wm_dde_unadvise`-Nachricht an den Server.

Senden von Daten vom Client zum Server

Für den Fall, daß ein Client von sich aus Daten an einen Server senden muß, bietet das DDE-Protokoll die `wm_dde_poke`-Nachricht an.

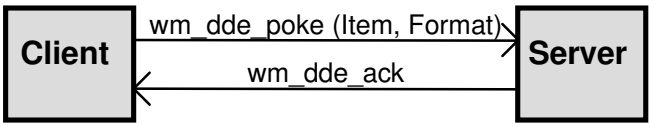


Bild 2.12: Datenübertragung vom Client zum Server mittels `wm_dde_poke`-Nachricht

Hat der Server die vom Client angebotenen Daten erfolgreich übernommen, bestätigt er dies mit dem Senden einer positiven `wm_dde_ack`-Nachricht. Konnte er die Daten nicht übernehmen, sendet er eine negative `wm_dde_ack`-Nachricht. Die `wm_dde_poke`-Nachricht zieht bei korrekter Ausführung den oben erläuterten Benachrichtigungszyklus bei “warm link” und “hot link” nach sich, um denselben als auch weitere Clients über die (gezielte) Datenveränderung zu informieren.

Senden eines Befehls

Zusätzlich zu den Nachrichten für den Datenaustausch bietet das DDE-Protokoll noch eine Möglichkeit zum Versenden von allgemein gehaltenen Befehlen.

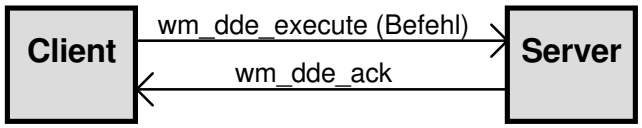


Bild 2.13: Senden eines Befehls an einen Server

Mittels einer `wm_dde_execute`-Nachricht kann der Client eine Befehlszeichenkette an den Server schicken. Konnte der Server die Zeichenkette verarbeiten, bestätigt er dies mit einer positiven `wm_dde_ack`-Nachricht andernfalls mit einer negativen `wm_dde_ack`-Nachricht.

Verbindungsabbau

Eine Verbindung kann durch beide Seiten jeweils durch Versenden der DDE-Nachricht `wm_dde_terminate` beendet werden. Der die Verbindung zuerst auflöst bezeichnet man als Initiator. Der Verbingungsabbau wird durch die jeweils andere Anwendung ebenfalls durch Senden von `wm_dde_terminate` bestätigt.

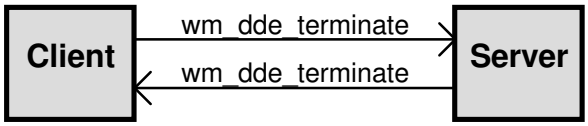


Bild 2.14: Verbindungsabbau

Auswahl der Nachrichten und Antworten

Je nachdem, ob eine Anwendung in einer DDE-Verbindung Client oder Server ist, darf sie nur einen Teil der DDE-Nachrichten benutzen. Tabelle 2.5 ordnet die DDE-Nachrichten je nach Rolle der Anwendung in der DDE-Verbindung (Client oder Server) zu und führt mögliche Reaktionen auf.

Befehl (Nachricht)	Absender	mögliche Reaktion	Hinweis
wm_dde_ack	Client, Server	-	enthält Information über Art der Bestätigung
wm_dde_advise	Client	wm_dde_ack	Server schickt bei Datenänderung wm_dde_Data
wm_dde_data	Server	wm_dde_ack	
wm_dde_execute	Client	wm_dde_ack	
wm_dde_initiate	Client	wm_dde_ack	
wm_dde_poke	Client	wm_dde_ack	
wm_dde_request	Client	wm_dde_ack wm_dde_data	Daten senden nicht möglich Daten senden ok.
wm_dde_terminate	Client, Server	wm_dde_terminate	
wm_dde_unadvise	Client	wm_dde_ack	

Tabelle 2.5: Absender von DDE-Nachrichten und möglich Reaktionen

Da eine Anwendung nebeneinander DDE-Verbindungen als Client und als Server aufbauen kann, darf sie auch alle Nachrichten benutzen. Maßgebend ist nur, welche Rolle sie in der jeweiligen Verbindung spielt – Client oder Server.

Probleme mit nachrichtenbasiertem DDE

Beim Aufbau einer DDE-Verbindung ist eine Client-Anwendung nicht in der Lage festzustellen, ob die gewünschte Server-Anwendung schon geladen ist. Standard-Clients versuchen, eine Datei mit dem Service-Namen der DDE-Hierarchie mittels der Windows-Funktion `WinExec()` zu starten. Mißlingt das, erscheint z.B. ein Dialogfeld, das zum Laden der Server-Anwendung auffordert. Da die Server-Anwendung generell nicht im Windows-Suchpfad steht, schlägt `WinExec()` meistens fehl. Daher muß der Anwender dafür sorgen, daß die Server-Anwendung zuerst gestartet wird.

Das DDE-Protokoll an sich bietet keine Möglichkeit, die Namen einer Anwendung, eines Themas oder die Syntaxbeschreibung zu erfragen. Der Programmierer einer Client-Anwendung ist von der Dokumentation der verwendeten Server-Anwendungen abhängig. Daher wurde von Microsoft festgelegt, daß alle DDE-Server ein System-Thema aufweisen, in dem in einer festgelegten Hierarchie die Namen und Fähigkeiten des Servers aufzulisten sind. Das ist jedoch kein Muß, und nicht jeder Client vermag das System-Topic auszuwerten. In der Regel müssen die verfügbaren Namen bekannt sein.

2.4.4 Datenaustausch über die DDE-Management-Bibliothek DDEML

Mit der Auslieferung der Windows Version 3.1 führte Microsoft auch die DDE Management Library (DDEML) ein. An der Funktionsweise des darunterliegenden DDE-Protokolls änderte sich damit nichts. Einziger Zweck der DDEML ist es, für den Programmierer eine Schnittstelle zu definieren, die Unterstützung bei der Nutzung und Verwaltung des DDE-Protokolls bietet. Unabhängig, ob eine Anwendung die DDEML nutzt oder nicht, funktioniert die Nachrichtenform des DDE-Protokolls auch weiterhin.

Im Gegensatz zur Nachrichtenform des DDE werden Aktionen nun durch die Benutzung der Funktionen der DDEML ausgelöst, von denen für Serveranwendungen in Frage kommende in Tabelle 2.6 aufgeführt sind. Zusätzlich wurden Funktionen hinzugefügt, die der Verwaltung gemeinsam nutzbaren Speichers dienen sowie solche zum Festlegen von sitzungs-globalen eineindeutigen Kennzahlen (Stringhandles, ein anderer Name für sog. „Atome“), basierend auf Zeichenketten.

DDEML Funktion	Beschreibung
DdeAccessData()	Greift auf ein globales DDE-Speicherobjekt zu
DdeAddData()	Fügt einem globalen DDE-Speicherobjekt Daten hinzu
DdeCmpStringHandles()	Vergleicht zwei DDE-String-Handles
DdeCreateDataHandle()	Erzeugt ein DDE-Daten-Handle
DdeCreateStringHandle()	Erzeugt ein DDE-String-Handle
DdeEnableCallback()	Aktiviert oder deaktiviert eine oder mehrere DDE-Konversationen
DdeFreeDataHandle()	Gibt ein globales Speicherobjekt frei

DdeFreeStringHandle()	Gibt ein DDE-String-Handle frei
DdeGetData()	Kopiert Daten eines globalen Speicherobjekts in einen Puffer
DdeGetLastError()	Liefert einen Fehlercode der durch eine DDEML-Funktion gesetzt wurde
DdeInitialize()	Registriert eine Anwendung mit der DDEML
DdeKeepStringHandle()	Inkrementiert den Benutzungszähler für ein String-Handle
DdeNameService()	Registriert bzw. streicht einen Servicenamen
DdePostAdvise()	Veranlaßt einen Server, Hinweisdaten an den Client zu senden
DdeQueryConvInfo()	Liefert Informationen über eine DDE-Konversation
DdeQueryString()	Kopiert Text eines String-Handles in einen Puffer
DdeSetUserHandle()	Ordnet einer Transaktion ein benutzerdefiniertes Handle zu
DdeUnaccessData()	Gibt ein globales DDE-Speicherobjekt frei
DdeUninitialize()	Gibt die DDEML-Ressourcen einer Anwendung frei

Tabelle 2.6: DDEML-Funktionen, die für DDE-Server in Frage kommen

Für die Reaktionen werden Rückruffunktionen (Callbacks) genutzt. Jede die DDEML nutzende Anwendung muß sich zuallererst mit der Funktion `DdeInitialize()` bei der DDEML anmelden. Dabei wird eine Rückruffunktion vereinbart. Diese Funktion kann die DDEML aufrufen, um ihr sogenannte Transaktionen als Reaktion zu übergeben. In Tabelle 2.7 befindet sich eine Liste aller bei Serveranwendungen eintreffenden möglichen Transaktionen.

Transaktion	**Beschreibung**
XTYP_ADVSTART	Ein hot oder warm link bezüglich eines Datenelements soll gestartet werden.
XTYP_ADVSTOP	Ein hot oder warm link soll beendet werden.
XTYP_CONNECT	Ein Client möchte eine DDE-Verbindung aufbauen.
XTYP_CONNECT_ CONFIRM	Diese Transaktion ist die Bestätigung der Einrichtung einer Konversation mit einem Client.
XTYP_DISCONNECT	Die DDEML-Verbindung wird gelöst.
XTYP_ERROR	Ein kritischer Fehler (z.B. Speicherplatzmangel oder sonstige Probleme mit einer Systemressource) ist aufgetreten.
XTYP_EXECUTE	Der Client übergibt dem Server ein ausführbares Kommando.
XTYP_POKE	Der Client ändert Datenelemente auf dem Server.
XTYP_REGISTER	Ein Server hat die Funktion DdeNameService benutzt, um einen Servicenamen (Anwendungsname) zu registrieren.
XTYP_REQUEST	Der Client benötigt ein Datenelement.
XTYP_UNREGISTER	Ein Server hat die Funktion DdeNameService benutzt, um die Registrierung eines Servicenamens (Anwendungsname) aufzuheben.
XTYP_ WILDCONNECT	Ein Client hat einen Servicenamen mit dem Wert NULL, einen Themennamen mit dem Wert NULL oder beides in einem Aufruf der Funktion DdeConnect angegeben.

Tabelle 2.7: DDEML-Transaktionen, die bei Server-Anwendungen eintreffen können

Eine Anwendung, die die DDEML nutzt, kann folgende Vorteile in Anspruch nehmen:

- kürzerer Code durch geringeren Verwaltungsaufwand

Bei einer Nachrichten nutzenden DDE-Anwendung entstand ein relativ hoher Verwaltungsaufwand, da die jeweilige Anwendung ihre Verbindungen selbst verwalten mußte. Dieser Aufwand fällt nun weg. Eine Anwendung, die die DDEML nutzt, muß sich bei der DDEML nur noch anmelden und ggf. eine Rückruffunktion (callback) vereinbaren. Um sinnvoll zu funktionieren, müssen DDE-Server eine Rückruffunktion aufweisen, für Clients ist sie kein Muß. Alle DDE-Verbindungen werden durch die DDEML verwaltet und mit einem Konversationshandle gekennzeichnet. Insbesondere wird die Verwaltung mehrerer Clients durch einen Server vereinfacht.

- bessere Kontrollmöglichkeiten

Bei Verwendung der DDEML wird eine DDE-Aktion nicht mehr durch das Senden einer Nachricht ausgelöst, sondern durch den Aufruf einer entsprechenden Funktion aus der DDEML. Zu diesen Funktionen gehören auch solche, die Informationen über beliebige DDE-Verbindungen liefern. Das erleichtert nicht nur das Einfügen von Kontrollmechanismen in eigene Anwendungen, sondern auch das Erstellen von Anwendungen, die den Dynamischen Datenaustausch überwachen.

- verbesserter Datenschutz

Das Wegkommen von Fenster-Nachrichten hin zu ausschließlich für DDE bestimmten Datenkanälen gestattet es in geschützten Systemen wie Windows NT, Lauschangriffe durch nicht privilegierte Programme bzw. Anwender abzuwehren. Programme sind durch Verwendung der DDEML zu Windows NT portabel.

- Nutzung von Verbesserungen des DDE-Mechanismus

Da der wesentliche Teil der DDE-Funktionalität in der DDEML abläuft, profitieren alle die DDEML nutzenden Anwendungen von ihren Upgrades. Beispielsweise ist unter Windows für Workgroups ein Verbindungsaufbau über Rechnergrenzen hinweg über das LAN möglich. Die Rolle der DDEML zeigt Bild 2.15.

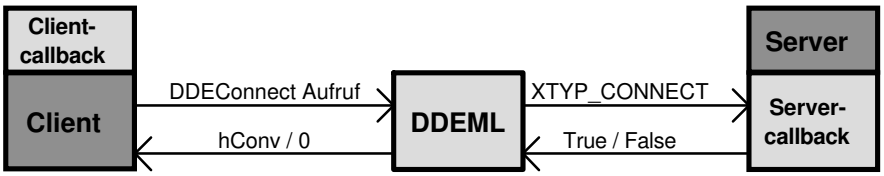


Bild 2.15: Position der DDEML am Beispiel der DDEConnect Funktion

Als Beispiel wurde hier ein Verbindungsaufbau gewählt. Zunächst muß die DDEML geladen und initialisiert werden. Danach kann der Client damit beginnen, sich mit der `DDEConnect()`-Funktion mit dem Server zu verbinden. Ein ebenfalls bei der DDEML angemeldeter Server erfährt mittels der Transaktion `XTYP_Connect` von dem Verbindungsgesuch des Clients. Ist er verbindungswillig, gibt die Rückruffunktion des Servers True (einen Wert ungleich Null) zurück. Andernfalls liefert die Rückruffunktion False (Nullwert). Bei einer erfolgreichen Anfrage beim Server erhält der Client darauf von der DDEML ein Konversationshandle. Bei einem negativen Ausgang der Anfrage würde der Client einen Nullwert von der DDEML erhalten.

Als bemerkenswert gilt, daß die DDEML einem DDE-Server den Unterschied zwischen “warm link” und “hot link” verbirgt. Ein DDE-Server braucht sich obendrein nicht darum zu kümmern, ob Clients überhaupt solche Verbindungen angelegt haben, ein Server kann in jedem Falle einer Datenveränderung `DdePostAdvise()` aufrufen. Die DDEML kümmert sich für jeden einzelnen Clienten um die automatische Datenaktualisierung.

3 Umsetzung der Programmieraufgaben

Entsprechend der Aufgabenstellung und den technischen Erfordernissen wird die Software für Windows entwickelt. Dazu werden nach Möglichkeit Funktionen verwendet, die Windows von vornherein anbietet, um einerseits Entwicklungsaufwand und Codegröße zu minimieren, als auch eine durchgehend konsistente Bedienoberfläche unter Windows zu erhalten. Besondere Beachtung finden dabei die in den Abschnitten 2.3 und 2.4 erläuterten speziellen Themen der Windows-Programmierung.

3.1 Treiber für Positioniermotor MPK3D.386

Aufgrund des geforderten Echtzeitverhaltens der Ansteuersoftware im unteren Millisekunden-Bereich fiel die Entscheidung, den gesamten Motortreiber von Grund auf neu als Virtuellen Gerätetreiber für Windows zu entwickeln. Es findet eine Festfrequenz-Interruptquelle Verwendung. Dazu werden neue Wege beschritten, um einen günstigen Weg der Ansteuerung, insbesondere für Beschleunigung und Abbremsen, zu finden.

3.1.1 Mögliche Ansteuertechniken

Um die Massenträgheit der Schrittmotorachse und der angeschlossenen Mechanik sicher ohne Schrittverlust zu überwinden und dennoch hohe Verfahrgeschwindigkeiten zu erreichen, ist ein korrekter Beschleunigungs- und Abbremsvorgang für jedes Bewegungssegment erforderlich. Bei Schrittmotoren ohne Mikroschritt-Ansteuerung erfolgt dies meist durch eine variable Ansteuerfrequenz. Beispielsweise kann ein Zeitgeber mit variablen Zeiten programmiert werden. Pro Interrupt wird genau 1 Schritt (Voll- oder Halbschritt) ausgeführt. Vorteilhaft ist hierbei die einfache Berechnung der Verzögerungszeiten.

Dieses Verfahren ist auch für den Mikroschrittbetrieb denkbar. Durch die Kleinheit eines Mikroschritts bedingt sind sehr hohe Interruptfrequenzen nötig, um gleiche Geschwindigkeiten zu erreichen. Andererseits ist es möglich, pro Interrupt mehrere Mikroschritte bis hinauf zu einem Vollschritt durchzuführen. Die Höhe der nötigen Interruptfrequenz fällt im gleichen Maße. Um dennoch Positioniergenauigkeiten im Mikroschrittbereich zu erreichen, ist eine geeignete Berechnung durchzuführen. Bei variabler Interruptfrequenz entscheidet der Interruptabstand über die Geschwindigkeit. Mehrere Mikroschritte können in einem Interrupt zusammengefaßt ausgeführt werden, um die Interruptfrequenz nach oben zu begrenzen. Dieses Berechnungsverfahren ist gegenüber der o.g. Vollschrittansteuerung kaum komplizierter.

Dieses Programm arbeitet mit einer festen Interruptfrequenz. Sie wird durch die CMOS-Echtzeituhr geliefert. Da diese Interruptquelle von anderen Programmen meist nicht verwendet wird, ergeben sich die geringstmöglichen Stör- oder Komplikationsmöglichkeiten mit anderer Software auf demselben Rechner. Die Ursachen für die seltene Verwendung in Anwenderprogrammen liegt daran, daß

- die Echtzeituhr erst seit den Rechnern der AT-Klasse zum Standard gehört
- die Interruptfrequenz nicht völlig frei programmierbar ist
- die Programmierung der Echtzeituhr wenig dokumentiert ist.

Die feste Interruptfrequenz hat zur Folge, daß an den Rampen (Beschleunigungs- und Bremsphase) pro Interrupt null, ein oder mehrere Mikroschritte auszuführen sind. Dazu wird intern mit „Nanoschritten“ gerechnet. 65536 Nanoschritte ergeben einen Mikroschritt, 256 Mikroschritte einen Vollschritt, und 4 Vollschritte eine Periode. Der angeschlossene Motor hat 200 Schritte pro Umdrehung, somit 50 Perioden (das sind die möglichen Raststellungen eines erregten Motors). Die aktuelle Nanoschritt-Position ist eine 64 bit breite Variable.

Die Angaben für Geschwindigkeit und Beschleunigung haben ebenfalls intern die Einheit Nanoschritt pro Tick bzw. Tick² und sind 32 bit breit. Dabei ist ein Tick die Zeit zwischen 2 Interrupts und somit eine Programmkonstante. Um diese Programmkonstante zu verbergen und stattdessen SI-Einheiten zu verwenden, werden sämtliche Angaben für Geschwindigkeit und Beschleunigung nach außen in Schritte/s bzw. Schritte/s² umgerechnet.

Die Schrittzahl des Motors pro Umdrehung (bzw. pro Millimeter bei Verwendung einer Spindel) kann ebenfalls durch Setzen eines Getriebefaktors verborgen werden. Dieser Getriebefaktor wird in diesem Programm als Bruch geführt, d.h. es sind Zähler und Nenner zu übergeben. Das vermeidet das Rechnen mit Fest- oder gar Gleitkommazahlen innerhalb des Treibers und ist aus der Programmiersprache FORTH bekannt.

3.1.2 Kurvenform

Das Originalprogramm verwendet eine sinusförmige Stromsteuerung, deutlich erkennbar an der im Programmcode eingebauten Sinustabelle. Jedoch ist dadurch die Gesamtstromaufnahme schwankend. Dabei schwankt auch das Haltemoment und vermutlich auch der Mikroschrittwinkel. Das verschlechtert die mechanisch machbaren Werte hinsichtlich Positioniergenauigkeit. Erforderlich ist eine Untersuchung, wie die Kurvenform, die je nach Motorkonstruktion und Linearität der Ansteuerschaltung von der idealen Sinusform abweicht, aussieht. In jeder Stellung sollte einerseits das Haltemoment gleich sein (Gesamtstrombedingung) als auch der erwartete Ruhwinkel stimmen (Stromverhältnisbedingung).

Folgende Versuchsanordnung könnte hierzu verwendet werden, beispielsweise als Teil einer Studienarbeit:

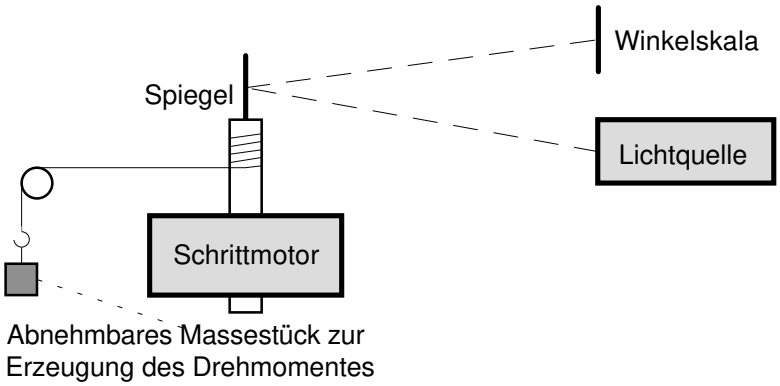


Bild 3.1: Vorschlag eines Versuchsaufbaus zur Erfassung der Strangstrom-Kurvenform

Das Ansteuerprogramm für den Versuchsaufbau könnte ein Windows-Programm mit einem Dialog als Hauptfenster sein, mit Rollbalken als Einstellorgan für die beiden Spulenströme. Zusätzlich können Rollbalken für Gesamtstrom und Stromverhältnis (ähnlich Lautstärke- und Balance-Steller an einer Stereo-Anlage anstelle von 2 Schiebestellern für linken und rechten Kanal) angeordnet werden. Die Ermittlung der korrekten Schieberstellungen muß vermutlich wechselweise erfolgen. Das Ergebnis, die neue Kurvenform, kann dann in die Programm-Quelldatei MPK3D.ASM eingebunden und diese Datei neu übersetzt werden.

3.1.3 Realisierung des Virtuellen Gerätetreibers

Der neue Treiber muß mit sehr hoher Privilegierung laufen. Dafür bieten sich unter Windows die VxD's an. Diese Programme arbeiten eng mit dem Kernel zusammen, die gemeinsam in der höchstprivilegierte Stufe des Mikroprozessors 80386, dem Ring 0, arbeiten.

Zum Treiber MPK3D existieren vergleichsweise viele Dateien. Wesentlich für den Treiber MPK3D.386 ist nur die .ASM und die .DEF-Datei. Die übrigen Dateien sind für die komfortable Erstellung mittels MAKE.EXE, die Verwendung des Treibers in anderen Programmen, wie dem Programm MOTORST.EXE vom Abschnitt 3.2, sowie für die Referenz erforderlich.

Datei	**...ist ein(e)**	**Beschreibung**
MPK3D.ASM	Quelle	Der eigentliche Quelltext
MPK3D.DEF	Quelle	Modul-Definitions-Datei für den Linker
MPK3D.MAK	Quelle	Das `makefile` für das Programm MAKE.EXE
MPK3D.OBJ	temp. Datei	Objekt-Datei als Zwischendatei beim Assemblieren mit TASM.EXE
MPK3D.MAP	temp. Datei	Map-Datei als Ausgabe beim Linken mit LINK386.EXE
MPK3D.SYM	Debughilfe	Symbol-Datei von MAP2SYM.EXE
MPK3D.386	Programm	Der Virtuelle Gerätetreiber
MPK3D.PAS	Quelle	Anwenderschnittstelle zum Gerätetreiber für Pascal-Programmierer
MPK3D.C	Quelle	Anwenderschnittstelle zum Gerätetreiber für C-Programmierer
MPK3D.H	Quelle	Headerdatei für C-Programmierer
MPK3D.TPW	Bibliothek	Unit für Pascal-Programmierer (Zielplattform Windows)
MPK3D.TPU	Bibliothek	Unit für Pascal-Programmierer (Zielplattform DOS)

MPK3D.LIB	Bibliothek	Bibliotheksdatei für C-Programmierer
MPK3D.RTF	Quelle	Quelldatei für die Windows-Hilfe-Datei „MPK3D-Referenz“
MPK3D.PRJ	Quelle	Projektdatei für die Windows-Hilfe-Erstellung
MPK3D.HLP	Hilfedatei	Windows-Hilfedatei „MPK3D-Referenz“

Tabelle 3.1: Liste der einzelnen Programm-Module zum Treiber

Um VxD's zu schreiben, benötigt man das DDK. Dieses kann man über das MSDN beziehen, bei dem man mindestens “Level 2” sein muß. Die dazu erforderliche Mitgliedschaft im MSDN ist recht kostenintensiv [MSDN]. Da diese Mittel nicht zur Verfügung standen, wurden Wege gesucht, das Ziel auch ohne dieses DDK zu erreichen.

Ausschlaggebend was das Buch [Dav], erhältlich in der hiesigen Universitätsbibliothek, dem eine Diskette mit DDK-Lite beilag. Hilfe für die ersten Schritte fand sich auch im Internet in der Newsgroup “comp.os.ms-windows.programmer.vxd”, die an der Technischen Universität Chemnitz bezogen werden kann.

Jedes VxD, das Funktionen exportiert, muß eine eineindeutige ID aufweisen. Um Doppelbelegungen sicher auszuschließen, werden diese ID's zentral von Microsoft verwaltet. Die Antragstellung erfolgt durch email an die Adresse “vxdid@microsoft.com”. Falls man noch kein Antragsformular hat, wird zunächst ein leerer Brief geschickt, worauf man sofort eine Eingangsbestätigung mit einem neuen Antragsformular für VxD-ID's erhält. Dies füllt man aus und schickt es wieder ein. Daraufhin bekommt man wiederum dieselbe Eingangsbestätigung mit einem neuen Antragsformular, wovon man sich aber nicht beirren lassen darf. Erst ca. 1..2 Wochen später bekommt man die VxD-ID mitgeteilt. So bekam der Motortreiber MPK3D.386 die ID 378Ch. Bei sehr kurzfristigen Projekten sollte man diese ID gleich zuerst besorgen.

3.1.4 Funktioneller Aufbau

Das Programm gliedert sich in 3 Abschnitte:

- Die Interruptserviceroutine mit den zugehörigen Unterprogrammen sorgt für die vom laufenden Vordergrundprogramm unabhängigen Antrieb des Motors. Aufgrund der Forderung, daß ISR's im RAM-Speicher stehen müssen, befindet sich der Code dafür im gesperrten Segment (`Locked_Code_Seg`). Das verhindert die Auslagerung dieses Bereiches auf die Festplatte. Hierzu gehört auch die Routine, die Zugriffe durch Anwenderprogramme auf die Ein-/Ausgabeports der *isel*-MPK3-Karte abfängt.
- Die Anwenderschnittstelle mit dem Funktionsverteiler sorgt für die Kommunikationsmöglichkeit zwischen dem VxD's und dem Anwenderprogramm. Dieser Code liegt im normalen Segment (`Code_Seg`); den Bereich darf Windows bei Nichtbenutzung auslagern.
- Der Initialisierungsteil installiert mit Hilfe von Standard-Systemfunktionen die ISR sowie die Trapserviceroutine für die Ein-/Ausgabeports.

Dieser Treiber benutzt die CMOS-Echtzeituhr als Interruptquelle. Die zugehörige Interruptanfrage-Leitung ist IRQ8. Um Kompatibilitätsschwierigkeiten zu umgehen und den Rechner nicht unnötig mit Interrupts zu belasten, wurde die Frequenz auf 1024 Hz festgelegt. Diese Frequenz wird beispielsweise vom BIOS erwartet, wenn die AT-Zeitgeber-Funktionen des Int 1Ah benutzt werden [PCTIM].

Die Frequenz liegt erheblich niedriger als beim Originaltreiber, wo sie 10 kHz betrug. Um dennoch hohe Verfahrgeschwindigkeiten des Schrittmotors zu erreichen, werden pro Interrupt („Tick“) gegebenfalls mehrere Mikroschritte bis hinauf zu einem Ganzschritt ausgeführt.

Der Treiber unterstützt die gleichzeitige Ansteuerung von allen 3 Motoren der Karte. Alle Prozeduren erwarten daher in EBX einen Zeiger auf die zu bearbeitende Motor-Struktur. Das entspricht objektorientierter Programmierung - allerdings ohne virtuelle Methoden, da es weder Objekte abzuleiten noch Code wiederzuverwenden gibt.

Von besonderer Schwierigkeit bei der Implementation des Treibers erwies sich die Realisierung eines korrekten Abbremsvorganges. Die Geschwindigkeit und der Restweg können unpassend zueinander sein, um genauso abzustoppen wie zu bescheunigen. Es sind folgende Bedingungen einzuhalten:

- Die Höchstgeschwindigkeit, d.h. die maximale Anzahl Mikroschritte pro Tick, darf nicht überschritten werden (Höchstgeschwindigkeitskriterium).
- Die Momentangeschwindigkeiten, d.h. die momentane Anzahl Mikroschritte pro Tick, zwischen zwei benachbarten Ticks dürfen sich höchstens um die Maximalbeschleunigung unterscheiden (Maximal-beschleunigungskriterium).

- Beschleunigung, Bewegung und das Abbremsen sollen maximal zügig vonstatten gehen (Optimierungskriterium).
- Der zurückgelegte Weg muß beim Anhalten stimmen; keinesfalls darf über das Ziel hinausgegangen werden. Zu kurze Wege können durch nachlaufende Ticks mit geringer Geschwindigkeit egalisiert werden; mehr als 1 solcher Tick verletzt jedoch das Optimierungskriterium.

Die folgenden Bilder sollen den Sachverhalt bei Darstellung optimaler Rampen visuell verdeutlichen.

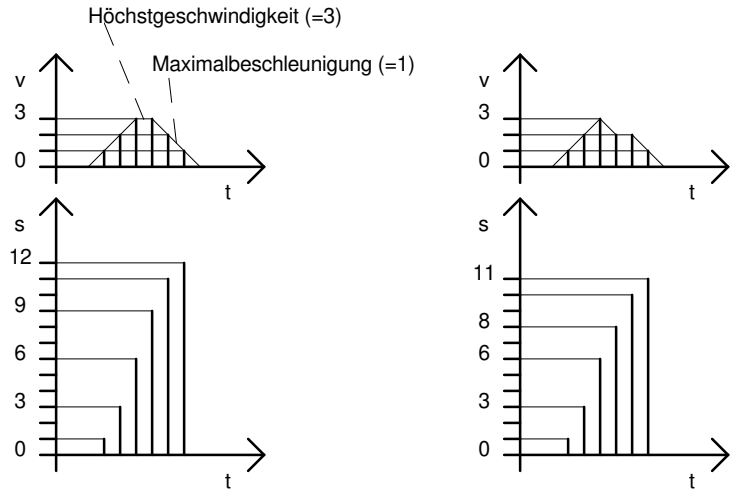


Bild 3.2: Beschleunigungs- und Abbremsrampen für s=12 und s=11

Angenommen wird eine Maximalgeschwindigkeit von 3 Mikroschritt pro Tick und eine Maximalbeschleunigung von 1 [Mikroschritt pro Tick] pro Tick. Wie die Bilder verdeutlichen, gibt es selbst für dieses eine Beispiel allein 3 Fälle, wenn der Verfahrweg mindestens 9 Mikroschritt lang ist, um die Maximalgeschwindigkeit zu erreichen.

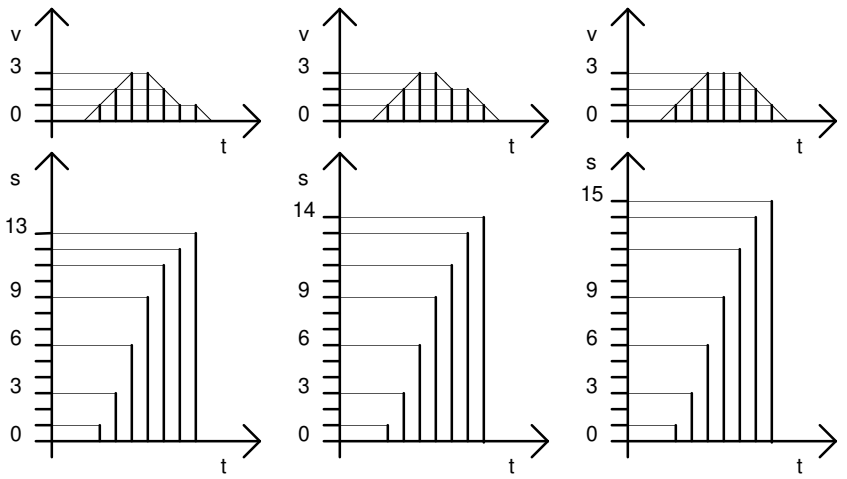


Bild 3.3: Beschleunigungs- und Abbremsrampen für s=13, s=14 und s=15

Es kommt darauf an, die richtige Stelle des Einschiebens eines „Verzögerer-Schritts“ zu finden.

Noch komplizierter wird es, wenn die Maximalbeschleunigung größer ist als ein Schritt, oder wenn gebrochene Bescheunigungen und Geschwindigkeiten erforderlich sind.

Deshalb wurde in MPK3D ein anderer Weg beschritten: Zunächst wird ein Mikroschritt intern in 65536 Nanoschritte aufgeteilt und damit gerechnet. Damit ist sichergestellt, daß keine gebrochenen Zahlen auftreten und die Einstellung von Geschwindigkeit und Beschleunigung fein genug erfolgen können. Beim Start eines Wegsegments wird die Länge in einen vorzeichenlosen 64-bit-Restweg-Speicher in Nanoschritte umgerechnet abgelegt. Effektiv werden von diesem 64-bit-Speicher nur 48 bit benötigt; die höchstwertigen 16 Bits sollten immer Null sein. Mit jedem Schritt erfolgt eine Prüfung, ob der vorhandene Restweg zum Beschleunigen oder – bei erreichter Maximalgeschwindigkeit – zum Halten der momentanen Geschwindigkeit ausreicht. Wenn nicht, wird abgebremst. Kernstück ist eine Routine, die in Abhängigkeit einer angenommenen Geschwindigkeit für den nächsten Schritt den minimalen Bremsweg berechnet. Dieses Ergebnis wird mit dem Restweg verglichen und – falls kleiner – kann der nächste Schritt mit der angenommenen Geschwindigkeit erfolgen.

Der minimale numerische Bremsweg s ergibt sich aus der Geschwindigkeit v und der Beschleunigung $a_{\max}$ zu:

$$d = a_{\max} \qquad n = \mathrm{int}\left(\frac{v}{d}\right)+1 \qquad s = n(n-1)\frac{d}{2}$$

Dabei ist d der Abstand einer arithmetischen Reihe, n die Anzahl der Glieder dieser Reihe, und s ergibt sich aus der Summenformel der arithmetischen Reihe. Von dieser Reihe ist das letzte Element (v) und die Differenz zweier Elemente (d) gegeben und die Summe (s) gesucht. Als erforderlicher Zwischenschritt wird n, die Anzahl der Glieder, derart berechnet, daß alle Glieder der Reihe positiv sind.

Alternativ läßt sich der Bremsweg über eine Schleife summieren.

C-Syntax:

```
for (s=0; v>0; v-=a_max) s+=v;
```

Pascal-Syntax:

```
s:=0; while v>0 do begin s:=s+v; v:=v-a_max end;
```

Diese Schleife „simuliert“ den Bremsvorgang so lange, bis die Geschwindigkeit Null erreicht oder negativ wird. Dabei wird mit jeder Schleifenrunde der Bremswegzähler um die sich erniedrigende Geschwindigkeit erhöht. Das Ergebnis s erfüllt folgende Ungleichung:

$$s \le \frac{v^2}{2a_{\max}}$$

Der numerische Bremsweg ist kleiner oder gleich dem rechnerischen Bremsweg auf der rechten Seite der Ungleichung. Der Fehlbetrag ist kleiner als die Beschleunigung pro Tick und kann an das Bewegungssegment angehangen werden.

Beide Varianten der Bremswegberechnung wurden in Assembler implementiert und liefern – voilà – dasselbe Ergebnis. Per bedingter Assemblierung kann mit dem Symbol `UseLoop` eine der beiden Routinen ausgewählt werden. Im Interesse definierbarer Abarbeitungszeiten ist die Formel-Variante vorzuziehen, da ansonsten der Kernel undefiniert lange blockieren könnte.

Mit jedem Tick wird ein neuer Geschwindigkeitswert nach dem Programmablaufplan in Bild 3.4 unter Zuhilfenahme der o. g. Bremswegberechnungsfunktion ermittelt. Anschließend wird dieser Geschwindigkeitswert vom Restweg abgezogen sowie entsprechend der Bewegungsrichtung vorzeichenrichtig zur Momentanposition addiert. Die sich dabei ergebende vorzeichenbehaftete Differenz in den Bitpositionen 47..16 wird der Prozedur `motor_move` übergeben, die dann den Motor um diesen Wert dreht.

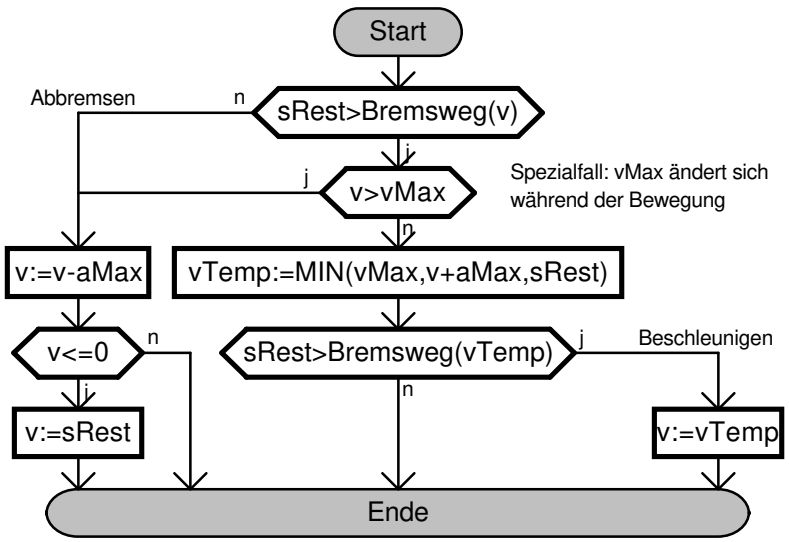


Bild 3.4: Programmablaufplan „Berechnung nächster Schritt“

Diese komplizierte Verfahrensweise ermöglicht es, sehr hohe Positionierfeinheiten bei niedrigen Interruptfrequenzen und hohen Verfahrgeschwindigkeiten zu erreichen. Zudem ist die Geschwindigkeit und die Beschleunigung durch das angewandte Nanoschrittverfahren feinfühlig einstellbar, sodaß dieses Prinzip für alle erdenklichen Anwendungen mit Einzelschrittmotoren, insbesondere auch in der Leistungselektronik, einsetzbar ist.

Wegen der Trapezform des Bewegungssegments mit linearen Rampen wechseln Kräfte auf bewegte Massen sprunghaft. Werden Masse-Feder-Systeme bewegt, kann es dadurch zu unerwünschten Einschwingvorgängen kommen. In einem solchen Fall müßten die Trapezecken abgerundet werden; im Idealfall müssen die Flanken wie eine cos²-Funktion aussehen.

3.1.5 Bereitgestellte Schnittstellenfunktionen und Installation

Eine Liste der vom Treiber zur Verfügung gestellten Funktionen mit ausführlicher Beschreibung befindet sich in der Hilfedatei zum Treiber MPK3D.HLP. Im wesentlichen wurde versucht, die im Originaltreiber vorhandenen Schnittstellen funktionell nachzubilden. Neu hinzugekommen sind die Funktionen zum Belegen und Freigeben des Motors, um im Multitaskingsystem Windows eine Mehrfachzuordnung zu verhindern.

Der Aufruf der Schnittstelle erfolgt nicht wie im Original über einen Interrupt, sondern über eine FAR-Adresse, die man sich über den Interrupt 2Fh Unterfunktion AX=1684h mit BX=378Ch besorgt. Ein Programmbeispiel dazu befindet sich ebenfalls in o. g. Hilfedatei. Dieser Aufrufweg ist unter Windows 3.x genormt. Sämtliche VxD's mit Anwenderschnittstelle werden auf diese Weise gerufen. Unterschiedlich ist nur die Belegung des Registers BX, das die ID des gewünschten VxD enthält.

Die Installation des Treibers erfolgt durch Kopieren der Datei MPK3D.386 in das Windows-Verzeichnis und dem Eintrag der folgenden Zeile in die [386Enh]-Sektion der SYSTEM.INI:

```
device=mpk3d.386
```

Der Treiber wird erst beim Neustart von Windows wirksam. In der Referenz MPK3D.HLP befinden sich weiterführende Installationshinweise.

3.2 DDE-Server für Positioniermotor

In diesem Abschnitt wird die Implementierung der Software zur Ansteuer-Oberfläche des Motors beschrieben. Der Name MOTORST.EXE ergibt sich von „Motorsteuerung“. Die Bedienfunktionen des Vorgängerprogramms sollten übernommen und nach Möglichkeit verbessert werden. Das Programm soll vor allem als DDE-Server dienen, jedoch sollen grundlegende Bewegungsfunktionen auch direkt ausgeführt werden können.

Erforderlich für die korrekte Funktion dieses Programms ist ein installierter Schrittmotortreiber MPK3D.386 vom vorhergehenden Abschnitt 3.1. Die Installation dieses Treibers ist sowohl ebenda als auch in der Referenz (Windows-Hilfe-Datei MPK3D.HLP) zu diesem Treiber beschrieben. Das Programm ist auch ohne diesen Treiber lauffähig; es sind dann jedoch fast alle Dialogelemente außer Funktion, sodaß nur wenige Programmfunktionen ausführbar sind.

Das Programm wurde als Windows-Anwendung unter Borland Pascal für DOS entwickelt. Die DOS-Programmierplattform bietet Geschwindigkeitsvorteile, insbesondere beim Zugriff auf die integrierte Hilfe. Zu beachten ist, daß die Quelltexte im OEM-Zeichensatz vorliegen und ggf. unter Zuhilfenahme des Werkzeugs FCONVERT.EXE in den Windows-Zeichensatz konvertiert werden müssen.

Das neue Programm wurde modular aus verschiedenen Quelltext-Teilen unter Verwendung von Borland-Pascal-Units aufgebaut. Entsprechend den Beispielen von Microsoft ist es unter Windows eine oft zu beobachtende Sitte, jeder Dialogbox eine extra Datei (in C sind es zwei: eine .C und eine .H-Datei pro Dialog) zuzuordnen. Dies wurde auch hier praktiziert. Dabei sind die Programm-Module in einer im Bild 3.5 gezeigten Weise voneinander abhängig.

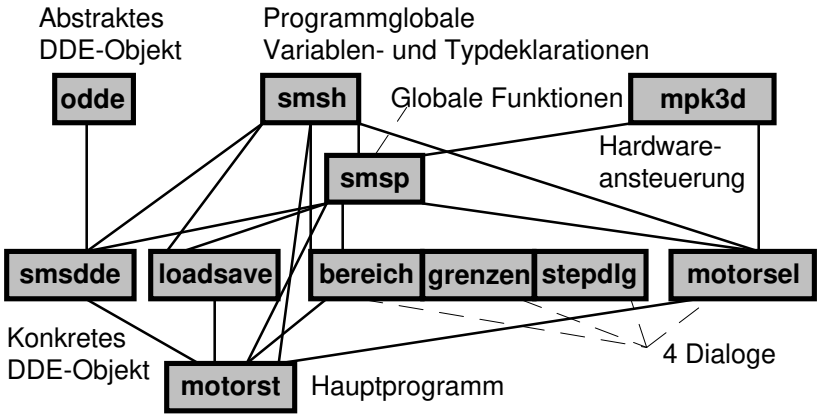


Bild 3.5: Graf der Abhängigkeiten der Programm-Module

Alle Programmteile beziehen sich auf eine globale Variablen- und Typensammlung **smsh** sowie Prozedursammlung **smsp**. Für das Ansprechen des Treibers wurde die zum Treiber gehörende Unit **mpk3d** eingebunden. Die Unit **loadsave** beinhaltet die erforderlichen Routinen zum Laden und Speichern der Einstellungswerte von bzw. zur WASSERMP.INI-Datei. Bis hierhin gibt es keine Objekte.

Des weiteren wurde die in einer objektorientierten Hierarchie umständlich zu handhabende DDEML-Callback-Funktion in ein abstraktes DDE-Objekt namens **odde** „verpackt“, die für jeden Callback-Typ eine virtuelle Methode aufruft.

Das Hauptprogramm selbst ist ein Dialogfenster. Der Vorteil liegt in der einfachen Anordnung der Dialogelemente mit dem Resource Workshop von Borland. Das Hauptfenster ist in der Objekthierarchie ein Nachkomme von TDlgWindow. Nur das ermöglicht, daß diesem Fenster auch ein Symbol (Icon) zugeordnet werden kann, da ein Dialogfenster in Windows eine globale Klasse (mit `TWndClass.hIcon=0`) ist, und das Ändern des Klassen-Icons kann andere Programme beeinflussen. TDlgWindow verbindet den Dialog mit einer neuen Klasse, wie dies in der SDK-Dokumentation angegeben ist, und kaschiert Differenzen zwischen Fenstern und Dialogen.

Das Basisobjekt TDlgWindow verhält sich in zwei Punkten nicht so wie ein Fenster. Das Programm WINHELP.EXE wertet mehrere solcher laufenden Programme wie eins aus, mit der Konsequenz, daß nur genau eins der Programme die Hilfe mittels der Windows-Funktion `WinHelp()` anzeigen kann. Den jeweils anderen Programm, das dieses Basisobjekt verwendet, wird die Hilfe in demselben Moment entzogen. Dieser Nachteil ist verkraftbar und fällt in praxi nur bei intensiver Verwendung solcher Programme auf.

Der andere Unterschied betrifft die Art und Weise der Rückgabe von Ergebniswerten bei anwenderdefinierten Fensternachrichten. Während bei normalen Fenstern das Setzen von `Msg.Result` genügt, muß bei Nachkommen von TDialog und TDlgWindow der Rückgabeparameter mittels

```
SetWindowLong(HWindow,DWL_MsgResult,Msg.Result);
```

in ein Feld der Dialogfensterklasse gesetzt werden, damit das aufrufende Programm bei `SendMessage()` einen Rückgabewert erhält.

Das Programm verwendet eine im Bild 3.6 aufgezeigte relativ einfache Objekthierarchie, die ausgehend von vordefinierten, allgemein verwendbaren Objekten nur eine Generation Nachkommen verwendet.

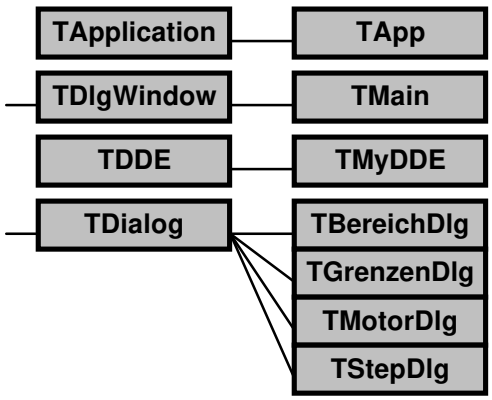


Bild 3.6: Vererbungs-Hierarchie der Objekte in MOTORST.EXE

Dabei beinhaltet TMain für das Hauptfenster den größten Anteil Code. Von hier werden die einzelnen Dialoge geöffnet sowie Dialogelemente und Menüpunkte bedient. Die Steuerung des Schrittmotors geht ebenfalls von diesem Fenster aus.

3.2.1 Ansteuerung des Gerätetreibers

Das Programm verwendet die im Abschnitt 3.1.5 erläuterte Schnittstelle. Ein Großteil der darüber angebotenen Funktionen wird benutzt, beispielsweise das Setzen des Getriebefaktors und die Absolutbewegung. Der Getriebefaktor wird derart gesetzt, daß jedes Winkelgrad 1000 Schritten entspricht. Da es nötig ist, mit Gleitkommazahlen zu hantieren, der Treiber jedoch nur Ganzzahlen akzeptiert, verhüllt eine häufig benutzte Funktion `VxDSupR()` in der Unit **smsp** dieses Problem und übergibt die Gleitkommazahl mit dem Faktor 1000 multipliziert an den Treiber. Die doppelte Umsetzung wäre für dieses Programm nicht nötig gewesen, ihre Benutzung hat jedoch bei der Fehlersuche noch einige Fehler des Motortreibers entdecken lassen.

3.2.2 Oberfläche

Das Programm bietet eine SAA-konforme Oberfläche in Form eines Fensters mit verschiedenen Dialogelementen. Das Bild 3.7 zeigt einen „Schnappschuß“ des Fensters mit verschiedenen Positionsangaben.

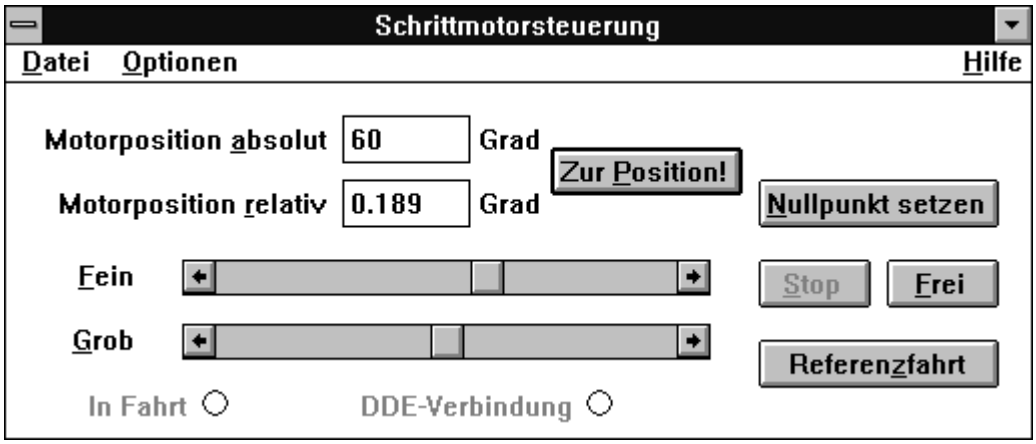


Bild 3.7: Hauptfenster des Programms MOTORST.EXE

Das Fenster zeigt auf einen Blick sämtliche relevanten Angaben. Die Eingabefelder zeigen die Position digital, die Rollbalken grafisch an. Die beiden Anzeigen am unteren Fensterrand geben Auskunft über wichtige Programmzustände. Die Schalter des Hauptfensters sind allgegenwärtig und leicht zu erreichen. So kann der Motor bequem mit Maus oder Tastatur „ziehen“ oder stoppen, und der Stop-Schalter ist gut ereichbar, wenn es notwendig wird. Alle Schalter deaktivieren sich, wenn ihr Betätigen keine Wirkung hätte. Eine detaillierte Funktionsbeschreibung zu jedem einzelnen Dialogelement befindet sich in der zugehörigen Hilfe.

Das Programm läßt sich sowohl vollständig mit der Tastatur oder mit Einschränkungen mit der Maus bedienen. Entsprechend der Aufgabenstellung, die Funktionalität des Originalprogramms beizubehalten, kann die Motorposititon relativ und absolut angegeben werden. Zu beachten ist, daß die Eingabe einer neuen Motorposition erst zur Wirkung kommt, wenn der voreingestellte Schalter Zur Position rechts daneben betätigt wird. Am Beginn der Programmentwicklung wurde versucht, ohne diesen Knopf auszukommen, jedoch verblieben Probleme, zu welchem Zeitpunkt der eingegebene Zahlenwert gültig sein soll, um die Bewegung zu starten. Die jetzige Lösung erlaubt es, einen eingegebenen Zahlenwert durch die Entertaste zu bestätigen. Dabei wird dasjenige Eingabefeld herangezogen, welches zuletzt fokussiert war. War keines der Felder fokussiert und der Anwender drückt den Knopf, „weiß“ das Programm nicht, welchen Wert es nehmen soll, und zeigt eine Fehlermeldung. Somit macht dieser Knopf eine Ausnahme und deaktiviert sich in diesem Falle nicht.

Das Menü weist nur wenige Unterpunkte auf und wurde in Anlehnung an übliche Windows-Programme gestaltet. So beinhaltet das Datei-Menü ausschließlich den Unterpunkt Beenden, da eigentliche Datei-Operationen von diesem Programm nicht unterstützt werden. Auf diese Weise läßt sich MOTORST.EXE mit der Tastenkombination ALT+D B beenden.

Der Menüpunkt Optionen weist einige Unterpunkte auf. Zum einen kann mit Bereich abfahren ein Winkelbereich in einstellbarer Schrittzahl „abgeschreitet“, d.h. schrittweise abgefahren werden. Mit Motor auswählen wird eines der drei Systeme der *isel*-MPK3-Karte selektiert. Dazu wird auf den geladenen Treiber MPK3D.386 zurückgegriffen. Der Unterpunkt Grenzen festlegen dient der Einstellung einiger seltener einzustellender Werte, wie z.B. Geschwindigkeit und Beschleunigung der Motorachse. Die Dialoge finden im Abschnitt 3.2.3 nähere Beachtung. Schließlich sichert Einstellungen speichern die momentan eingestellten Werte in der Datei WASSERMP.INI, womit diese für den nächsten Programmstart von MOTORST.EXE als Startwerte dienen. Diese Datei befindet sich im Windows-Verzeichnis und wird gemeinsam mit dem Programm SENDERST.EXE (siehe Abschnitt 3.3) verwendet.

Zum Programm gehört eine Windows-Hilfedatei MOTORST.HLP, in der sowohl die Bedienung des Programms in Verbindung mit der Hardware als auch die Verwendung des Programms als DDE-Server beschrieben ist. Bilder mit anklickbaren Bereichen erleichtern es, die richtige Hilfeseite zu jedem Dialogelement zu erhalten und ersparen es, das Programm MOTORST.EXE an jeder Stelle mit kontextsensitiver Hilfe auszustatten. Diese Bereiche sind auch mit der Tabulator-Taste selektierbar und sichern die vollständige Bedienbarkeit der Hilfe mit der Tastatur.

Dieses Programm unterstützt einen Kommandozeilenschalter „/debug“, mit dem das Programm zur vollständigen Funktion mit MPK3D.386, jedoch ohne Motor und Referenztaster gezwungen werden kann. Die Referenzfahrt wird dabei so durchgeführt, als ob der Endschalter sofort anspricht. Die normalen Bewegungsfunktionen reagieren zeitlich ebenso, als wäre ein Motor angeschlossen und zu bewegen.

3.2.3 Dialoge

Dieses Programm bietet 3 Dialoge an, mit denen Aktionen gestartet oder Einstellungen vorgenommen werden. Sie befinden sich unter dem Menüpunkt Optionen. Alle Dialoge bieten Hilfe über je einen Hilfe-Knopf an.

Dialog „Bereich abschreiten“

Gemäß einer Vereinbarung über die Präzisierung der Aufgabenstellung mit dem Betreuer wurde dieser Dialog implmentiert. Hiermit ist es bei einem halbautomatischen Meßplatz möglich, einen Bereich schrittweise abzufahren.

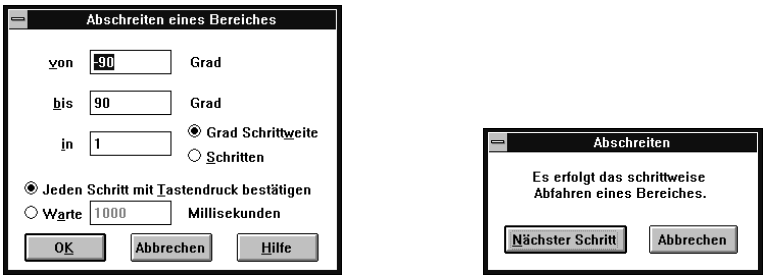


Bild 3.8: Dialogfenster zum Abfahren eines Bereiches sowie Fortsetzungsfenster

Wie aus den angegebenen Dialogelementen ersichtlich, können alle Parameter des abzufahrenden Bereiches eingestellt werden. Das Fortsetzen mit dem nächsten Schritt kann sowohl per Hand als auch zeitgesteuert automatisch erfolgen. Die Angabe der Wartezeit ist eine maximale, da während des Wartens ebenso Tastendrücke auf den Knopf Nächster Schritt zur Weiterschaltung verwendet werden. Die Zeitangabe ist nicht so genau zu nehmen, da der Windows-Zeitgeber etwa 55 ms Auflösung hat, und andere den Prozessor blockierende Anwendungen dazu führen, daß dieses Programm die Zeitgeber-Nachrichten verzögert erhält.

Dialoge „Motor auswählen“ und „Grenzen festlegen“

Diese beiden Dialoge dienen der Konfiguration dieses Progamms. Mit dem einen wird einer der 3 verfügbaren MPK3D-Treiber-Motor-Einheiten ausgewählt. Am Wassermeßplatz ist der Schrittmotor als Einheit Z geschaltet.

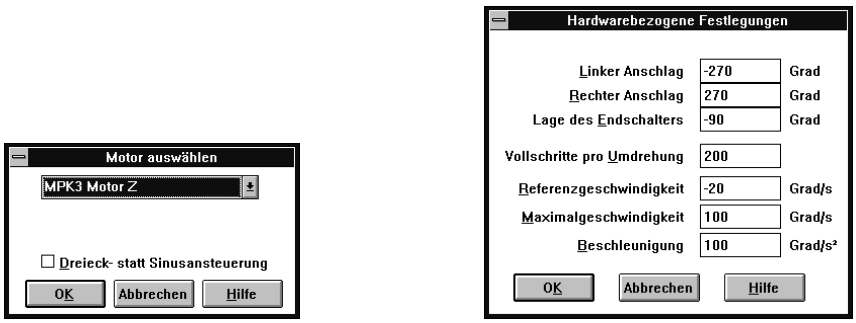


Bild 3.9: Dialogfenster zur Motor-Auswahl und zum Konfigurieren

Die Dialogfenster im Bild 3.9 zeigen dabei standardmäßige Werte für das Arbeiten am Meßplatz in der aktuellen Konfiguration der Hardware an. Detaillierte Informationen zur Einstellung jedes einzelnen Dialogelementes befindet sich in der zugehörigen Hilfedatei, deren Konsultation bei der Arbeit mit diesen Dialogen dringend angeraten wird. Die Parameter sind im Interesse universeller Verwendbarkeit dieses Programms sehr freizügig einstellbar. So ist es möglich, als Maximalgeschwindigkeit 5 Umdrehungen pro Sekunde zu setzen, aber auch wenige Bogenminuten pro Sekunde. Bei offensichtlichen Fehlern oder Inkonsistenzen erfolgt eine Fehlermeldung beim Versuch, den Dialog mit OK zu beenden. Die fehlerhafte Stelle wird hervorgehoben.

Die weiten einstellbaren Bereiche sollen die universelle Einsatzfähigkeit für später sichern. Um eine völlig außer Kontrolle geratene Konfiguration wiederherzustellen, sollte die gesamte Sektion [MOTORST] in der Initialisierungsdatei WASSERMP.INI, die sich im Windows-Verzeichnis befindet, mit einem Texteditor gelöscht und das Programm neu gestartet werden. Das Programm verwendet dann intern gespeicherte „mittlere“ Werte, die zu einer normalen Funktion führen. Gegebenenfalls sind noch Anpassungen, wie z. B. die Veränderung der Lage des Referenztasters, nötig. Alternativ lassen sich die vorgeschlagenen Werte aus der Hilfe übernehmen.

Die eingestellten Werte werden nicht automatisch gespeichert, sondern durch Anwahl des Menüpunktes Optionen/Einstellungen speichern. Die Daten werden textuell in die Windows-Initialisierungsdatei WASSERMP.INI unter die Sektion [MOTORST] eingetragen und sind für den nächsten Programmstart Vorgabewerte.

3.2.4 DDE-Kommunikation

MOTORST.EXE ist ein DDE-Server unter Benutzung der DDEML. Er ist in der Lage, Datenabfragen zu beantworten *(REQUEST)*, Datenveränderungen dem Client mitzuteilen *(ADVISE)*, neue Daten entgegenzunehmen *(POKE)* sowie Befehle auszuführen *(EXECUTE)*.

Folgende Zeichenkettenkonstanten müssen für die Konversation mit MOTORST verwendet werden:

Service-Name ..... **MOTORST** (service)
- System-Thema..... **System** (standard topic)
  - System-Datenelemente..... **SysItems** (data item)
  - Themen-Liste..... **Topics** (data item)
  - Formate-Liste..... **Formats** (data item)
  - Item-Liste..... **TopicItemList** (data item)
  - Hilfe..... **Help** (data item)
- Themen-Name..... **Motor** (topic)
- Themen-Alias..... **MotorX**, **MotorY** oder **MotorZ** (topic) je nach Auswahl
  - Position..... **Position** (data item) enthält die Relativposition
  - In-Bewegung-Flag..... **InMove** (data item) enthält 0 (stehend) oder 1 (in Bewegung)
  - Synchron-Flag..... **Synced** (data item) enthält 0 oder 1 (Referenzfahrt ausgeführt)
  - Initialisierung!..... **Init** (execute command): Gesamte Einrichtung initialisieren
  - Referenzfahrt!..... **Sync** (execute command): Referenzfahrt durchführen
  - Bewegen!..... **Move xxx** (execute command): Bewegen nach **xxx**
  - Anhalten!..... **Stop** (execute command): Motor stoppen und halten
  - Freigeben!..... **Free** (execute command): Motor stromlos schalten

Der Datenaustausch erfolgt auf Textbasis *(CF_TEXT)*, d.h. es werden Zeichenketten übertragen, die die Zahlen repräsentieren. Dies ist gängigste Art, geringe Datenmengen auszutauschen.

Es steht ein **System**-Thema zur Verfügung, welches über die DDE-Möglichkeiten dieses Programms informiert.

MOTORST unterstützt die automatische Verbindungssuche *(WILDCONNECT)* nicht.

Erläuterungen zum Motor-Thema

Das wesentliche Thema dieses DDE-Servers ist das Motor-Thema. Es enthält 3 Datenelemente und stellt 5 Execute-Funktionen zur Verfügung. Alle Datenelemente sind “advisable”, können also Änderungen automatisch dem Client mitteilen. Das Datenelement **Position** kann außerdem noch verändert werden (POKE), die beiden zweiwertigen Datenelemente **InMove** und **Synced** sind nur lesbar.

Bei diesem DDE-Server ist die *EXECUTE*-Funktionalität besonders stark ausgebaut. Nicht jeder Client unterstützt DdeExec. Neben der Möglichkeit, den Motor sowohl über *EXECUTE* als auch über *POKE* (auf Item **Position**) zu bewegen, bietet hier DdeExec ausschließlich die übrigen Steuermöglichkeiten der Referenzfahrt (Kommando **Sync**), dem Soforthalt (**Stop**) und des Freischaltens (**Free**) der Motorachse an. Übrigens liefern alle Execute-Funktionen eine negative DDE-Bestätigung, wenn kein gültiger Motor ausgewählt wurde.

Position:

Eine Veränderung der **Position** zieht immer eine Bewegung des Motors nach sich; das Setzen der **Position** ist demnach gleichbedeutend mit dem Execute-Kommando **Move xxx**. Der DDE-Server bezieht sich hierbei ausschließlich auf die Relativposition, somit hat der Anwender die Möglichkeit, die tatsächliche Nullage vor dem DDE-Clienten zu verbergen. Dagegen sind die Angaben selbst absolut zum Nullpunkt und in Grad zu verstehen, d.h. ein Akkumulieren ist nicht möglich. Das vermeidet ein Zunehmen systematischer Fehler. Eine negative DDE-Bestätigung wird geliefert, wenn die Zielposition außerhalb festgelegter Grenzen liegt, oder der Motor freigeschaltet war (**Synced**=0).

Erfolgt momentan eine Bewegung (**InMove**=1), ist das kein Fehler - die neue Position wird in eine 1 Element lange Warteschlange gestellt. Diese neue Position wird dann automatisch beim Beenden des laufenden Bewegungssegmentes angefahren. Ist die Warteschlange bereits gefüllt, wird der Wert dort überschrieben, das bedeutet, daß die dort stehende Position wird nicht mehr angefahren wird.

Das Programm unterstützt die automatische Aktualisierung. Wenn ein Client die automatische Aktualisierung aktiviert *(ADVSTART)*, werden bei jeder Wertänderung, sowohl von anderen Clients als auch direkt bei Manipulation an den Dialogelementen des Programms, die veränderten Daten aktualisiert.

InMove:

Das Datenelement **InMove** zeigt den Bewegungsstatus des Motors an und kann von einem DDE-Client sowohl durch zyklische Abfrage (Polling) als auch über die automatische Aktualisierung *(ADVSTART)* ausgewertet und für einen Programmfluß einer DDE unterstützenden Skriptsprache herangezogen werden. Dieses Datenelement ist ebenfalls "advisable", jedoch nicht mit *POKE* veränderbar.

Synced:

Das Datenelement **Synced** informiert darüber, ob eine erfolgreiche Referenzfahrt ausgeführt wurde (Synced=1), oder ob eine Referenzfahrt erforderlich ist (Synced=0), nachdem z.B. die Funktion **Free** aufgerufen wurde. Dieses Datenelement ist ebenfalls "advisable" und nicht mit *POKE* veränderbar.

Init:

Diese Funktion initialisiert die Schrittmotorsteuerung komplett. Es wird der Versatz zwischen relativer und absoluter Position nullgesetzt, d.h. fortan sind beide Werte gleich. Danach wird eine Referenzfahrt in exakt derselben Folge durchgeführt wie beim Drücken auf den Schalter Referenzfahrt:

- Bewegen der Motorachse auf eine Seite zum Referenzschalter, wenn Motor bestromt ODER Meldungsfenster zeigen, wenn der Motor freigeschaltet (stromlos)
- Referenzfahrt ausführen
- Motor zur Position 0 bewegen

Im Gegensatz zu allen anderen Execute-Funktionen verbleibt die Funktion **Init** solange im Server, bis sie beendet wird. Außerdem ist Init die einzige nicht-stille Funktion, d.h. sie kann Meldungsfenster erzeugen. Da die Beantwortung dieses Fensters einige Zeit in Anspruch nehmen kann, sollte hierbei ein möglichst hohes DDE-Timeout (z.B. 1 Minute) client-seitig vorgegeben werden. **Init** liefert eine negative DDE-Bestätigung, wenn das Meldungsfenster durch *Abbrechen* geschlossen wurde oder sich ein Fehler einstellte.

Sync:

Hiermit wird eine „stille" Referenzfahrt ausgeführt. Es wird niemals ein Meldungsfenster gezeigt, und es wird nicht geprüft, ob sich der Motor auf der richtigen Seite des Schalters befindet. Auch wird im Anschluß keine Bewegung zur Position 0 durchgeführt. Ein DDE-Client muß ggf. selbst entsprechende Warnungen dem Anwender mitteilen, was die Funktion **Init** automatisch tut.

Achtung: Der Server (genauer: die DDE-*EXECUTE*-Funktion) kehrt bei Aufruf von **Sync** sofort zurück, er wartet also nicht auf das Ende der Bewegung. Das Ende der Bewegung kann durch Beobachtung der Variable **InMove** oder **Synced** abgewartet werden.

Move xxx:

Hiermit wird der Motor zu einer bestimmten Position bewegt. **xxx** ist dabei eine Gleitkommazahl im ASCII-Format. Dezimaltrenner darf Punkt oder Komma sein. Das Trennzeichen zwischen **Move** und **xxx** ist Weißraum (Leerzeichen oder Tabulator) und kann auch weggelassen werden.

Achtung: Der Server (genauer: die DDE-*EXECUTE*-Funktion) kehrt bei Aufruf von **Move xxx** sofort zurück, er wartet also keinesfalls auf das Ende der Bewegung. Die Motorbewegung wird im Hintergrund vom Virtuellen Gerätetreiber MPK3D.386 ausgeführt, der nur unmerklich Rechenleistung verbraucht. Die Aktualisierung der **Position** erfolgt erst am Bewegungsende, so auch der *ADVISE*-Zyklus. Dabei wird die Istposition angenommen; dieser Wert kann sich von dem bei **Move xxx** angegebenen um unvermeidliche Quantisierungsfehler unterscheiden.

Stop:

Dieses Kommando bewirkt einen Soforthalt. Der Motor wird weiterhin bestromt, erzeugt also ein Haltemoment. Da ein **Stop**-Kommando aus geringen Geschwindigkeiten heraus selten zu Schrittfehlern führt, bleibt **Synced**=1 gesetzt. Dennoch ist im folgenden eine Referenzfahrt angeraten. War der Motor in Bewegung, wird **InMove** selbstverständlich auf 0 gesetzt. Dann erfolgt auch ein *ADVISE*-Zyklus auf die Position; dort wird die Halteposition eingetragen.

Free:

Wie **Stop** bewirkt dieses Kommando einen Soforthalt. Die Motorspulen werden stromlos geschaltet, es wird kein Haltemoment erzeugt. Da der Verlust der Synchronisation sehr wahrscheinlich ist, wird **Synced**=0 gesetzt. Es erfolgt kein *ADVISE*-Zyklus auf die nun ungültige Motorposition.

Für das allgemeine Thema **Motor** existiert ein Alias (Bezeichner gleicher Bedeutung), der sich je nach ausgewähltem Treiber unterscheidet. Das ermöglicht sowohl die einmalige Verwendung dieses Programms mit einem beliebigen der 3 Motoren als auch die mehrfache Verwendung und Steuerung mittels DDE z.B. in einem künftigen mehrmotorigen Meßplatz.

Erläuterungen zum System-Thema

Dieses Programm stellt auch ein **System**-Thema zur Verfügung, welches über die DDE-Möglichkeiten informiert.

Alle Datenelemente können nur im Textformat *(CF_TEXT)* abgefragt werden.

Datenelemente, die Aufzählungen enthalten (wie z.B. **SysItems** die Liste aller System-Datenelemente auflistet), enthalten die einzelnen Zeichenketten durch je 1 Leerzeichen getrennt.

Das Hilfe-Datenelement **Help** enthält schließlich menschenlesbare Kurzinformationen.

3.3 DDE-Server für Wandler-Ansteuereinheit

In diesem Abschnitt wird die Implementierung der Software zur Ansteuerung des Wandlers am Wassermeßplatz unter Windows beschrieben. Der gewählte Name SENDERST.EXE ergibt sich von „Ultraschall-Sendersteuerung“. Dabei stand zur Bedingung, daß alle Bedienfunktionen des Vorgängerprogramms zu übernehmen und nach Möglichkeit zu verbessern sind. Ebenso wie die Motorsteuerung soll dieses Programm als DDE-Server dienen, jedoch Ausgabefunktionen auch von Hand durchgeführt werden können. Die Angaben für Winkel und Frequenz sind über den Wandlerparameter voneinander abhängig. Deshalb und zur Auflösung von Gleichzeitigkeitsproblemen in Verbindung mit DDE erfolgt das Editieren von Einzelwerten in extra Dialogfenstern. Der gegenseitige mathematische Zusammenhang wird im folgenden erläutert.

Abgefangen werden Fehler, die durch nicht vorhandene oder angeschlossene Hardware hervorgerufen werden: Das Programm stürzt nicht ab, sondern warnt mit einem entsprechenden Hinweisfenster, und das Programm kann fortgesetzt oder auch abgebrochen werden.

Das Programm wurde als Windows-Anwendung unter Borland Pascal für DOS entwickelt. Die in Abschnitt 3.2 gemachten Ausführungen sind auch hier zu beachten.

Das Programm wurde ebenso wie das Programm MOTORST.EXE im Abschnitt 3.2 modular aus verschiedenen Quelltext-Teilen unter Verwendung von Units aufgebaut. Das Bild 3.10 zeigt den Zusamenhang dieser Module.

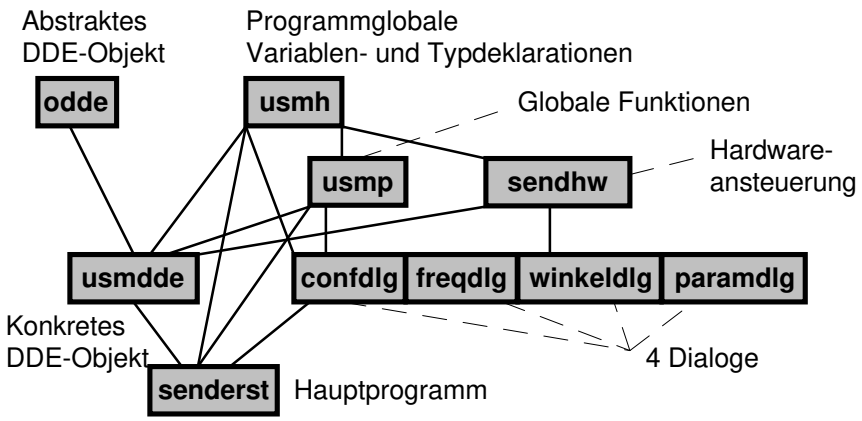


Bild 3.10: Graf der Abhängigkeiten der Programm-Module

Alle Programmteile beziehen sich auf eine globale Variablen- und Typensammlung **usmh** sowie Prozedursammlung **usmp**. Dinge, die sich auf die Hardware beziehen, befinden sich im Modul **sendhw**. Diese drei genannten Programmodule definieren bzw. benutzen keine Objekte.

Die Unit **odde** als abstraktes DDE-Objekt wurde unverändert vom Programm MOTORST übernommen.

Die weitere Gliederung entspricht ebenso dem Programm MOTORST.EXE. So ist das Hauptfenster wieder ein von TDlgWindow abgeleitetes Dialogfenster.

Das Programm verwendet eine im Bild 3.11 aufgezeigte relativ einfache Objekthierarchie, die ausgehend von vordefinierten, allgemein verwendbaren Objekten nur eine Generation Nachkommen verwendet.

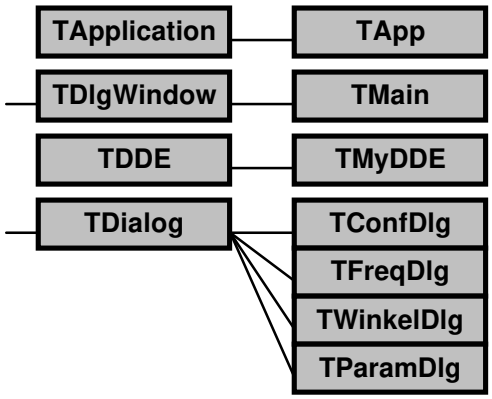


Bild 3.11: Vererbungs-Hierarchie der Objekte in SENDERST.EXE

Dabei beinhaltet TMain für das Hauptfenster den größten Anteil Code. TMain ist Startpunkt für alle Dialoge, bearbeitet Meldungen von Dialogelementen des Hauptfensters sowie Menüpunkte. Die Steuerung der Hardware und des DDE geht ebenfalls von diesem Fenster aus.

3.3.1 Ansteuerung des Gerätes

Wie im Kapitel 2.1.2 erläutert, wird das Sendersteuer-Gerät an eine parallele Schnittstelle des Rechners angeschlossen. Während das Ursprungsprogramm fest auf LPT1 programmiert wurde, ist in diesem Programm eine Auswahlmöglichkeit vorgesehen.

Die Kommunikation erfolgt mit Hilfe der Windows-Funktionen zum Ansprechen der seriellen und parallelen Schnittstellen. Die Liste aller möglichen Windows-Funktionen, die mit der seriellen oder parallelen Schnittstelle zusammenhängen, erreicht man im integrierten Hilfesystem von Borland Pascal unter „Kommunikationsfunktionen“. Für die hier benötigte Ansteuerung genügen folgende 3 der 16 Funktionen:

```
function OpenComm(ComName: PChar; InQueue, OutQueue: Word): Integer;
function TransmitCommChar(Cid: Integer; Chr: Char): Integer;
function CloseComm(Cid: Integer): Integer;
```

Diese stellen die unterste Schicht der Hardware-Ansteuerung innerhalb der Unit **sendhw** dar. Da das Sendersteuer-Gerät sich wie ein Drucker verhält, kann probeweise auch ein Drucker zur Ausgabe benutzt werden. Dieser sollte vorzugsweise im Hex-Dump-Modus arbeiten, in den die meisten Nadeldrucker beim Einschalten durch Festhalten einer bestimmten Drucker-Tastenkombination gebracht werden können.

Zur besseren Abstraktion der angeschlossenen Hardware und zum Verbergen der Funktion, die die Bits der beiden Steuerbytes zusammensetzt, wurde darauf eine weitere Schicht aufgebaut. Diese verarbeitet nur die Angaben für die Frequenz (Variable `Frequency`) und die Anzahl der abzugebenden Pulse (Variable `Pulses`). Diese Schicht besteht aus den Routinen `Actualize()` und `StopPermanent()`.

Als obere Schicht in der Unit **sendhw** stehen Funktionen zur Verfügung, die die Werte für Abstrahlwinkel bzw. Frequenz gegenseitig umrechnen. Die Ansteuerfrequenz wird in der Einheit MHz gespeichert, der Wandlerparameter (Mittenabstand der Einzelwandler) in µm, die Schallgeschwindigkeit in m/s, und der Abstrahlwinkel in Grad. Der Zusammenhang zwischen Ansteuerfrequenz f, Abstrahlwinkel α und Wandlerparameter a_p ergibt sich nach den Formeln

$$f = \frac{c_w}{a_p \sin\alpha} \qquad \text{bzw. als Umkehrung} \qquad \alpha = \arcsin\frac{c_w}{a_p f}.$$

Dabei ist c_w die konstante Schallgeschwindigkeit (hier in Wasser).

Sonderfälle und Singularitäten werden durch spezielle Zahlen repräsentiert:

- Unendlich Impulse: `Pulses` = 0

- Keine Frequenz – mithin geradlinige Abstrahlung: `Frequency` = 0
- Umkehrung des Abstrahlwinkels: Negative Frequenz-Angabe

Die Festlegung des Nullwinkels auf Frequenz = 0 ist ein Kompromiß, da die Frequenz für abnehmende Winkel nach ±∞ strebt, das Vorzeichen gilt je nach Annäherungsrichtung. Die symbolischen Gleitkommazahlen `+NAN`, `-NAN` (not-a-number, keine Zahl) sowie `+INF`, `-INF` (infinite, unendlich) werden unter Pascal nicht unterstützt und sind für Anwender ungebräuchlich.

Kurvendiskussion

Das Bild 3.12 stellt die Ansteuerfrequenzen für gewünschte Abstrahlwinkel als Kurvenschar mit dem Wandlerparameter als Kurvenparameter dar. Dieser Zusammenhang ist mit der Optik an Beugungsgittern verwandt; dort heißt der Wandlerparameter Gitterkonstante mit dem Formelzeichen *a*. f_u ist die untere Grenzfrequenz des Oszillators mit ca. 4,75 MHz, f_o die obere mit ca. 80 MHz.

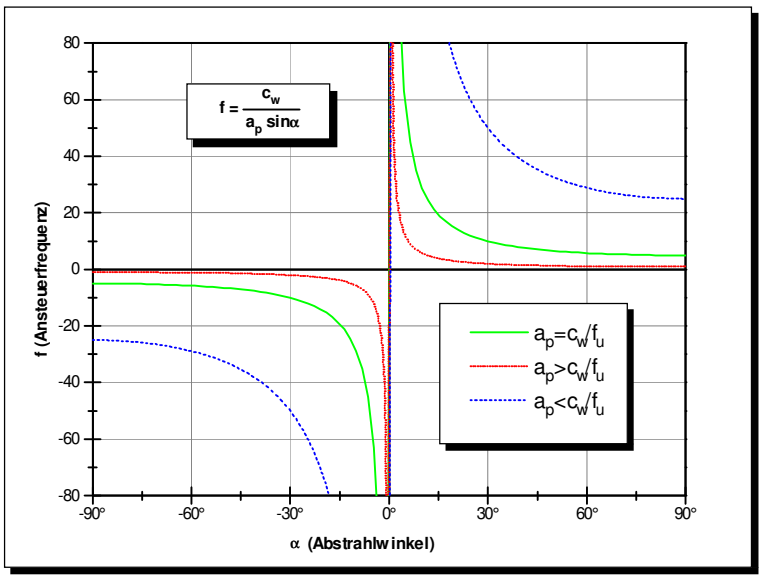


Bild 3.12: Frequenz als Funktion des Abstrahlwinkels, mit dem Wandlerparameter als Parameter

Abhängig vom Wandlerparameter ergeben sich verschiedene Fälle, bei denen entweder der der gesamte verfügbare Frequenzbereich der Sendersteuerung nicht nutzbar ist, sowie daß bestimmte Winkel nicht erreichbar sind, weil sie betragsmäßig zu groß oder zu klein sind. Die folgenden Bilder sollen 3 dieser Fälle verdeutlichen. Dabei wird die Kurvendiskussion für den I. Quadranten des Bildes 3.12 durchgeführt, für den III. Quadranten gilt aus Symmetriegründen sinngemäß dasselbe. Die Darstellung der Ordinatenachse erfolgt in den Bildern 3.13..3.15 logarithmisch.

Das Bild 3.22 zeigt den Optimalfall. Der gesamte Frequenzbereich der Sendersteuerung von ca. 4,75..80 MHz ist nutzbar, und der Winkel 90° ist dabei gerade erreichbar. Der Wandlerparameter a_p hat gerade den Wert

$$a_p = \frac{c_w}{f_u}.$$

Unter Einsatz von 1460 m/s für die Schallgeschwindigkeit in Wasser ergibt sich a_p zu 307 µm.

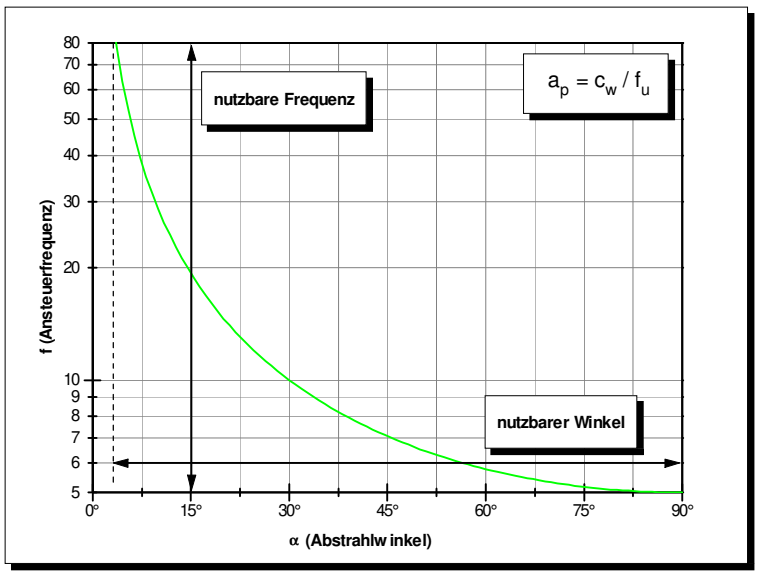


Bild 3.13: Sonderfall: Frequenz und Winkel maximal

In dem im Bild 3.14 dargestellten Fall mit 5fach gegenüber dem Optimum zu großen Wandlerparameter zeigt sich, daß man einerseits zum Nullpunkt näher herankommen kann, dies aber auf Kosten der Erreichbarkeit größerer Winkel erfolgt. Da der gesamte Frequenzbereich nutzbar ist, ist der Winkel sehr fein einstellbar.

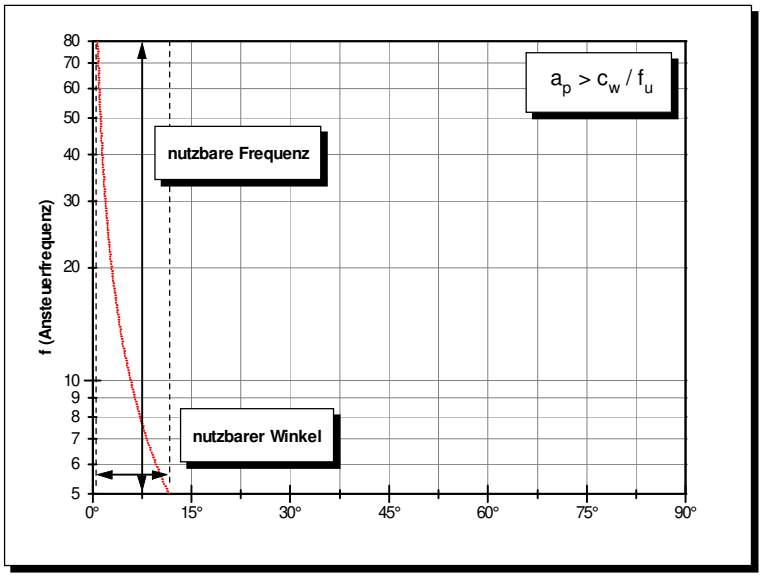


Bild 3.14: Durch den großen Wandlerparameter bleiben größere Winkel nicht nutzbar

Im Falle des Bildes 3.15 sieht es noch ungünstiger aus. Hier liegt der Wandlerparameter bei einem Fünftel des Optimalwertes vom Bild 3.13. Nicht nur, daß der wichtige Winkelbereich um den Nullpunkt nicht erreichbar ist, durch den begrenzten Frequenzbereich ist der Winkel in gröberen Schritten einstellbar. Hier hilft nur die Erhöhung der oberen Grenzfrequenz oder ein Schieberegister mit mehreren zeitversetzten Phasen.

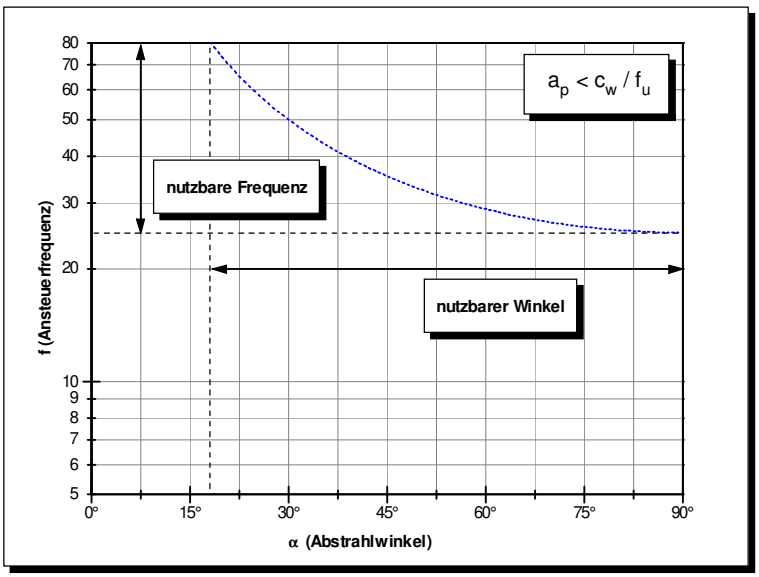


Bild 3.15: Der kleine Wandlerparameter verhindert die Abstrahlung in der Nähe von Null

Interface der Hardwaresteuerung

Die Hardwaresteuerungs-Unit **sendhw** stellt 5 Variablen öffentlich zur Verfügung, auf die jedoch nur lesend zugegriffen werden darf. Zum Schreiben (Setzen) der Werte gibt es je eine Zugriffsfunktion, die sowohl die Gültigkeit des übergebenen Wertes prüft als auch davon abhängige andere Werte setzt. Alle 5 Zugriffsfunktionen liefern ein Ergebnis vom Typ `boolean` zurück, der true ist, wenn der Setzvorgang fehlerfrei verlief, d.h. alle Prüfungen bestanden wurden.

Außerdem gibt es eine Funktion zur Auswahl einer Schnittstelle. Diese kann man ebenfalls als Zugriffsfunktion auf die Nur-Lese-Variable `Cid` interpretieren.

Ausgegeben wird mit den Funktionen dieser obersten Schicht nichts, nur ineinander umgerechnet. Die Ausgabe erfolgt mittels `Actualize()` und `StopPermanent()` aus der mittleren Schicht.

Die Unit **sendhw** bindet die Unit **usmh** ein, um den verwendeten Gleitkommatyp global festlegen zu können. Zur Zeit wird Real (6-byte-Gleitkomma) benutzt; zum Umstellen genügt eine Änderung in **usmh** und Neucompilierung. Ein einheitlicher Datentyp vermeidet unnötige Umrechnungsaktionen seitens des Compilers und hilft, die Größe der .EXE-Datei zu minimieren und das Programm schneller zu machen.

3.3.2 Oberfläche

Das Programm ist sowohl im strukturellen Aufbau als auch im Aussehen mit dem Motorsteuerprogramm vom Kapitel 3.2 ähnlich gehalten. Das Bild 3.17 zeigt einen „Schnappschuß“ des Hauptfensters.

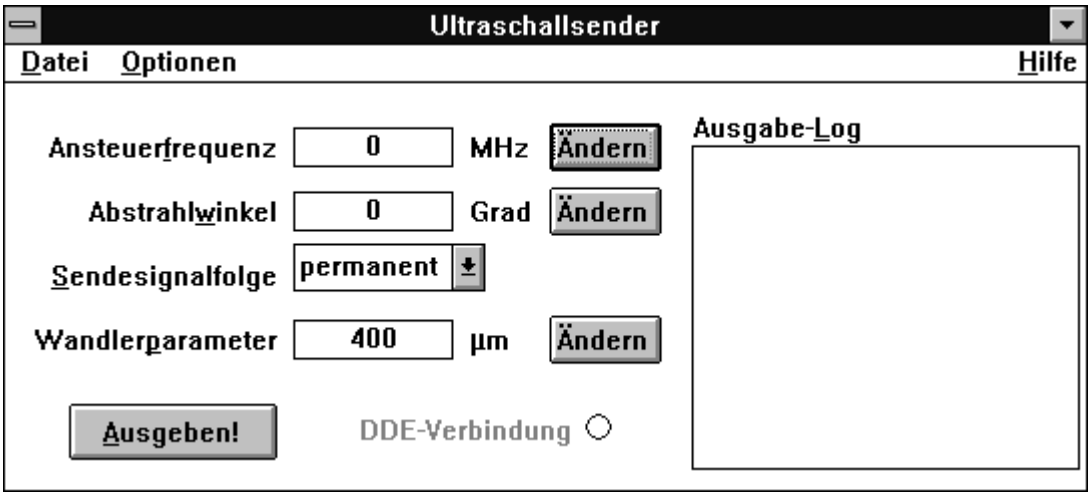


Bild 3.16: Hauptfenster des Programms SENDERST.EXE

Das Programm bietet eine SAA-konforme Oberfläche in Form eines Fensters mit verschiedenen Dialogelementen. Es läßt sich sowohl vollständig mit der Tastatur als auch mit der Maus bedienen. Alle relevanten Angaben sind auf einem Blick erfaßbar. Zu besondes ist jedoch, daß sich die angezeigten Werte für Frequenz, Winkel und Wandlerparameter nicht direkt geändert werden können, sondern daß dies durch Drücken der Taste „Ändern“ jeweils rechts daneben zu erfolgen hat. Zu Beginn der Programmentwicklung wurde versucht, die Zahlen direkt eingeben zu können, jedoch verblieben Probleme, zu welchem Zeitpunkt der eingegebene Zahlenwert gültig sein soll, um die davon abhänigen anderen Zahlenwerte zu berechnen. Insbesondere das Wechseln der aktiven Anwendung während der Eingabe führte zu unerwarteten Reaktionen, wenn die Fenster-Nachricht `WM_KillFocus` als Eingabe-Ende interpretiert werden sollte. Die jetzige Lösung mit den Extra-Dialogboxen für die Zahleneingabe ist zwar nicht so elegant, jedoch sicher in Bezug auf gewohnte Bedienbarkeit. Außerdem bieten sie einen Schutz vor „dazwischenfunkenden“ DDE-Zugriffen sowie mit den dortigen Rollbalken eine komplette Mausbedienung.

Das Menü weist nur wenige Unterpunkte auf und wurde in Anlehnung an übliche Windows-Programme gestaltet. Das Datei-Menü beherbergt ausschließlich den Unterpunkt Beenden, da eigentliche Datei-Operationen von diesem Programm nicht unterstützt werden. Auf diese Weise läßt sich SENDERST.EXE mit der Tastenkombination ALT+D B beenden.

Das Menü Optionen beinhaltet einen Menüpunkt Konfiguration auf, mit dem die Schnittstelle ausgewählt werden kann, an dem die Ultraschallsender-Hardware angeschlossen ist. Diese Einstellung wird automatisch in der Datei WASSERMP.INI im Windows-Verzeichnis für den nächsten Programmstart von SENDERST.EXE gesichert. Diese Initialisierungsdatei wird gemeinsam mit dem Programm MOTORST.EXE vom Abschnitt 3.2 verwendet.

Zu den vier Dialogfenstern dieses Programms erfolgen im Abschnitt 3.3.4 nähere Erläuterungen.

Zum Programm gehört eine Windows-Hilfedatei SENDERST.HLP, in der sowohl die Bedienung des Programms in Verbindung mit der Hardware als auch die Verwendung des Programms als DDE-Server beschrieben ist. Bilder mit anklickbaren Bereichen erleichtern es, die richtige Hilfeseite zu jedem Dialogelement zu erhalten und ersparen es, das Programm an jeder Stelle mit kontextsensitiver Hilfe auszustatten. Diese Bereiche sind auch mit der Tabulator-Taste selektierbar und ermöglichen die vollständige Bedienbarkeit der Hilfe mit der Tastatur.

Dieses Programm akzeptiert einen Kommandozeilenschalter „/debug“. Hiermit können alle Funktionen auch ohne angeschlossenem Sendersteuer-Gerät bzw. einem Drucker ausgeführt werden.

3.3.3 Log-Anzeige

Die Funktion der Log-Anzeige wurde vom Vorgängerprogramm nahezu unverändert übernommen. Unter Windows ist sie am leichtesten mit dem Windows-Stilelement LISTBOX realisierbar. Angezeigt werden sämtliche Ausgaben an die Hardware zu Kontrollzwecken. Insbesondere bei Verwendung von DDE läßt sich die korrekte Funktion der DDE-Verbindung leicht überwachen. Die Listenlänge wird programmintern auf 1000 Einträge begrenzt, um Überlaufeffekte auszuschließen.

3.3.4 Dateneingabe in Dialogen

Die Dateneingaben für Ansteuerfrequenz, Abstrahlwinkel und Wandlerparameter erfolgen in je gesonderten Dialogen. Das hat zum Vorteil, daß ein konkreter Abschluß der Eingabe beim Drücken der OK-Taste vorliegt.

Ein weiterer Dialog, der nur über das Menü der Programms erreichbar ist, stellt seltener einzustellende Parameter ein.

Dialoge für Frequenz und Winkel

Die Angaben für Frequenz und Winkel werden ständig im Hauptfenster angezeigt, sind aber nur in den folgenden kleinen Dialogen wie im Bild 3.17 zu editieren:

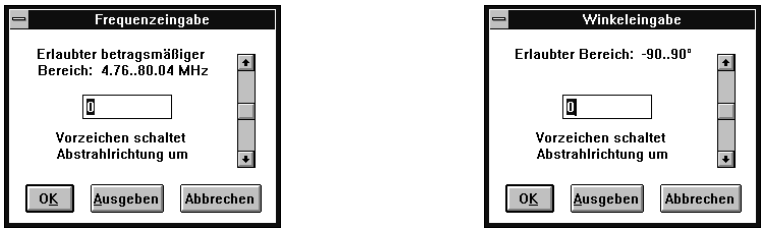


Bild 3.17: Dialoge für Ansteuerfrequenz und Abstrahlwinkel

Zusätzlich wurde in die Dialoge je ein Rollbalken zur bequemen Auswahl eines Wertes mit der Maus eingebaut. Der Rollbalken zeigt „Thermometerverhalten“, d.h. die großen Werte sind oben, die kleinen unten. Da Windows-Rollbalken (genauer: die Rollbalken des Objekts TScrollBar) ein umgekehrtes Verhalten aufweisen, wurden bei allen Parameterübergaben die Vorzeichen vertauscht.

Diese Dialogfenster haben außerdem noch einen Schalter Ausgeben. Dieser bewirkt, daß der eingegebene Frequenzwert – wie beim Drücken der OK-Taste – übernommen wird. Jedoch wird das Dialogfenster nicht geschlossen, und der ausgewählte Wert auf die Hardware ausgegeben. Das vermeidet den umständlichen Arbeitsgang „OK - Ausgeben - Ändern“, um in sequentieller Folge den Parameter ein wenig zu ändern und die Wirkung zu beobachten.

Bei der Angabe eines ungültigen Wertes, sei es durch Überschreitung der in den Dialogen gezeigten Grenzen als auch bei Überläufen bei der Berechnung des jeweils anderen Parameters, erfolgt beim Drücken auf OK bzw. Ausgeben eine Fehlermeldung.

Die Hilfe erklärt jedes einzelne Dialogelement.

Dialoge für Wandlerparameter und Einstellung

Etwas seltener werden die Dialogfenster des Bildes 3.18 aufgerufen.

Bild 3.18: Dialoge für Wandlerparameter und Einstellung

Wie bei Frequenz und Winkel weist der Rollbalken Thermometerverhalten auf. Zusätzlich ist er logarithmisch.

Den Einstellungsdialog erreicht man über den Menüpunkt Optionen/ Einstellungen. Zur Auswahl werden alle im System verfügbaren Druckerports gestellt. Wird das Ankreuzfeld markiert, wird der eingestellte Wandlerparameter aus dem Dialog links sowie die Schallgeschwindigkeit bem Drücken von OK gespeichert. Die Auswahl der parallelen Schnittstelle wird dagegen immer bei OK gesichert.

3.3.5 DDE-Kommunikation

Gemäß der Forderung der Aufgabenstellung wurde das Programm auch als DDE-Server unter Verwendung der DDEML ausgelegt. Er ist in der Lage, Datenabfragen zu beantworten (REQUEST), Datenveränderungen dem Client mitzuteilen (ADVISE), neue Daten entgegenzunehmen (POKE) sowie Befehle auszuführen (EXECUTE). Die automatische Aktualisierung erfolgt über die DDEML via `DdePostAdvise()`. Während des Bestehens einer DDE-Verbindung „leuchtet" die DDE-Verbindungsanzeige auf. Dazu wurde ein Dialogelement RADIOBUTTON verwendet, welches deaktiviert (grau dargestellt) ist, da es im Gegensatz zu seiner regulären Bestimmung keine Benutzereingaben entgegennimmt.

SENDERST unterstützt die automatische Verbindungssuche *(WILDCONNECT)* nicht.

Alle eingebbaren Parameter mit Ausnahme der Schnittstelle können abgefragt, automatisch abgefragt und modifiziert werden. Die Ausgabe der Daten an das angeschlossene Sendersteuer-Gerät erfolgt clientseitig mit der `DdeExec()`-Funktion.

Schreibende DDE-Zugriffe werden sofort sichtbar gemacht, d.h. die Anzeigen im Hauptfenster laufen stetig mit und zeigen an, welche Werte gerade aktuell sind. Dialogfenster sind von der automatischen Aktualisierung ausgeschlossen, daher kann man während laufender schreibender DDE-Zugriffe „in Ruhe" seinen gewünschten Wert auswählen und mit den Tasten OK bzw. Ausgeben im gewünschten Moment zum Hauptfenster bringen.

Folgende Zeichenkettenkonstanten müssen für die Konversation mit SENDERST verwendet werden:

Service-Name ..... **SENDERST** (service)
- System-Thema..... **System** (standard topic)
  - Themen-Liste ..... **Topics** (data item)
  - Formate-Liste..... **Formats** (data item)
  - Systemdaten-Liste ..... **SysItems** (data item)
  - Themen-Liste ..... **TopicItemList** (data item)
  - Hilfe..... **Help** (data item)
- Themen-Name..... **SENDERST** (topic)
  - Ansteuerfrequenz ..... **Frequency** (data item)
  - Abstrahlwinkel..... **Angle** (data item)
  - Wandlerparameter..... **Param** (data item)
  - Sendesignalfolge ..... **Pulses** (data item)
  - Schallgeschwindigkeit..... **Cw** (data item)
  - Ausgeben! ..... **Output** (execute command)
  - Stop! ..... **Stop** (execute command)

Der Datenaustausch erfolgt auf Textbasis *(CF_TEXT)*, d.h. es werden Zeichenketten übertragen, die die (intern gänzlich anders dargestellte) Zahl repräsentieren. Dies ist gängigste Art, geringe Datenmengen auszutauschen. Bei der Übertragung der **Pulses** wird „permanent" als „0" kodiert, ansonsten werden die Anzahl der Pulse direkt angegeben.

Die Daten können sowohl gelesen *(REQUEST)* als auch geschrieben *(POKE)* werden. Wie auch bei der Dateneingabe in den Dialogfenstern erfolgen Fehlerprüfungen, sodaß bei fehlerhaften POKEs eine negative DDE-Bestätigung erfolgt. Im Gegensatz zur „Handeingabe" erscheinen keine Hinweisfenster.

SENDERST unterstützt die automatische Aktualisierung für jedes Datenelement. Wenn ein Client die automatische Aktualisierung aktiviert *(ADVSTART)*, werden bei jeder Wertänderung, sowohl von anderen Clients als auch vom Server SENDERST, die veränderten Daten aktualisiert.

Zusätzlich zu den lese- und schreibbaren Datenelementen *(ITEMs)* existieren 2 *EXECUTE*-Kommandos. Die Groß-/Kleinschreibung des Befehls ist dabei nicht von Belang, jedoch dürfen weder führende noch folgende Leerzeichen u.ä. angegeben werden. Auch hierbei werden Fehler, beispielsweise ein **Stop**, ohne daß eine permanente Ausgabe läuft, mit einer negativen DDE-Bestätigung quittiert. Zu beachten ist, daß **Output** die Ausgabe nur startet. Ob die Ausgabe gerade läuft, kann nicht abgefragt werden.

Außerdem wird das System-Thema mit einigen standardmäßigen Datenelementen unterstützt. Es hat den gleichen Aufbau wie schon im Abschnitt 3.2.4 beschrieben.

3.4 Komplettpaket für den Luftmeßplatz

Dieser Abschnitt beschreibt die Implementierung der Software für den Luftmeßplatz. Aufgrund der weitaus größeren Komplexität dieses Programms, die sich aus erweiterten Erfordernissen gegenüber den beiden vorangegangenen Programmen ergibt, wurde dieses Programm konzeptionell etwas anders aufgebaut. Gleich geblieben ist die Programmiersprache, Borland Pascal 7.0 mit Objekten. Der Name LUFTMP.EXE ergibt sich aus „Luftmeßplatz“.

Das Programm wurde als MDI-Programm, also ein Programm mit der Fähigkeit, gleichzeitig mehrere „Dokumente“ in Fenstern verarbeiten zu können, entwickelt. Jedes Fenster stellt dabei eine Meßreihe dar, die für sich geladen und gespeichert, eingestellt und angezeigt werden kann. Spezielle Belange, die sich durch den Einsatz eines Monochrom-Monitors ergeben, finden im Programm besondere Beachtung.

Zwei spezielle Fenster dienen zur Arbeit mit dem Positioniermotor bzw. zum Beobachten des Analog-Digital-Wandlers, dazu später mehr.

Das Programm macht dabei von 2 Units Gebrauch, die im Vorfeld entwickelt wurden, eine DDE-Unit sowie eine für die Statuszeile. Diese Units entsprechen besonders den Belangen für MDI-Programme, wo die Anzahl der Themen eine schwankende Größe bzw. eine Statuszeile Quasi-Standard ist. Eine Toolbar wurde nicht vorgesehen, da die wenigen Aktionen fast ebenso leicht über das Menü ausgeführt werden können und diese Leiste noch mehr von dem knappen Platz auf dem Monochrom-Bildschirm beansprucht.

Dieses Programm läßt sich nur einmal starten; der übergebene Parameter `HPrevInstance` wird dazu ausgewertet. Es akzeptiert eine ganze Reihe von Kommandozeilen-Argumenten, von denen auch mehrere durch Leerzeichen getrennt hintereinander stehen dürfen. Sie sind nicht case-sensitiv.

<dateiname>	Datei laden bzw. zum laufenden Programm hinzufügen
/debug	Hardware (ADC und Motor) simulieren
/noadc	Nur ADC simulieren
/embedding	Eingebettetes OLE-Objekt bearbeiten (in Vorbereitung)
/embedding *<dateiname>*	Verknüpftes OLE-Objekt bearbeiten (in Vorbereitung)

Die Kommandozeilenschalter mit „/embedding“ sind so standardisiert und wurden für die einfachere künftige Unterstützung von Objekteinbettung und -verknüpfung (OLE) implementiert.

Außerdem unterstützt das Programm Drag&Drop, das „Ziehen“ von Dateien aus dem Datei-Manager und „Fallenlassen“ derselben über dem laufenden Programm. Die fallengelassene Datei wird als Matlab-Datei geöffnet.

Zusätzlich wird echte kontextsensitive Hilfe über die Taste F1 oder der rechten Maustaste angeboten. Verantwortlich dafür ist eine Hook-Funktion vom Typ eines Nachrichten-Filters. Sie wirkt global für das gesamte Programm und benötigt vorteilhafterweise keine Hilfs-DLL.

Die Programm-Module sind in einer im Bild 3.19 gezeigten Weise voneinander abhängig.

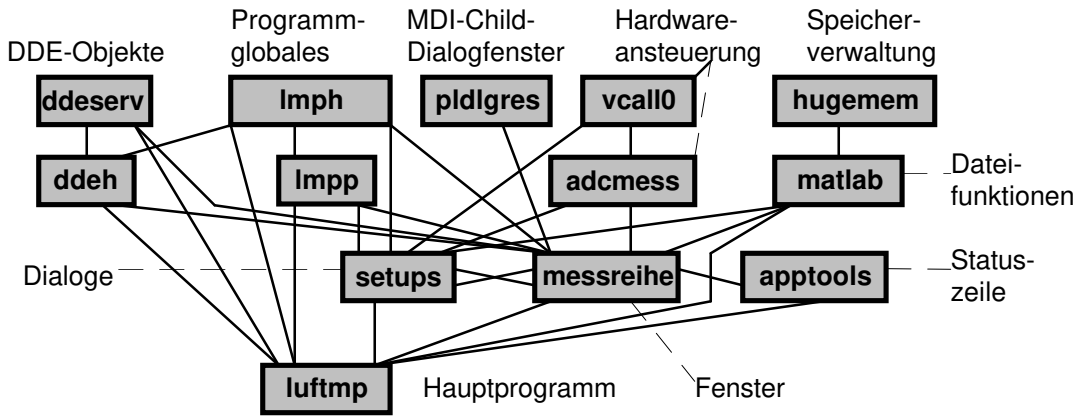


Bild 3.19: Graf der Abhängigkeiten der Programm-Module

Viele Programmteile beziehen sich auf die globalen Variablen-, Typen- und Prozedursammlungen **lmph** und **lmpp**. Daneben existieren eine Reihe isolierter Module, die so auch in anderen Programmen eingesetzt werden können. Zwischen **setups** und **messreihe** liegt ein Über-Kreuz-Bezug vor, die eine Unit inkludiert die andere jeweils im `implementation`-Teil. Dies ist im Sinne der „Schichtung“ von Programm-Modulen unfein, ließ sich hier jedoch nicht vermeiden.

Die Units **ddeserv** und **apptools** sind im Vorfeld dieser Arbeit entwickelte Programm-Module, original für GPIB-DSO.EXE [Ahl].

Die Unit **matlab** vereinfacht den Zugriff auf die binären .MAT-Dateien. Dabei genügt die Angabe des Variablennamens für sämtliche Dateibezüge. Diese Unit führt außerdem sämtliche erforderlichen Typkonvertierungen automatisch, schnell und speichersparend aus. Sie verarbeitet Variablenfelder mit mehr als 64 KB Größe. Dafür macht sie sich die Hilfe der Unit **hugemem** zunutze. Beide Units enthalten eine ausführliche Dokumentation in ihrem `interface`-Abschnitt.

Die kurzgehaltene Unit **pldlgres** (PlayDialogResource) ermöglicht es, in normalen Fenstern eine Reihe Kindfenster auf der Basis einer normalen Windows-Dialog-Ressource anzulegen. Eine ähnliche Funktionalität weisen die Windows-Funktionen `DialogBox…` und `CreateDialog…` auf. Diese sind jedoch als MDI-Kindfenster nicht nutzbar. Der Versuch, es dennoch irgendwie zu erreichen, führt dazu, daß die Rollbalken des MDI-Client-Bereiches nicht so funktionieren, wie sie es sollen. Das liegt an der Umgehung des normalen Weges, mit der richtige MDI-Kindfenster erzeugt werden müssen (via `WM_MDICreate`). Die Unit **pldlgres** erspart die Mühe, jedes einzelne Fenster mit `CreateWindow()` programmäßig zu erzeugen und zeigt gleichzeitig auf, wie Windows in etwa mit Dialog-Ressourcen arbeitet.

Schließlich ist die Unit **vcall0** ein Modul, mit der ein Virtueller Gerätetreiber namens VCALL0.386 angesprochen werden kann, mit dem wiederum beliebiger 16-bit-Code als privilegierter Code im Ring 0 des 386-Prozessors abgearbeitet werden kann. VCALL0.386 wiederum ist ein Programm, das im Rahmen dieser Arbeit entstand. Es ist recht einfach gehalten, sodaß es nicht in einem Extra-Abschnitt wie z.B. der Motortreiber im Abschnitt 3.1, sondern im Rahmen des Luftmeßplatz-Programms erläutert wird.

Im Gegensatz zu den vorangegangenen kleineren Programmen wurden Fenster und Dialoge jeweils zusammen in eine Datei gepackt, da es ansonsten zu viele Dateien werden. Weil im Programm BP.EXE 9 Fenster schnell per Hot-Key (ALT+Ziffer) erreichbar sind, und eines davon fast immer das Hilfe-Fenster ist, sind mehr als 8 Dateien nicht mehr so leicht zu handhaben.

Die Vererbungs-Hierarchie ist mit nur einer Generation Nachkommen von Standard-Objekten noch recht überschaubar, beachtet man nicht die vielen Objekte innerhalb von **ddeserv**. Das Bild 3.21 zeigt diese Hierarchie.

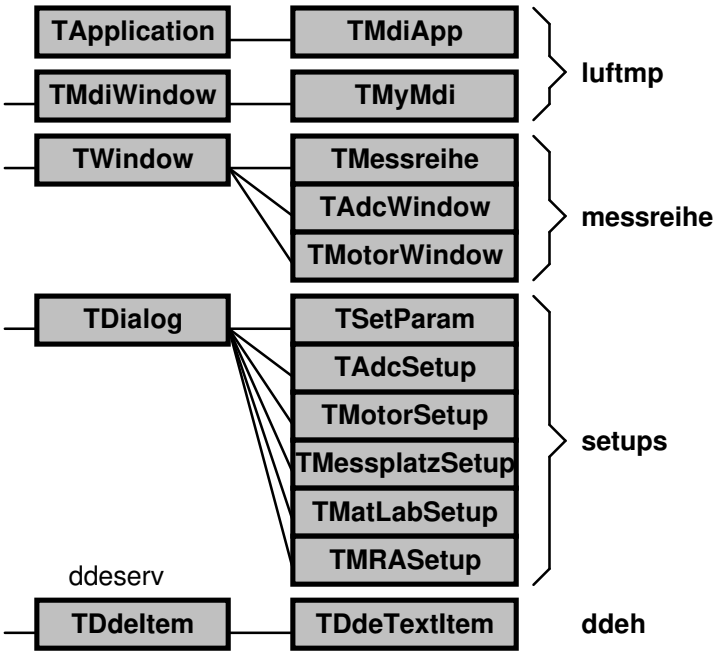


Bild 3.20: Vererbungs-Hierarchie der Objekte in LUFTMP.EXE

Programmglobale Aktionen, wie das Öffnen neuer Fenster, werden von TMyMdi aus gestartet; der meiste Code ist jedoch fensterbezogen und liegt in den drei Objekten der Unit **messreihe**. Die Dialoge der Unit **setups** sind recht einfach und uniform gehalten. Nur TSetParam weicht davon ab, da eine Vielzahl Abhängigkeiten zwischen Dialogelementen zu beachten sind. TDdeTextItem repräsentiert ein spezielles DDE-Datenelement, welches ausschließlich mit dem Zwischenablageformat CF_TEXT arbeitet und maximal 32 Zeichen überträgt. Dieser Spezialfall genügt an fast allen Stellen den Anforderungen.

Programmtechnisch wurde dafür gesorgt, daß die Fenster weitestgehend autonom sind. Beispielsweise erfolgt die Steuerung des Motor-Fensters vom Meßreihen-Fenster aus nur über Fenster-Nachrichten.

3.4.1 Oberfläche

Das Programm stellt sich als MDI-Fenster mit Menü- und Statuszeile dar. Als Beispiel mit 2 Kindfenstern und einer laufenden Messung zeigt Bild 3.21 einen „Schnappschuß“ des Programms.

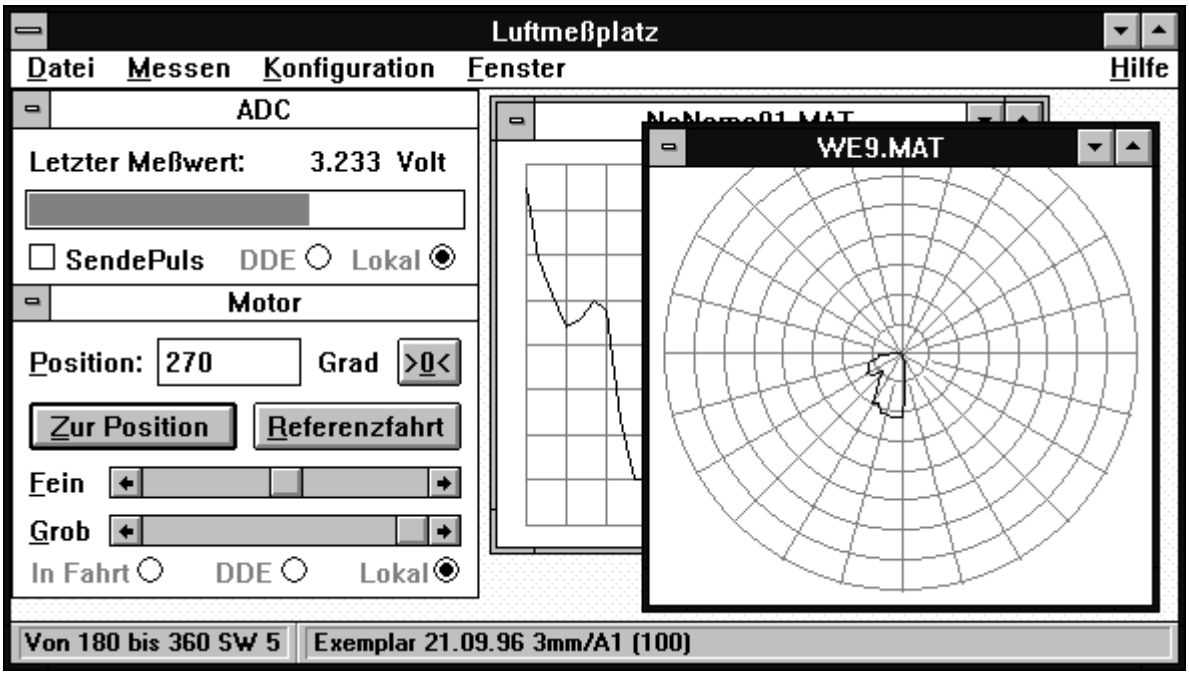


Bild 3.21: MDI-Hauptfenster des Programms LUFTMP.EXE

Das Menü Datei beinhaltet Menüpunkte zum Anlegen, zum Laden und Speichern sowie zum Ausdrucken von Meßreihen. Die Datenspeicherung kann sowohl im Matlab-Format erfolgen, welches von diesem Programm wieder gelesen werden kann, als auch als Wertetabelle für den Export zu gängigen Funktionsdarstellungsprogrammen wie Excel oder Origin. Die Wertetabelle kann nicht wieder eingelesen werden. Beim Laden einer Meßreihe findet ein leicht veränderter Datei-Öffnen-Dialog Anwendung, bei dem der Kommentar einer selektierten Matlab-Datei sofort angezeigt wird. Der Ausdruck einer Meßreihe erfolgt papierformatfüllend in derselben Form, wie sie gerade im Fenster dargestellt wird. Ist dieses Verhalten nicht gewünscht, sollte die Meßreihe über die Zwischenablage in gängige Textverarbeitungs- oder DTP-Programme eingebunden und von dort aus ausgedruckt werden.

Der Menüpunkt Datei/Beenden beendet das Programm. Ist eine DDE-Verbindung aktiv, oder wird gerade eine Meßreihe aufgezeichnet, erfolgt eine Sicherheitsabfrage.

Unter dem Menü Messen sind Funktionen untergebracht, die sich auf die Meßreihen-Fenster beziehen. So startet Messen/ Parameter einstellen ein Dialogfenster zum Einstellen von Start- und Endwinkel, der Schrittweite, dem Kommentar usw. Die Menüpunkte Start! bzw. Halt! beginnen bzw. beenden die automatische Meßreihen-Aufnahme. Während der Aufnahme sind die Lokal-Anzeigen des Motor- und ADC-Fensters aktiviert, um auf den Betriebsfall der programminternen Automatik hinzuweisen, wie es im Bild 3.21 zu sehen ist. Während der Meßreihen-Aufnahme wird die zugehörige grafische Anzeige stets aktualisiert, um sofort einen Eindruck vom Ergebnis zu bekommen. Ebenso funktionieren alle anzeige-relevanten Aktionen, wie das Kopieren in die Zwischenablage oder das Umschalten zwischen kartesischer und Polardarstellung. Da das Starten der Aufnahme eine bereits aufgezeichnete Meßreihe überschreibt, existiert der Menüpunkt Start im neuen Fenster, welches mit den gleichen Daten für Startwinkel, Endwinkel und Schrittweite des zuletzt fokussierten Meßreihen-Fensters sofort die Aufnahme in einem neuen Fenster beginnt. Mit Maximum suchen kann eine automatische Nullpositionierung des Motors auf die Hauptstrahlrichtung des Wandlers hin erfolgen. Der Suchbereich ist dabei auf ±10° beschränkt.

Der Menüpunkt Messen/ Kopieren bringt die gerade fokussierte Meßreihe in die Zwischenablage. Das programmtechnische Vorgehen bei der Anzeige und beim Kopieren erläutert Abschnitt 3.4.5. Schließlich dient Kartesische Anzeige als ein markierbarer Menüpunkt zur schnellen Umschaltmöglichkeit zwischen kartesischer und polarer Darstellung einer Meßreihe.

Das Menü Konfiguration beherbergt Startpunkte für fünf Einstell-Dialoge. Sämtliche Einstellungen werden nicht automatisch gespeichert, sondern erst beim Anwählen von Konfiguration speichern. Standardmäßig warnt das Programm nicht, wenn eine Meßreihe ohne zu speichern geschlossen wird. Ist dieses Verhalten unerwünscht, kann der Menüpunkt Warnung beim Schließen markiert werden.

Das Fenster-Menü weist gegenüber anderen MDI-Programmen nur 2 zusätzliche Menüpunkte auf. Diese Menüpunkte, ADC-Fenster und Motor-Fenster, verstecken jene Fenster bzw. holen sie hervor. Diese beiden Fenster werden auf keinen Fall wirklich geschlossen, auch nicht über ihr jeweiliges Systemmenü (das sieht nur so aus). Da diese beiden Fenster wichtige Programm-Funktionalitäten beinhalten, werden sie nur versteckt.

Das Hilfe-Menü weist drei häufig gebrauchte Einsprungpunkte in die zugehörige Hilfedatei LUFTMP.HLP auf. Der vierte Menüpunkt Über informiert über das Programm und die laufende Version in einem kleinen Meldungsfenster.

3.4.2 Meßfunktion und Timing - das Fenster „ADC“

Der Analog-Digital-Umsetzer sowie der Sendeverstärker mit Spitzenwertgleichrichter wird innerhalb dieses Fensters verwaltet. Die Darstellung des Meßwertes als Zahl und als Bargraph ist nur ein Teil der Fenster-Funktionalität.

Funktionsprinzip der rechnerunabhängigen Zeitnahme

Als Quelle der rechnerunabhängigen Zeitmessung dient der Kanal 0 des Zeitgeberschaltkreises 8253. Unter Windows arbeitet er im Betriebsmodus Nadelimpulsgenerator. Dabei kann sein 16-bit-Zählerstand jederzeit abgefragt werden. Folgende Programmsequenz holt den momentanen Zählerstand vom Kanal 0:

```
      xor   al,al
      out   43h,al
      in    al,40h
      xchg  ah,al
      in    al,40h
      xchg  al,ah
      neg   ax
```

Der Zähler zählt abwärts. Deshalb erfolgt eine Negierung des gelesenen Wertes am Ende der Sequenz.

Am Takteingang CLK2 des Zählers liegt die viergeteilte Taktfrequenz des Ur-PC von 1,193180 MHz an. Alle 838 ns inkrementiert der Zählerstand. Zu beachten ist, daß der Zähler unter DOS in der Betriebsart Rechteckgenerator programmiert ist. Hierbei arbeitet der Zähler in Zweierschritten [PCTIM]. Diesen Unterschied macht das Testprogramm WINTMR.PAS deutlich, welches sowohl unter DOS als auch unter Windows läuft.

Das genaue Warten erfolgt nunmehr dadurch, daß der Zeitgeber solange permanent ausgelesen wird, bis vom Startpunkt aus gemessen ein bestimmter Differenz-Zählerstand erreicht ist.

```
      call  GetTimerVal
      mov   bx,ax
@@1:  call  GetTimerVal
      sub   ax,bx
      cmp   ax,[TicksToWait]
      jc    @@1
```

Dieser Schleifendurchlauf erfolgt am besten bei gesperrten Interrupts, um Fehlmessungen durch zufällig einstreuende Interrupts zu vermeiden. Auch wird unter Windows die preemptive Taskumschaltung hin zu einer DOS-Box unterbunden. Die machbare Warte-Präzision liegt bei 6..16 Ticks je nach Rechner-Geschwindigkeit. Damit kann eine Genauigkeit im 10-µs-Bereich garantiert werden.

Unter Windows im Enhanced Mode gibt es den Standard-Gerätetreiber VTD (Virtual Timer Device), welcher gerade die Portzugriffe auf die Adressen 40h und 43h abfängt. Das kostet nicht nur Zeit und schmälert so die machbare Präzision, sondern gibt dem Kernel wieder die Möglichkeit zur preemptiven Task-umschaltung. Die dadurch entstehenden Fehlmessungen im Millisekundenbereich (je nach eingestellter Zeitscheiben-Dauer) sind inakzeptabel.

Eine Lösung wäre ein spezieller Virtueller Gerätetreiber für die Zeitnahme am Luftmeßplatz. Doch der ungestörte Portzugriff ist ein häufiger auftauchendes Problem; daher fiel die Entscheidung zugunsten eines allgemein verwendbaren VxD's.

Der Virtuelle Gerätetreiber VCALL0.386

Dieser Treiber ermöglicht das Laufenlassen von 16-bit-Code im privilegierten Modus. Die Idee dazu lieferte das Kapitel 5 des Buches [Dav], Abschnitt “Calling Code in a TSR at Ring 0”. Der 16-bit-Codeabschnitt wird an der angegebenen Adresse über ein Aliasdeskriptor ausgeführt. Wichtigste zu beachtende Regel für diese Code-Abschnitte ist, daß FAR-Aufrufe zu nierderprivilegiertem Code verboten sind. Länger andauernde Berechnungen sind zu vermeiden. Ansonsten gibt es keine Besonderheiten zu beachten. Der Treiber arbeitet auch mit Real-Mode-Programmen zusammen.

Da auch dieses Programm eine Anwenderschnittstelle (API) zur sinnvollen Funktion benötigt, wurde für diesen Treiber eine ID beantragt. Er bekam die Nummer 3A7Ah. Zum VxD gehören eine Reihe Dateien.

Datei	Beschreibung
VCALL0.386	Der Virtuelle Gerätetreiber für Windows
VCALL0.ASM	Quelldatei für Turbo Assembler 3.2
VCALL0.DEF	Modul-Definitions-Datei für den Linker LINK386.EXE
VCALL0.TXT	Kurzdokumentation
VCALL0.PAS	Unit-Quelldatei
VCALL0T.PAS	Einfaches Beispiel zur Benutzung von VCALL0
VCALL0.C	Quelldatei für Borland C
VCALL0.H	Header-Datei für Borland-C-Programme

Tabelle 3.2: Liste der einzelnen Programm-Module zum VCALL0.386-VxD

Auch dieses VxD wurde mit dem Turbo Assembler entwickelt. Im wesentlichen kommen Funktionen des VMM zum Einsatz, um Selektoren zu beschaffen bzw. zu setzen. Kern des Programms ist eine Routine, die einen Stack-Rahmen zur Rückkehr zum VxD aufbaut. Dazu ist ein Mini-Deskriptor mit einer Länge von 2 Bytes mit einem Adreßlängenattribut von 32bit im Adreßbereich des VxD erforderlich, mit dem dann der Rücksprung zum flachen VxD-Speichermodell gelingt.

Funktion der Analog-Digital-Umsetzung

Wie bei allen A/D-Umsetzern nach dem Verfahren der sukzessiven Approximation wird die Umsetzung zunächst gestartet, dann die Umsetzzeit gewartet und danach der Meßwert ausgelesen. Aufgrund des binären Funktionsprinzips des Wägeverfahrens liefern diese Umsetzer meistens Binärzahlen. Das Warten erfolgt meist durch Polling (zyklische Abfrage) einer Statusleitung des Umsetzers, häufig mit dem Namen EOC, “End Of Convert”. Dies ist bei dieser Karte leider nicht vorgesehen, sodaß mit Hilfe des voranstehend genannten Algorithmus' die maximale Umsetzzeit gewartet werden muß.

Die Zeitsteuerung und die Umsetzung liegen in der Unit **adcmess**. Hier ist die hardwarenahe Steuerung des Programms zusammengefaßt.

3.4.3 Ansteuerung des Motors - das Fenster „Motor“

Der Positioniermotor wird über das Motor-Fenster, wie es im Bild 3.21 zu sehen ist, gesteuert. Es stellt eine verkleinerte Form des Hauptfensters des Programms MOTORST.EXE des Abschnitts 3.2 dar. Aus dieser Ähnlichkeit ergibt sich eine leichter erlernbare Bedienung des Motors.

Umbauarbeiten an der Motorsteuerung und am Motor

Am Luftmeßplatz wurde der Motor mit Getriebe einerseits sowie der Endschalter andererseits auf einer Experimentierschiene befestigt. Eine im vorhandenen Luftmeßplatzprogramm angesteuerte Schaltung zur Deaktivierung des Endschalters war nicht zu finden, und der Not-Aus-Taster war defekt. Deshalb wurde zunächst die Motorsteuerung mit folgenden Zielsetzungen überarbeitet:

- Verbindung von Motor, Anschlußleiterplatte und Endschalter zu einer mechanischen Einheit
- Einbau einer Endschalter-Abschaltung
- Einbau eines funktionsfähigen Not-Aus-Tasters
- Einbau einer Endstufen-Abschaltung zur Störungsunterdrückung beim Messen
- Steuerung der Endschalter- und Endstufen-Abschaltung vom Computer aus

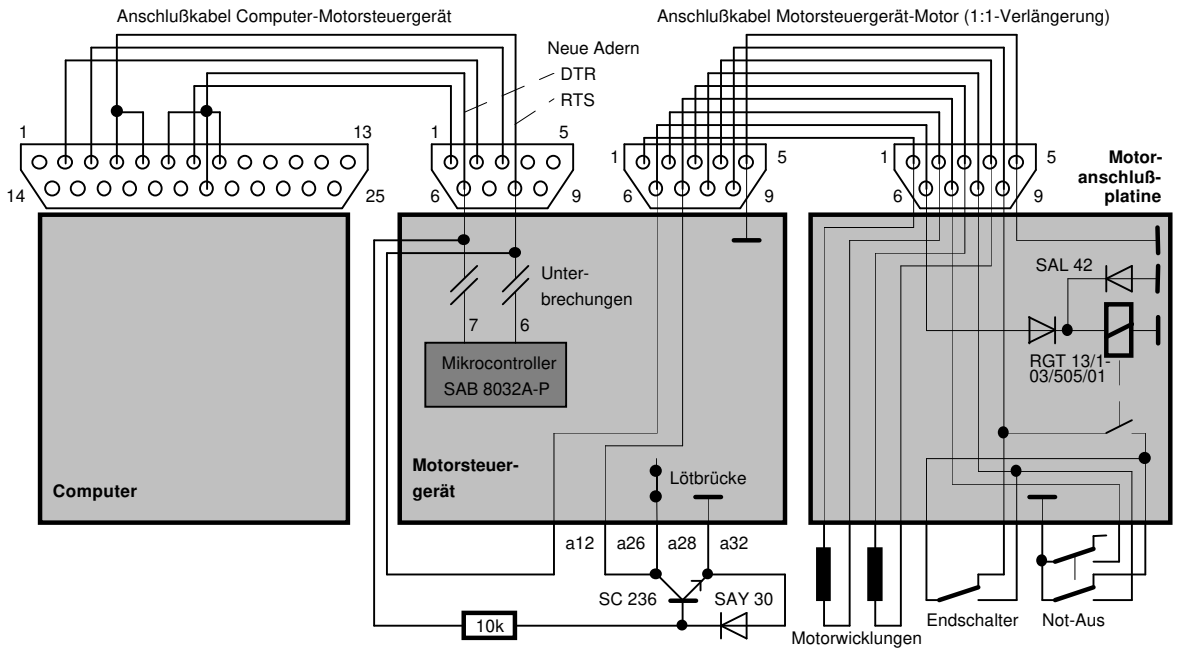


Bild 3.22: Schaltplan der Ergänzungen am Motor und an der Motorsteuerung

Die dazu notwendigen Schaltungsmaßnahmen sind im Bild 3.22 dokumentiert. Zusätzlich befindet sich der Schaltplan an der Rückseite des Getriebeblocks. Die Steuerung der beiden Automatiken wäre über freie Portausgänge der A/D-Wandlerkarte denkbar gewesen. Dieses Vorgehen bringt schlecht handhabbare Querverbindungen mit sich. Daher wurden die beiden nicht benutzten Steuerausgänge DTR ("Data Terminal Ready") und RTS ("Request To Send") des seriellen Ports für diesen Zweck genutzt. Um die frontseitige serielle Anschlußbuchse der Motorsteuerung für die Signalzuführung verwenden zu können, wurden zwei laut Dokumentation [isel2] nicht verwendete Ausgänge durch Trennen zweier Leiterzüge zum Mikrocontroller innerhalb der Steuereinheit freigemacht.

Das verwendete Verbindungskabel zwischen Computer und Motorsteuergerät ist ein spezielles und kein gewöhnliches Nullmodem-Kabel. Ursache ist die Beschaltung des Pin 5 des Steuergerätes (linke Buchse im Bild 3.22) mit 5 Volt Betriebsspannung statt der normalen Beschaltung dieses Pins mit Masse.

Der Transistor SC 236 zieht den Anschluß a28 nach Masse, wenn auf der Leitung DTR eine positive Spannung liegt. Damit werden die Endstufen deaktiviert, die Motorachse wird frei, und Störimpulse verschwinden. Zur Aktivierung des Eingangs a28 mußte eine Lötbrücke gesetzt werden. Im Falle negativer Spannung auf DTR schließt die Diode SAY 30 die Basis-Emitter-Strecke kurz.

Die im Schaltplan linke Diode der Doppeldiode SAL 42 läßt nur positive Spannungen auf der Leitung RTS zum DIL-Relais RGT… durch. Die Relaiswicklung hat einen Innenwiderstand von ca. 750 Ω und eine Nennspannung von 5 V. Diese Belastung ist für eine serielle Schnittstellenleitung zumutbar; die Spannung bricht im Versuchsfall von 9 auf 5 Volt zusammen. Die serielle Schnittstelle eines Computers ist dauerkurzschlußfest. Negative Spannungsspitzen beim Abschalten des Relais werden durch die zweite Diode der SAL 42 abgeleitet.

Das Relais überbrückt im Arbeisfall den Ruhekontakt des Endschalters. Ein Versuch zeigte, daß dies genügt. Die Steuerkarte wertet offenbar nur den Pegel an Pin 9 („normalerweise geschlossen“) aus. Im Ruhefall liegt dieser Anschluß auf Masse, und im Arbeitsfall des Endschalters wird Pin 8 („normalerweise offen“) auf Massepotential gelegt.

Der Not-Aus-Taster bewirkt dasselbe wie das Anfahren des Endschalters: der Schrittmotor hält an. Im Gegensatz zum Endschalter ist der Taster nicht per Computerprogramm und Relais deaktivierbar. Um eine zusätzliche Sicherheit zu gewinnen, werden während des Tastendruckes die Endstufen abgeschaltet. Dadurch ist es im übrigen möglich, während des Festhaltens dieses Tasters die Achse von Hand zu bewegen.

Steuerung im Programm mit TimeOut

Der Computer kommuniziert mit der Motorsteuereinheit über die serielle Schnittstelle. Dabei sendet der Computer ein Kommando, und nach Vollzug des Kommandos antwortet das Steuergerät durch Übertragung eines Quittungszeichens. Die Möglichkeit des Steuergeräts, bei einigen Kommandos sofort die Quittung zu liefern, fiel außer Betracht, da sonst für den Computer keine Möglichkeit besteht, das tatsächliche Ende eines Kommandos abzuwarten. Da während der Kommandoausführung das Steuergerät keine weiteren Befehle verarbeitet, ist das Kommando-Ende für den Computer sehr wichtig.

Von der Vielzahl möglicher Kommandos kommen folgende vier zur Anwendung:

Kommando (C-`printf`-Notation)	**Funktion**
`"@01\n"`	Initialisierung
`"@0d%d\n", RefSpeed`	Setzen der Referenzgeschwindigkeit
`"@0R1\n"`	Start der Referenzfahrt
`"@0A %ld,%d\n", LastMoveBy, MovSpeed`	Start einer Bewegung mit Entfernungs- und Geschwindigkeitsangabe

Tabelle 3.3: Verwendete Kommandos für die Motorsteuerung

Die Geschwindigkeitsangaben haben dabei die Einheit Schritt pro Sekunde. Dabei sind nur positive Zahlen zwischen 21 und 10000 zulässig. Die Motorsteuerung liefert nach vollständiger Verarbeitung der Befehle entweder das Zeichen „0“ als Code für fehlerfreie Verarbeitung oder aber einen Fehlercode zurück.

Um abzusichern, daß bei defekter serieller Verbindung der Computer nicht „ewig“ auf die Antwort vom Steuergerät wartet, wurde eine Zeitüberschreitungs-Logik eingebaut. Die etwaige Dauer einer Kommando-Ausführung läßt sich aufgrund der Bekanntheit von Weglänge und Geschwindigkeit recht genau ermitteln. Auf diese Weise ist kein benutzerunfreundliches, pauschal maximal großes TimeOut vonnöten, sondern der Zeitüberschreitungsfehler tritt so früh wie möglich auf. Programmtechnisch simuliert die TimeOut-Prozedur das Senden eines „T“ vom Steuergerät. So fällt dieser Fehler automatisch in die gleiche Fehlerbehandlungsprozedur wie beim Auftreten „echter“ Fehlercodes vom Steuergerät.

Das Programm führt in der LongInt-Variablen `ExactPos` Buch über die in Schritten genaue Momentanposition der Motorachse. Bei Bewegung zu einem neuen Punkt wird die Differenz ermittelt, in `LastMoveBy` gespeichert und die Bewegung um `LastMoveBy` gestartet. Bei fehlerfreier Quittung wird `ExactPos` um `LastMoveBy` vorzeichenrichtig erhöht und repräsentiert nunmehr die neue exakte Position. Mit Hilfe dieser Technik können sich keine Quantisierungsfehler aufsummieren. Im übrigen verhält sich die Motorsteuerung auch bei Bewegung um null Schritte so; dieser Sonderfall benötigt daher keine programmtechnische Beachtung.

Drei boolesche Variablen sind für Statusinformationen wichtig. Die Variable `InMove` ist true, wenn sich der Motor gerade bewegt, sowohl bei einer normalen Bewegung als auch bei Referenzfahrt. `InSync` dagegen ist parallel zu `InMove` wahr, wenn eine Referenzfahrt durchgeführt wird. Die Variable `Synced` gibt an, ob eine korrekte Synchronisierung durch eine fehlerfreie Referenzfahrt und nachfolgend fehlerfreien Bewegungen vorliegt. Bei einem Fehler geht das Programm davon aus, daß die Synchronisierung außer Tritt ist und setzt `Synced` auf false. `InMove` und `Synced` können per DDE überwacht werden.

Im Gegensatz zum Wassermeßplatz-Motortreiber sind Bewegungen auch ohne vorhergehende Referenzfahrt möglich. Dies ist hier auch wichtig, da sich die Achse wegen des angeflanschten 50:1-Getriebes nur schwer von Hand bewegen läßt. Daher kann unter Verwendung der Rollbalken oder durch Zahleneingaben die Achse um die gewünschten Winkelgrade gedreht werden. Der Anwender muß dabei selbst auf die mechanischen Grenzen der Anlage acht geben. Vor dem Ausführen der Referenzfahrt sollte sich die Achse auf der richtigen Seite zum Referenzschalter befinden. Die Referenz-Fahrtrichtung ist steuerungsintern festgelegt und programmäßig nicht beeinflußbar.

Ein Abbrechen einer laufenden Bewegung kann softwaremäßig durch Abschalten der Endstufen erfolgen. Die Steuerkarte bemerkt dies und sendet den Fehlercode „F“ zurück, der nicht dokumentiert ist. In diesem Fall muß der Taster „Reset“ am Steuergerät gedrückt werden, bevor weitere Bewegungen möglich sind.

3.4.4 Datenaufnahme in einzelne MDI-Fenster

Obgleich mehrere Fenster gleichzeitig Meßreihen darstellen können, kann zu einer Zeit nur eines davon Daten aufnehmen. Dies wird dadurch abgesichert, daß die Fenster „ADC“ und „Motor“ jeweils nur einen „Besitzer“ haben können, die diese Fenster steuern. Durch diese Technik wird es erleichtert, künftig evtl. mehrere solcher Fenster vorzusehen, womit das gleichzeitige Aufnehmen mehrerer Meßreihen möglich wäre.

Steuerung der Fenster „ADC“ und „Motor“

Das Fenster, das mit der Meßreihenaufnahme beginnen möchte, prüft zunächst, ob die Fenster „ADC“ und „Motor“ nicht bereits in Lokalbesitz sind. Sind sie frei, werden diese Fenster in Besitz genommen. Die Befehlsübergabe für die Motorbewegungen bzw. die Meßwertaufnahme erfolgt mittels `SendMessage()` zwischen den Fenstern. Nach erfolgter Meßreihenaufnahme werden die beiden Fenster wieder freigegeben.

Die Lokalsteuerung hat Vorrang vor der DDE-Steuerung dieser beiden Fenster. In diesem Fall werden schreibende DDE-Zugriffe abgelehnt. Der Zustand der Lokalsteuerung dieser Fenstern wird durch „Aufleuchten“ der „Lokal“-Anzeige in diesen Fenstern kenntlich gemacht.

3.4.5 Datendarstellung

Für die grafische Darstellung eines Kurvenzugs ist die Windows-Funktion `PolyLine()` wie geschaffen. Das dazu erforderliche Punktefeld wird während der Messung generiert. Eine Neuberechnung des Kurvenzuges erfolgt nur noch beim Umschalten der Anzeige, z.B. zwischen kartesisch und polar, sowie beim Laden einer Meßreihen-Datei, nicht jedoch bei Größenveränderungen. Grundlage dafür sind benutzerdefinierte Abbildungsmaßstäbe.

Der Spezialfall eines Monochrom-Bildschirms findet bei der Darstellung des Koordinatensystems Beachtung. Bei mehr als 2 Farben werden diese Linien grau dargestellt, ansonsten schwarz gepunktet. „Graue“ Linien führten auf dem Monochrom-Bildschirm zu weißen Linien auf weißem Grund.

Der Einsatz von Abbildungsmodi

Das Programm macht für die grafische Darstellung der Daten Gebrauch von Abbildungsmodi. Ein festes virtuelles Koordinatensystem wird in das - von der Fenstergröße abhängige - Gerätekoordinatensystem von Windows automatisch umgerechnet. Vorteilhaft ist, daß der Kurvenzug, die Poly-Linie, aus einem immer konstanten Punktefeld besteht, das auch beim Verändern der Fenstergeometrie niemals umgerechnet werden muß. Dieses Verfahren spart Zeit.

Grundsätzlich verschieden ist die kartesische und die Polarkoordinatendarstellung zu betrachten. Während bei kartesischer Darstellung ein anisotropischer Abbildungsmodus mit voneinander unabhängiger X- und Y-Skalierung wie „auf den Leib geschneidert“ ist, erfordert die korrekte Darstellung im Polarkoordinatensystem den isotropischen Abbildungsmodus, damit Kreise als Kreise und nicht als Ellipsen erscheinen.

Das Kopieren des Bildes in die Zwischenablage

Der Menüpunkt Messen/Kopieren bringt das Bild des gerade fokussierten Meßreihen-Fensters in die Zwischenablage. Dazu wird der Programmteil zum Darstellen der Meßreihe, welchem ein Gerätekontext-handle `HDC` übergeben wird, wiederverwendet. Diesmal wird als Gerätekontext eine Speicher-Metadatei übergeben. Eine Metadatei ist eine Aneinanderreihung von Zeichenbefehlen. Sie bleibt frei skalierbar und benötigt i.a. weniger Speicher als eine Bitmap. Nach Beendigung des Zeichenvorgangs wird die Metadatei in die Zwischenablage gestellt. Das Handle kann nicht direkt übergeben werden, sondern muß in eine mit `GlobalAlloc()` zu beschaffene Struktur gestellt werden, in der auch die Idealabmessungen der Metadatei abgelegt werden.

Der isotropische Modus ist nicht so sehr für Metadateien geeignet; daher wird auch im Falle des Polar-koordinatensystems der anisotropische Modus mit frei skalierbaren X- und Y-Achsen gewählt. Der Anwender muß beim Importieren der Grafik selbst auf das richtige Seitenverhältnis achten. WinWord zum Beispiel zeigt beim Skalieren die Streckungen beider Achsen in Prozent an; sie sollten gleich sein, damit Kreise Kreise bleiben. Im Falle des Exports einer isotropischen Metadatei war die Grafik unter WinWord nicht skalierbar, stattdessen beschneidbar.

3.4.6 Laden, Speichern und das Dateiformat

Als Dateiformat wurde das Matlab-1.0-Format nach [EIG] gewählt, um die Daten problemlos mit diesem Programm austauschen zu können. Es handelt sich dabei um ein Binärformat, welches mehrere Variablen unter Variablennamen (Bezeichnern) mit verschiedenen Datentypen abzuspeichern vermag.

Typ	Bedeutung der Ziffernstelle
0xxx	IEEE Little Endian („Intel-Notation“) für die Byte-Ordnung
1xxx	IEEE Big Endian („Motorola-Notation“) für die Byte-Ordnung
x0xx	Felder sind spaltenweise abgelegt
x1xx	Felder sind zeilenweise abgelegt
xx0x	`double` – Gleitkommazahlen 64 bit
xx1x	`single` – Gleitkommazahlen 32 bit
xx2x	`long` – vorzeichenbehaftete Ganzzahlen 32 bit (Pascal: `LongInt`)
xx3x	`short` – vorzeichenbehaftete Ganzzahlen 16 bit (Pascal: `Integer`)
xx4x	`unsigned short` – vorzeichenlose Ganzzahlen 16 bit (Pascal: `Word`)
xx5x	`unsigned char` – vorzeichenlose Ganzzahlen 8 bit (Pascal: `Byte`)
xxx0	Numerische Matrix
xxx1	Text-Matrix
xxx2	Sparse-Matrix (dünn besetzte Matrix)

Tabelle 3.4: Bedeutung der Typ-Angabe

Für die Konvertierung zwischen den verschiedenen Formaten existiert ein Funktionen-Array (`array [0..5,0..5] of function`), welches die Konvertierung in das Wunschformat beim Lesen bzw. in ein erlaubtes Zielformat beim Schreiben in höchstmöglicher Geschwindigkeit erlaubt. Denkbar wäre aber auch eine Lösung mit einem eindimensionalen Funktions-Array und einem Zwischenschritt über das größtmögliche Zahlenformat `double`.

Bei der Suche eines erlaubten Schreibformates wird vom gegebenen Format aus hin zu größeren Datentypen ein erlaubtes gesucht. Schlägt dies fehl, wird vom gegebenen Format aus hin zu kleineren Datentypen gesucht. Beim Speichern sind im letzteren Fall Abschneide-Effekte möglich. Mindestens eins der Matlab-Formate muß zum Schreiben erlaubt sein, nach Möglichkeit sollte sich ein Gleitkommatyp darunter befinden.

3.4.7 Dialoge

Dieses Programm bietet 6 Dialoge zu verschiedenen Einstellungen an. Zu jedem Dialogelement wird kontextsensitive Hilfe durch Druck auf die Taste F1 oder die rechte Maustaste angeboten.

Dialoge zur Meßreihen-Aufnahme, -Darstellung und -Speicherung

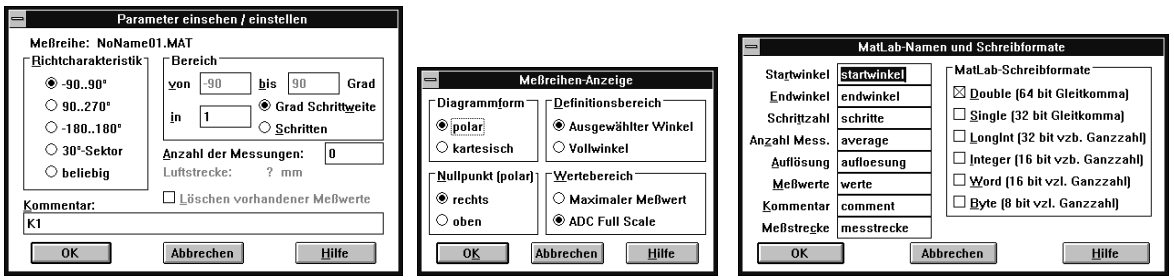


Bild 3.23: 3 Dialoge zur Meßreihe

Für die Arbeit mit Meßreihen existieren die im Bild 3.23 zu sehenden Dialoge. Damit sind vielfältige Manipulationen möglich. Die Matlab-Zeichenketten sollten jedoch möglichst nicht verändert werden. Die hier gezeigten Einstellungen sind Vorgabewerte. Man beachte, daß – angezeigt durch quadratische Ankreuzfelder – auch mehrere Matlab-Schreibformate erlaubt sind.

Dialoge zur Hardware-Einstellung

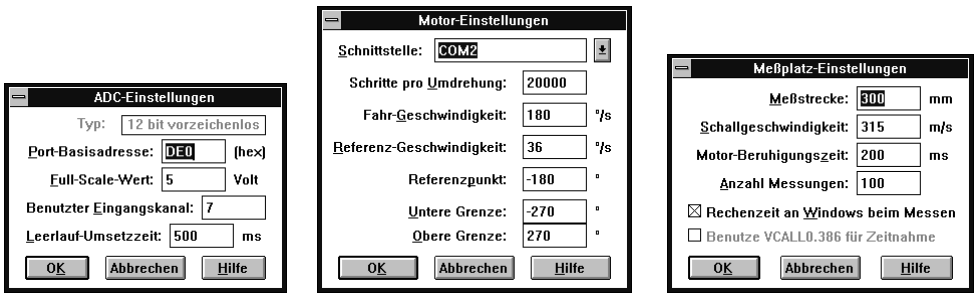


Bild 3.24: 3 Dialoge zur Hardware-Einstellung

Zur Anpassung an variierende Hardware-Einstellungen existieren die im Bild 3.24 gezeigten Dialogfenster. Sie sind allesamt unter dem Menü Konfiguration zu erreichen. Graue Zeichenketten innerhalb der Dialoge weisen auf feste, programminterne Einstellungen hin, die nicht verändert werden können. Sie könnten in einer künftigen Version dieses Programms änderbar sein.

3.4.8 DDE-Konversation

LUFTMP.EXE ist ein DDE-Server unter Benutzung der DDEML. Er ist in der Lage, Datenabfragen zu beantworten *(REQUEST)*, Datenveränderungen dem Client mitzuteilen *(ADVISE)*, neue Daten entgegenzunehmen *(POKE)* sowie Befehle auszuführen *(EXECUTE)*.

Folgende Zeichenkettenkonstanten müssen für die Konversation mit LUFTMP verwendet werden:

Service-Name ..... **LUFTMP** (service)
- System-Thema..... **System** (standard topic)
  - System-Datenelemente..... **SysItems** (data item)
  - Themen-Liste..... **Topics** (data item)
  - Formate-Liste..... **Formats** (data item)
  - Item-Liste..... **TopicItemList** (data item)
  - Hilfe..... **Help** (data item)
  - Neue Meßreihe!..... **New** (execute command): Neues Meßreihen-Fenster öffnen
- Thema ADC ..... **ADC** (topic)
  - Abstand..... **Distance** (data item) enthält den Abstand Sender-Mikrofon in mm
  - Schallgeschwindigkeit..... **Cl** (data item) enthält die Schallgeschwindigkeit im m/s

Arbeitend-Flag ..... **Working** (data item) enthält 0 oder 1 (messend)
Hintergrundarbeit-Flag ..... **EnabBkg** (data item) enthält 0 oder 1 (Hintergrundarbeit)
Meßwert ..... **Value** (data item) enthält den letzten Meßwert
Messungen pro Salve ..... **Avg** (data item) enthält die Anzahl der Messungen für Mittelwert
Start Messung! ..... **Start** (execute command): Start einer Messung
Thema Motor ..... **Motor** (topic)
Position ..... **Position** (data item) enthält die Relativposition
In-Bewegung-Flag ..... **InMove** (data item) enthält 0 (stehend) oder 1 (in Bewegung)
Synchron-Flag ..... **Synced** (data item) enthält 0 oder 1 (Referenzfahrt ausgeführt)
Initialisierung! ..... **Init** (execute command): Gesamte Einrichtung initialisieren
Referenzfahrt! ..... **Sync** (execute command): Referenzfahrt durchführen
Bewegen! ..... **Move xxx** (execute command): Bewegen nach **xxx**
Anhalten! ..... **Stop** (execute command): Motor anhalten
Freigeben! ..... **Free** (execute command): Motor stromlos schalten (bei Stillstand)
Thema Meßreihe ..... ***<dateiname>*** (topic)
Startwinkel ..... **Start** (data item) enthält den Startwinkel in Grad
Endwinkel ..... **End** (data item) enthält den Endwinkel in Grad
Schrittweite ..... **Step** (data item) enthält die Schrittweite in Grad
In-Aufnahme-Flag ..... **Working** (data item) enthält 0 oder 1 (Aufnahme läuft)
Meßwerte-Liste ..... **Values** (data item) enthält die Meßwerte
Messungen pro Salve ..... **Avg** (data item) enthält die Anzahl der Messungen für Mittelwert
Kommentar ..... **Comment** (data item) enthält den Datei-Kommentar
Dateiname ..... **Name** (data item) enthält noch einmal den Dateinamen
Start Aufnahme! ..... **Start** (execute command): Start einer Messung

Der Datenaustausch erfolgt auf Textbasis *(CF_TEXT)*, d.h. es werden Zeichenketten übertragen, die die Zahlen repräsentieren. Dies ist gängigste Art, geringe Datenmengen auszutauschen.

Es steht ein **System**-Thema zur Verfügung, welches über die DDE-Möglichkeiten dieses Programms informiert.

LUFTMP unterstützt die automatische Verbindungssuche *(WILDCONNECT)* nicht.

Erläuterungen zum ADC-Thema

Ein unveränderliches Thema dieses DDE-Servers ist das ADC-Thema. Es enthält 6 Datenelemente und stellt eine Execute-Funktionen zur Verfügung. Alle Datenelemente sind „advisabe“, können also Änderungen automatisch dem Client mitteilen. Bis auf **Working** und **Value** sind alle Datenelemente veränderbar. Während aktiver Lokalsteuerung dieses Fensters werden jedoch schreibende Zugriffe (*POKE*) sowie *EXECUTE* unterbunden und mit *DDE_FBUSY* beantwortet. Die Lokalsteuerung, also der programminterne Datenaustausch, hat Vorrang vor DDE.

Das DDE-Thema „ADC“ beherbergt nicht nur Funktionen zum Ansprechen des Analog-Digital-Konverters, wie der Name vermuten läßt, sondern auch meßplatzrelevante Angaben für die Zeitsteuerung, wie den Abstand Sender-Mikrofon und die Schallgeschwindigkeit. Es wird immer mit Zeitsteuerung gemessen.

Distance:

Dieses Datenelement enthält die Entfernung Sender-Mikrofon in Millimetern.

Cl:

Die Schallgeschwindigkeit wird in diesem Datenelement in Metern pro Sekunde abgelegt. Die Einheit entspricht Millimetern pro Mikrosekunde. Gemeinsam mit **Distance** dient diese Variable zur Berechnung der Wartezeiten für die Zeitsteuerung. Siehe hierzu auch Abschnitt 2.2.3.

Working:

Dieses zweiwertige Datenelement ist während einer laufenden Messung (Meß-Salve) ist dieser Wert auf 1 gesetzt. Während der Messung kann kein *EXECUTE* ausgeführt werden. Veränderungen an anderen Variablen sind möglich, werden jedoch nicht für die laufende Messung wirksam.

Diese Variable abzufragen macht nur bei aktiviertem **EnabBkg** Sinn, da ansonsten ohnehin der Prozessor während der Meß-Salve blockiert ist, d.h. eine Abfrage durch außenstehende Programme immer 0 liefert.

EnabBkg:

Dieses zweiwertige Datenelement schaltet die Taskumschaltung zwischen Einzelmessungen einer Meß-Salve ein oder aus. Eingeschaltet ist Windows flüssiger, der Meßvorgang jedoch langsamer und auf dem Oszilloskop wegen auftretender Unregelmäßigkeiten schwieriger zu beobachten.

Value:

Dieses Datenelement enthält den letzten Meßwert einer Meß-Salve in Volt. Es kann nur gelesen werden.

Avg:

Hiermit wird die Anzahl der Einzelmessungen pro Meß-Salve angegeben. Die Zahl 100 ist dabei sinnvolle Vorgabe.

Start:

Mit diesem einzigen *EXECUTE*-Kommando innerhalb des ADC-Themas wird eine Meß-Salve gestartet.

Erläuterungen zum Motor-Thema

Ein weiteres unveränderliches Thema dieses DDE-Servers ist das Motor-Thema. Es ist sehr ähnlich zum Motor-Thema des Programms MOTORST.EXE aufgebaut, siehe hierzu auch Abschnit 3.2.4. Im folgenden werden speziell Unterschide herausgearbeitet.

Position:

Eine Veränderung der **Position** zieht immer auch eine Bewegung des Motors nach sich; das Setzen der **Position** ist also gleichbedeutend mit dem Execute-Kommando **Move xxx**. Die Angabe selbst ist absolut zum Nullpunkt und in Grad zu verstehen, d.h. ein Akkumulieren ist nicht möglich. Das vermeidet ein Zunehmen systematischer Fehler. Eine negative DDE-Bestätigung wird geliefert, wenn die Zielposition außerhalb festgelegter Grenzen liegt, oder der Motor nicht synchronisiert war (**Synced**=0).

Erfolgt momentan eine Bewegung (**InMove**=1), ist das kein Fehler - die neue Position wird in eine 1 Element lange Warteschlange gestellt. Diese neue Position wird dann automatisch beim Beenden des laufenden Bewegungssegmentes angefahren. Ist die Warteschlange bereits gefüllt, wird der Wert dort überschrieben.

InMove:

Das Datenelement **InMove** zeigt den Bewegungsstatus des Motors an und kann von einem DDE-Client sowohl durch zyklische Abfrage (Polling) als auch über die automatische Aktualisierung *(ADVSTART)* ausgewertet und für einen Programmfluß einer DDE unterstützenden Skriptsprache herangezogen werden. Dieses Datenelement ist „advisable“, jedoch nicht mit *POKE* veränderbar.

Synced:

Das Datenelement **Synced** informiert darüber, ob eine gültige Referenzfahrt ausgeführt wurde (Synced=1), oder ob eine Referenzfahrt erforderlich ist (Synced=0). Dieses Datenelement ist ebenfalls „advisable“, und nicht mit *POKE* veränderbar.

Init:

Diese Funktion initialisiert die Schrittmotorsteuerung komplett, genauso wie bei MOTORST.EXE. Danach wird eine Referenzfahrt in exakt derselben Folge durchgeführt wie beim Drücken auf den Schalter Referenzfahrt:

- Bewegen des Motors zu einer Position neben dem Referenztaster, wenn Motor synchronisiert war ODER Meldungsfenster zeigen, wenn der Motor nicht synchronisiert war
- Referenzfahrt ausführen
- Motor zur Position 0 bewegen

Sync:

Hiermit wird eine „stille“ Referenzfahrt ausgeführt. Ein DDE-Client muß ggf. selbst entsprechende Warnungen dem Anwender mitteilen, was die Funktion **Init** automatisch tut. Der Server kehrt bei Aufruf von **Sync** sofort zurück, er wartet also nicht auf das Ende der Bewegung. Das Ende der Bewegung kann durch Beobachtung der Variable **InMove** oder **Synced** abgewartet werden.

Move xxx:

Hiermit wird der Motor zu einer bestimmten Position bewegt. **xxx** ist dabei eine Gleitkommazahl im ASCII-Format. Dezimaltrenner darf Punkt oder Komma sein. Das Trennzeichen zwischen **Move** und **xxx** ist Weißraum (Leerzeichen oder Tabulator) und kann auch weggelassen werden. Der Server kehrt bei Aufruf von **Move xxx** sofort zurück, er wartet also keinesfalls auf das Ende der Bewegung. Die Aktualisierung der **Position** erfolgt erst am Bewegungsende, so auch der *ADVISE*-Zyklus. Dabei wird die Istposition angenommen; dieser Wert kann sich von dem bei **Move xxx** angegebenen um unvermeidliche Quantisierungsfehler unterscheiden.

Stop:

Dieses Kommando bewirkt einen Soforthalt. Die Endstufen werden abgeschaltet, und eine Fehlermeldung der Motorsteuerung wird erzwungen. **Synced** wird auf 0 gesetzt. Dieses Kommando wirkt nur während der Bewegung. **InMove** wird auf 0 gesetzt. Dieses Kommando hat eine vergleichbare Funktion zum Not-Aus-Taster an der Anlage. Dieses Kommando unterscheidet sich in seiner Funktion von der in MOTORST.EXE dahingehend, daß hier die Synchronisierung mit Sicherheit verlorengeht..

Free:

Im Gegensatz zu **Stop** wirkt dieses Kommando nur während des Stillstandes. Die Motorspulen werden stromlos geschaltet; das Haltemoment wird nur durch das Getriebe erzeugt. Es dient zur Störungsvermeidung während einer Messung. **Synced** bleibt auf 1 gesetzt. Beim Start der nächsten Bewegung werden die Endstufen automatisch wieder zugeschaltet. Auch dieses Kommando unterscheidet sich in Funktion und Nutzen von dem **Free**-Kommando aus Abschnitt 3.2.4.

Erläuterungen zum *<dateiname>*-Thema

Das veränderliche Thema dieses DDE-Servers ist das *<dateiname>*-Thema. Sein Name hängt von der gerade geladenen .MAT-Datei ab. Es enthält 8 Datenelemente und stellt eine Execute-Funktion zur Verfügung. Alle Datenelemente sind „advisabe“, können also Änderungen automatisch dem Client mitteilen. Viele Datenelemente können verändert werden (*POKE*).

Start, End, Step:

Diese Datenelemente legen Startwinkel, Endwinkel und Schrittweite einer Meßreihe in Grad fest. Es sind Gleitkommazahlen; das Dezimaltrennzeichen darf Punkt (englisch) oder Komma (deutsch) sein. Das Verändern dieser Werte zieht das Löschen einer evtl. vorhandenen Meßreihe nach sich.

Working:

Während einer Meßreihen-Aufnahme ist dieses zweiwertige Datenelement auf 1 gesetzt. Es ist nur lesbar.

Values:

Hier stehen die Meßwerte einer Meßreihe in Volt. Die Zahlen sind durch Leerzeichen getrennt. Diese Daten sind nur lesbar.

Avg:

Die Anzahl der Einzelmessungen pro Meß-Salve wird hier vermerkt. Ist dieser Wert 0, wird der Wert **Avg** aus dem ADC-Thema verwendet.

Comment:

Dieses textuelle Datenelement enthält beliebigen Kommentar zur Meßreihe. Er darf ca. 40 KB Umfang annehmen.

Name:

Der Themenname ist hier zusätzlich vermerkt. Durch Verändern dieses Datenelements wird das Thema umbenannt. Themen-Umbenennungen durch Abspeichern seitens des Anwenders werden hier reflektiert und sind so für einen DDE-Clienten nachvollziehbar.

Start:

Die Meßreihe wird aufgenommen. Eine evtl. vorhandene Meßreihe wird gelöscht. Diese *EXECUTE*-Funktion kehrt sofort zurück. Das Ende der Meßreihen-Aufnahme kann durch Abfrage der Variable **Working** erfolgen.

Erläuterungen zum System-Thema

Dieses Programm stellt auch ein **System**-Thema zur Verfügung, welches über die DDE-Möglichkeiten informiert. Das Thema ist genauso wie in den beiden vorhergehenden Programmen aufgebaut und ist im Anschnitt 3.2.4 beschrieben. Zusätzlich reflektieren sich dynamische Änderungen der vorhandenen DDE-Themen in den entsprechenden Listen-Datenelementen **Topics** und **TopicItemList**.

Das Kommando **New** konnte nirgendwo anders untergebracht werden. Es dient zum Öffnen neuer Meßreihen-Fenster.

3.5 Hilfe-Dateien

Ein Programm ist nur so gut, wie man es ohne ein Handbuch zu wälzen bedienen kann. Deshalb wurde auf eine gute und durchgehende Hilfe Wert gelegt. Unter Windows wird man bei diesem Vorhaben nicht allein gelassen.

Für die Erstellung der Hilfe gibt es 2 Ansätze: Entweder wird mit einem Texteditor Hilfetext mit speziell festgelegten Symbolen für Sprünge (Links) angefertigt und dann zu einer Hilfe compiliert, oder die Hilfe wird direkt mit einem speziellen Hilfe-Editor erstellt. Für das zuerst genannte Verfahren werden von Microsoft Compiler angeboten, die .RTF-Text (im Rich-Text-Format) zusammen mit einer Projektdatei zu einer Hilfedatei compiliert. Die andere Variante wird von Microsoft bisher nicht unterstützt. Hierzu gibt es Shareware-Produkte mit hohem Qualitätsstandard. Da letztere Produkte nicht gerade billig und in ihrer freien Version arg eingeschränkt sind, wurde der Weg über den Hilfe-Compiler beschritten.

3.5.1 Erstellung

Für alle 4 erstellten Programmkomponenten wurden Hilfedateien erstellt. Als Texteditor fand WinWord in der Version 2.0b Verwendung, mit dem die für den Windows-Hilfe-Compiler benötigten Rich-Texte (Endung .RTF) erstellt wurden. Gegenüber der direkten Eingabe des Rich-Textes überwiegt der Vorteil der WYSIWYG-Darstellung, die fast deckungsgleich mit der Darstellung durch WINHELP.EXE ist. Die zu beachtenden „feinen Unterschiede“ werden im folgenden aufgezeigt.

Auswahl des Hilfe-Compilers

Zur Compilierung der Hilfe wurden nicht die zu Borland Pascal mitgelieferte Hilfe-Compiler HC.EXE bzw. HC31.EXE eingesetzt, da diese DOS-Programme nur Real-Mode-Speicher als Zwischenspeicher einsetzen, der bei größeren Dateien mit Bildern schnell zu knapp wird, insbesondere in der Windows-DOS-Box. Geeigneter ist der Protected-Mode-Hilfe-Compiler HCP.EXE, der auf verschiedenen FTP-Servern unter HC505.ZIP als Freeware verfügbar ist. Er ist ebenfalls ein DOS-Programm, welches jedoch Erweiterungsspeicher (XMS) nutzen kann.

Erstellung und Einbindung von Bildern

Die Erstellung der Bilder mit den anklickbaren Bereichen erfolgte nach folgender Vorschrift:

- Windows in 16-Farben-Modus umstellen (z.B. Standard-VGA-Auflösung)
- Programm starten, dessen Fenster in die Hilfe aufgenommen werden sollen
- SHED.EXE (Hotspot Editor, im Lieferumfang von Borland Pascal 7.0) starten
- Gewünschtes Fenster erzeugen und mit ALT+PrintSreen-Taste in die Zwischenablage „drucken“
- Nach SHED gehen und mit SHIFT+Einfg-Taste das Bild von der Zwischenablage holen
- Anklickbare Bereiche markieren und zur Hilfedatei passende Schlüsselwörter einsetzen. Dabei ist von rechts unten nach links oben (rückwärts) zu verfahren, damit dann in der Hilfe die Auswahl mit der Tabulator-Taste in einer erwarteten Reihenfolge (von links oben nach rechts unten) erfolgt. Die Bereichsmarkierung sollte pixelgenau erfolgen, da die Bereichsgrenzen in der Hilfe als gestrichelte Linien sichtbar sind – oder man schaltet die Umrahmung aus.
- Das Bild mit den anklickbaren Bereichen wird gespeichert (Endung .SHG) und ggf. mit dem nächsten Fenster fortgefahren.

Das „Einfangen“ von aufgeklappten Menüs gelingt mit ALT+PrintScreen nicht. In einem solchen Fall kann beispielsweise der gesamte Bildschirm mit der Taste PrintScreen in die Zwischenablage gebracht werden, aus der dann das Menü oder der gewünschte Bereich mit einem geeigneten Editor (z.B. PaintBrush) herausgeschnitten wird.

Von Hause aus unterstützt das Rich-Text-Format keine textumfließbaren Grafiken. Da diese jedoch von WINHELP.EXE unterstützt werden und zur Verbesserung der Lesbarkeit und der Platzausnutzung auf dem Bildschirm beitragen, dienen 2 „Pseudo-RTF-Anweisungen“ `bml` und `bmr` (im Gegensatz zu den Ausführungen in der Borland-Pascal-Hilfe ohne Backslash „\“ einzugeben) dem Hilfe-Compiler als Token zur Einbindung solcher Bilder. Es können sowohl normale (.BMP) als auch Bilder mit anklickbaren Bereichen (.SHG) angegeben werden. Alternativ sind auch Metadateien erlaubt. Sogenannte Multi-Resolution-Bitmaps (.MRB) enthalten mehrere Bitmaps für im Idealfall jeden der Grafikadapter CGA, EGA, VGA und 8514 und können ebenfalls eingebunden werden. Sie werden mit dem Werkzeug MRBC.EXE erstellt. Sie fanden hier keine Verwendung, da hier nur VGA und Hercules (monochrom) benötigt wird und MRBC.EXE gerade diese Karte nicht unterstützt.

Der Textumfluß um ein textumfließbares Bild endet mit dem ersten Absatzendezeichen.

Normale, also Bilder auf der Grundlinie einer Textzeile, können unter WinWord sehr einfach mit Einfügen/Grafik eingesetzt werden. Ebenso ist es möglich, daß eingebettete Objekte Verwendung finden, was z.B. mit MS-Draw gut funktioniert. In diesem Fall wird eine Metadatei eingebunden, was insbesondere für Strichzeichnungen vorteilhaft ist und dabei die Größe der Hilfedatei gegenüber Bitmaps reduziert. Man beachte jedoch bei solcherart eingebundenen Grafiken, daß mehrere gleiche Grafiken auch mehrfach eingebunden werden. Für wiederholte Grafikelemente, wie Bullets, Logos u.ä., sollte die Anweisung `bmc` verwendet werden, um die Hilfedatei nicht unnötig aufzublähen.

Besonderheiten beim Erstellen von Text

Der Text wird unter Verwendung der für Textsysteme üblichen Formatierungsanweisungen, wie hängenden Einzug oder Verwendung von Fettschrift erstellt. Auch Farben können zweckmäßig verwendet werden, da farbige Bildschirme wesentlich häufiger anzutreffen sind als z.B. Farbdrucker. Zu beachten ist, daß einige Formatierungsanweisungen für die Generierung von Links vorbehalten sind, die im Interesse einer konsistenten Windows-Hilfe unverändert verwendet werden sollten. Es sollten nur wenige, klar verschiedene Fonts (Zeichensätze) verwendet werden, die möglichst leicht lesbar sind und auf allen Windows-Systemen verfügbar sein sollten. Die True-Type-Schriftarten Arial und Times New Roman sowie die Systemschrift sind solche Schriften. Zusätzlich wurde die Schriftart `Courier` als nichtproportionaler Zeichensatz für Programm-Passagen benutzt.

Die Verwendung von „handgemachten“ Einrückungen und Formatierungen ist unzweckmäßig, da sie nur bei einer bestimmten Breite des Hilfefensters korrekt dargestellt werden. Bei Verwendung des hängenden Einzugs muß beachtet werden, daß für die gleiche Darstellung des Textes unter WINHELP.EXE ein (für WinWord unnötiger) Tabstop an der linken Schreibkante erforderlich ist. Der hängende Einzug ist insbesondere für Aufzählungen geeignet.

Die Verwendung einer nicht-numerierten Aufzählung unter Zuhilfenahme von “Bullets” (kleinen Markern) wird nicht korrekt übersetzt; hier bleibt ein Ausweg in der Einbindung von kleinen Bildern mit der Anweisung `bmc`.

Ebenfalls zu beachten ist, daß die Windows-Hilfe jede Formatanweisung als potentielle Trennstelle ansieht. So ist es bei einer Hervorhebung vor einem Komma zweckmäßig, das Komma in die Hervorhebung mit einzuschließen. Das ist typografisch nicht korrekt, verhindert aber, daß WINHELP das Komma an den Anfang einer Zeile setzt.

Sinngemäß gilt das auch für Hervorhebungen in Klammern. Hier ist es ratsam, den Satz derart umzustellen, daß die Klammer unnötig wird, da dieser typografische Fehler mehr auffällt. Möglicherweise ist dieser mitunter sehr lästige Fehler unter Windows95 beseitigt.

Windows-Hilfeseiten dürfen zuoberst einen Absatz aufweisen, der mit dem nächsten Absatz verbunden ist. WINHELP stellt diese Zeile permanent am oberen Bildrand dar. Die Hintergrundfarbe dieses Bildschirmabschnitts wird in der Projektdatei im `[windows]`-Abschnitt festgelegt. Unnütze sind solche Zeilen in Sekundärfenstern, da es in diesem Falle nur zur Darstellung des Titels kommt. Der Hilfe-Compiler gibt hierzu keine Warnung aus. Einige Hilfeseiten der Borland-Pascal-Hilfe weisen diesen Fehler auf. An den Seiteninhalt kann man in diesem Falle nicht herankommen.

Kompatibilitätsfragen

Nach dem Erstellen der .RTF-Datei (mit WinWord) und der Hilfe-Projektdatei (.HPJ) wird die Hilfe mit einem der oben aufgeführten Hilfe-Compiler erzeugt. Die so erzeugte Hilfe ist unter allen Windows-Versionen ab 3.1 (HC30.EXE: Windows 3.0) verwendbar. Man beachte, daß Windows-Hilfe im allgemeinen nicht abwärtskompatibel ist, d.h. Windows 95-Hilfe nicht mit WINHELP.EXE der Version 3.1 anzeigbar ist.

Microsoft schlägt vor, im Falle von Windows 3.0 die WINHELP.EXE dieser Version durch die der Version 3.1 auszutauschen, wohl wegen erheblich erweitertem Umfang und einiger beseitigter Fehler. WINHELP.EXE von Windows 95 ist unter Windows 3.1 auch mit Win32s nicht lauffähig.

4 Ergebnisse

Nach dem Abschluß der Programmierarbeiten konnte die entstandene Software im Einsatz zur Testung kommen. Dabei kamen insbesondere die Programme zum Wassermeßplatz zum Einsatz. Aufgrund der begrenzten Zeit kann das Luftmeßplatzprogramm nicht so gründlich getestet werden.

Alle Programme wurden so entwickelt, daß sie zu Vorführungszwecken o.ä. auch ohne Hardware funktionieren. Diese kleine Mühe beim Programmieren und das bißchen Code mehr lohnt sich. Der Code wurde so eingebaut, daß möglichst viele Programmteile noch durchlaufen werden. Auf diese Weise lassen sich eine Reihe Fehler auch ohne angeschlossene Gerätschaften auffinden.

4.1 Anwendung der Software am Wassermeßplatz

Die Programmkomponenten für den Wassermeßplatz, bestehend aus dem Motortreiber MPK3D.386, der Motorsteuerung MOTORST.EXE und der Ultraschallsendewandlersteuerung SENDERST.EXE funktionieren in Verbindung mit dem Programm GPIB-DSO.EXE [Ahl] unter der Steuerung von Matlab problemlos zusammen. Einziges Problem ist Matlab selbst. Einerseits hat das Matlab-DDE-Protokoll Fehler, beispielsweise funktioniert die Übergabe von Fehlercodes bei `DdeExec()` nicht. Noch problematischer ist, daß während einiger notwendiger Wartezeiten der Rechner vollkommen blockiert und somit ein vernünftiges Arbeiten mit dem Multitasking-Betriebssystem Windows nicht mehr möglich ist. Somit verschwindet der Vorteil, der zu erhoffen war, daß der Rechner durch den Meßvorgang nicht komplett belegt ist.

Saubere Abhilfe bringt nur die Korrektur der fehlerhaften Matlab-Programmbestandteile. Möglicherweise kann unter Windows 95 das Matlab in einer eigenen 16-bit-Windows-Maschine gestartet werden, wodurch sich die Blockierung nicht mehr systemweit erstrecken kann.

Für das Starten mehrerer Programme gleichzeitig wurde das kurze Programm GRPSTART.EXE entwickelt, welches eine ganze Programmgruppe des Programm-Managers zu starten vermag. Der Name der Gruppe wird über die Kommandozeile festgelegt. Die Gruppen-Daten werden über DDE unter Verwendung der DDEML vom Programm-Manager beschafft. Der Aufbau einer DDE-Konversation mit PROGMAN.EXE ist in der Borland-Pascal-Hilfe dokumentiert.

4.1.1 Elektromagnetische Verträglichkeit

Die Störungen, die durch die Schrittmotorsteuerkarte erzeugt werden, sind erheblich. Die geeignetste Lösung hierfür ist, die Impulse für die Zeit der Messung abzuschalten. Dadurch würde der Motor sein Haltemoment verlieren und außer Tritt geraten. Ein elektromagnetisch betätigter Bremsklotz für die Motorachse könnte dieses Problem beseitigen. Er ließe sich recht einfach über den Relaisausgang der Schrittmotorsteuerkarte steuern. Zur Verwendung des Relaisausgangs sind kleinere Umstellungen an der Motorsteuerung MOTORST.EXE erforderlich.

4.2 Anwendung der Software am Luftmeßplatz

Am Luftmeßplatz kommt ein relativ leistungsschwacher Rechner 386SX/16 mit Monochrom-Monitor zum Einsatz. Dennoch läuft das Programm recht zügig ab. Wenn das Programm längere Zeit rechnet, beispielsweise beim Umschalten von kartesischer auf Polardarstellung, wird programmäßig die Sanduhr eingeblendet. Ebenfalls spürbare Wartezeiten treten beim Laden und Speichern von Meßreihen auf. Ansonsten beinhaltet das Programm keine Phasen mit länger andauernden Berechnungen. Sämtliche Blockierzeiten liegen an diesem Rechner unter 1 s, das ist erheblich weniger lang als das Matlab-Skript am Wassermeßplatz blockiert.

Für die Feststellung des Zeitverhaltens von Windows und dem Zeitgeberschaltkreis 8253 wurde das Programm WINTMR.EXE entwickelt. Das Pascal-Programm ist sowohl für DOS als auch für Windows compilierbar. Es brachte interessante Erkenntnisse. Die beiden entstehenden .EXE-Dateien werden vorzugsweise mit dem Freeware-Werkzeug GLUE.EXE zu einem „Zwitter-Programm“ kombiniert.

4.3 Wiederverwendbarkeit der Software

Es ist nicht anzunehmen, daß die im Rahmen dieser Arbeit entstandene Software ohne Änderungen bei anderen Steuerungs- und Meßaufgaben eingesetzt werden kann. Jedoch wurde mit der durchgehenden Modularisierung der Programme in Hierarchiestufen eine Basis vorhandener Programm-Module geschaffen, die ohne oder mit geringem Änderungsbedarf auch anderweitig eingesetzt werden kann.

Ebenso kann vergleichsweise leicht die Plattform geändert werden. Die Programmteile für die Gestaltung der Programmoberfläche muß hierbei meist komplett umgestellt werden; wiederverwendbar bleiben die Routinen der Hardwaresteuerung.

Verzeichnisse

Literaturverzeichnis

zu Schrittmotoren:

[Fisch] Rolf Fischer: „Elektrische Maschinen“, 8. Aufl., S. 343ff., Carl Hanser Verlag München, 1992, ISBN 3-446-16482-0

[isel1] Dokumentation zur „*isel*-Microstep-Steuerkarte“, *isel*-automation Hugo Isert Eiterfeld, 1994

[isel2] Dokumentation zur „Schrittmotorsteuerkarte mit Mikroprozessor“, *isel*-automation Hugo Isert Eiterfeld, 1989

zu Virtuellen Gerätetreibern für Windows:

[Dav] David Thielen, Bryan Woodruff: “Writing Windows Virtual Device Drivers”, Addison-Wesley, 1994

[Haz] Karen Hazzah: “Writing Windows VxDs and Device Drivers”, R&D Publications

[MSDN] Microsoft Informationsblatt zur CeBit '96: „MSDN – 3 Levels braucht der Entwickler“, 1996

[MSJ] Microsoft System Journal: „Gerätetreiber für Windows 95“, 2/96, S. 120 ff.

[Nor] Daniel Norton: “Writing Windows Device Drivers”, Addison Wesley

zu Matlab:

[EIG] “Matlab External Interface Guide”, The MathWorx Company

Sonstiges:

[AD12] „Technische Beschreibung der AD-12 Bit Karte mit 4x sample & hold 16 TTL-I/O und IRQ“, Kolter-Electronic

[Ahl] Software zur Ansteuerung eines Oszilloskops GPIB-DSO

[LKP] „Linux-Kernel-Programmierung“, Addison-Wesley, 2. Aufl. 1994

[Mess] Hans-Peter Messmer: „PC-Hardwarebuch“, 3. Aufl, Addison-Wesley

On-Line-Referenzen:

[BPOLH] Borland Pascal für Windows Referenz

[MSDN6] Microsoft Developer Network (MSDN): “Development Library Disc Six”, 1994

[New] Newsgroup “comp.os.ms-windows.programmer.vxd”

[PCTIM] ftp://ftp.tu-chemnitz.de/pub/simtel/msdos/info/pctim003.zip: “FAQ / Application notes: Timing on the PC family under DOS”, 1996

[RBIL] ftp://ftp.tu-chemnitz.de/pub/simtel/msdos/info/intwin48.zip: “Ralph Brown Interrupt List”, 1996

Studien- und Diplomarbeiten:

[Dietz] Dietze, Heiko: Studienarbeit „Entwurf eines Programms zur rechnergesteuerten Erfassung der Richtcharakteristik von Ultraschallwandlern“, Technische Universität Chemnitz, Fakultät für Elektrotechnik und Informationstechnik, Professur Meß- und Sensortechnik, 1994

[Fri] Fritz, Egbert: Studienarbeit „Meßplatz zur Aufnahme der Richtcharakteristik von PVDF-Hydrophonzeilen“, Technische Universität Chemnitz, Fakultät für Elektrotechnik und Informationstechnik, Professur Meß- und Sensortechnik, 1995

[Hart] Hartmann, Ralf: Studienarbeit „Entwurf und Realisierung von Hardware für einen Meßplatz zur rechnergesteuerten Erfassung der Richtcharakteristik von Ultraschallwandlern“, Technische Universität Chemnitz, Fakultät für Elektrotechnik und Informationstechnik, Professur Meß- und Sensortechnik, 1994

[Kräm] Krämer, Karsten: Diplomarbeit „Untersuchung zum Einfluß der Passivierungs-Schutzschicht von PVDF-Hydrophonzeilen bei elektrisch steuerbarer gerichteter Abstrahlung“, Technische Universität Chemnitz, Fakultät für Elektrotechnik und Informationstechnik, Professur Meß- und Sensortechnik, 1995

Abbildungs- und Tabellenverzeichnis

Abbildungsverzeichnis

Bild 2.1: Gerätetechnischer Aufbau des Wassermeßplatzes....10
Bild 2.2: Etwaiger interner Aufbau der isel-Mikroschritt-Steuerkarte....11
Bild 2.3: Gerätetechnischer Aufbau des Luftmeßplatzes....15
Bild 2.4: Anschluß der Motorsteuerung über eine 3-Draht serielle Schnittstelle....16
Bild 2.5: Zeitlicher Ablauf der Ultraschallmessung....18
Bild 2.6: Speicherbelegung im linearen Adreßraum unter Windows (2 DOS-Boxen geöffnet)....22
Bild 2.7: Programmfluß bei vollständiger Virtualisierung eines Interrupts mittels VPICD_Virtualize_IRQ....30
Bild 2.8: Dreistufige Hierarchie bei der Adressierung der Daten durch das DDE-Protokoll....34
Bild 2.9: Erfolgreicher Verbindungsaufbau zwischen 2 DDE-Anwendungen....35
Bild 2.10: Datenanforderung....36
Bild 2.11: Erfolgreicher Datentransfer mittels “hot link”....36
Bild 2.12: Datenübertragung vom Client zum Server mittels wm_dde_poke-Nachricht....37
Bild 2.13: Senden eines Befehls an einen Server....37
Bild 2.14: Verbindungsabbau....37
Bild 2.15: Position der DDEML am Beispiel der DDEConnect Funktion....40
Bild 3.1: Vorschlag eines Versuchsaufbaus zur Erfassung der Strangstrom-Kurvenform....42
Bild 3.2: Beschleunigungs- und Abbremsrampen für s=12 und s=11....44
Bild 3.3: Beschleunigungs- und Abbremsrampen für s=13, s=14 und s=15....44
Bild 3.4: Programmablaufplan „Berechnung nächster Schritt“....46
Bild 3.5: Graf der Abhängigkeiten der Programm-Module....47
Bild 3.6: Vererbungs-Hierarchie der Objekte in MOTORST.EXE....48
Bild 3.7: Hauptfenster des Programms MOTORST.EXE....48
Bild 3.8: Dialogfenster zum Abfahren eines Bereiches sowie Fortsetzungsfenster....50
Bild 3.9: Dialogfenster zur Motor-Auswahl und zum Konfigurieren....50
Bild 3.10: Graf der Abhängigkeiten der Programm-Module....53
Bild 3.11: Vererbungs-Hierarchie der Objekte in SENDERST.EXE....54
Bild 3.12: Frequenz als Funktion des Abstrahlwinkels, mit dem Wandlerparameter als Parameter55
Bild 3.13: Sonderfall: Frequenz und Winkel maximal....56
Bild 3.14: Durch den großen Wandlerparameter bleiben größere Winkel nicht nutzbar....56
Bild 3.15: Der kleine Wandlerparameter verhindert die Abstrahlung in der Nähe von Null....57
Bild 3.16: Hauptfenster des Programms SENDERST.EXE....58
Bild 3.17: Dialoge für Ansteuerfrequenz und Abstrahlwinkel....59
Bild 3.18: Dialoge für Wandlerparameter und Einstellung....59
Bild 3.19: Graf der Abhängigkeiten der Programm-Module....62
Bild 3.20: Vererbungs-Hierarchie der Objekte in LUFTMP.EXE....63
Bild 3.21: MDI-Hauptfenster des Programms LUFTMP.EXE....64
Bild 3.22: Schaltplan der Ergänzungen am Motor und an der Motorsteuerung....67

Bild 3.23: 3 Dialoge zur Meßreihe 71
Bild 3.24: 3 Dialoge zur Hardware-Einstellung 71

Tabellenverzeichnis

Tabelle 2.1: Steuerprozedur-Rufe beim Start und Beenden von Windows 24
Tabelle 2.2: Steuerprozedur-Rufe beim Start und Beenden einer VM 24
Tabelle 2.3: Standard-Zwischenablage-Formate 33
Tabelle 2.4: Die in Windows verfügbaren DDE-Botschaften 35
Tabelle 2.5: Absender von DDE-Nachrichten und möglich Reaktionen 38
Tabelle 2.6: DDEML-Funktionen, die für DDE-Server in Frage kommen 39
Tabelle 2.7: DDEML-Transaktionen, die bei Server-Anwendungen eintreffen können 39
Tabelle 3.1: Liste der einzelnen Programm-Module zum Treiber 43
Tabelle 3.2: Liste der einzelnen Programm-Module zum VCALL0.386-VxD 66
Tabelle 3.3: Verwendete Kommandos für die Motorsteuerung 68
Tabelle 3.4: Bedeutung der Typ-Angabe 70

Anhang

A1 Technische Daten von Sendeendstufe und Spitzenwertgleichrichter für Luftmeßplatz

Komponenten der Hardware:
- Sendeendstufe
- Spitzenwertgleichrichter
- Netzteil

Leistungsmerkmale der Sendeendstufe:
- Ausgangsspannung umschaltbar auf 50 / 100 / 200 V
- Einstellbare Impulsbreite in zwei Bereichen:
  Bereich 1: 1 µs bis 16 µs
  Bereich 2: 8 µs bis 210 µs
- Interner Impulsgenerator für den Test (Testfrequenz 250 Hz)
- Referenzausgang mit 5 V Gleichspannung
- Wiederholgrenzfrequenz für Meßimpulse: 10 kHz bei 200 V

Leistungsmerkmale des Spitzenwertgleichrichters:
- Eingangsspannung: 0..5 V
- Ausgangsspannung: 0..5 V
- Bandbreite: 1 MHz
- Impulshaltezeit: einige µs

Leistungsmerkmale des Netzteils
- Eingangsspannung: 220 V ~ / 50 Hz
- Ausgangsspannungen und Ströme:
  +15 V / 300 mA stabilisiert
  -15 V / 300 mA stabilisiert
  35 V ~ / 50 mA

A2 Technische Daten AD-12-Bit-Karte für Luftmeßplatz

Die Karte beinhaltet einen 12-bit-Analog-Digital-Umsetzer, dessen Eingang auf 16 mögliche Eingangskanäle geschaltet werden kann. Vier dieser Eingangskanäle enthalten eine Sample&Hold-Schaltung, die für die erforderliche Konstanthaltung des Meßsignals während des Umsetzvorganges eingesetzt werden kann. Die Umsetzung kann wahlweise uni- oder bipolar erfolgen, diese Auswahl erfolgt auf der Karte mit einem Jumper.

Zusätzlich befindet sich auf dieser Karte der parallele Ein-/Ausgabebaustein 8255, von dem 16 seiner 24 programmierbaren Ein-/Ausgabeanschlüsse nach außen geführt sind.

Die Karte belegt im Ein-/Ausgabeadreßraum des Prozessors 8 aufeinanderfolgende Adressen sowie eine IRQ-Leitung.

Technische Daten des Wandler-Teils:

Umsetzverfahren:	Sukzessive Approximation
Zahlendarstellung:	12 bit Binärwert, als Zweierkomplementzahl oder ohne Vorzeichen je nach Betriebsart
Umsetzzeit:	7..25 µs (je nach eingesetztem Wandler-Schaltkreis)
Genauigkeit:	1 LSB (244 ppm FS)
Referenzspannung:	intern, 10 V, abgleichbar
Nullpunkt:	abgleichbar (keine Automatik)
Grenzfrequenz der S&H-Schaltung:	2,5 MHz
Ansteuerung der S&H-Schaltung:	intern oder extern
Eingangsspannungsbereich:	–2,5..+2,5 V, –5..+5 V, 0..5 V, Auswahl mittels Jumper

Technische Daten des Digital-Teils:

Wählbare I/O-Basisadressen:	Alle durch 16 teilbaren Adressen im Bereich 000h..FF0h per Jumper
Ausdekodierte Adreßleitungen	12
Wählbare IRQ-Signale:	2, 3, 4 und 7 per Jumper

(Aus dem mitgelieferten Schaltplan wird die IRQ-Verschaltung nicht ersichtlich.)

Belegte Portadressen:

Basis+0..Basis+3: Adressierung des Wandlerschaltkreises HS 574 (nur Lesen)
+0, +1: Umsetzvorgang einleiten (Dummy-Lesezugriff)
+2: High-Byte
+3: Low-Nibble in den Bits 4..7, übrige Bits undefiniert

Basis+4..Basis+7: Adressierung des PIA 8255 (Lesen und Schreiben)
+4: PIA Port A: Bits 0..3 wählen den analogen Eingang aus, Bit 7 steuert die S&H-Schaltungen (alle 4 gleichzeitig)
+5: PIA Port B: Alle Bits sind frei verfügbar nach außen geführt.
+6: PIA Port C: Alle Bits sind an einem 8pol. Pfostenstecker frei verfügbar
+7: PIA Steuerport

A3 Formular zum Erhalt einer VxD-ID

```
   ********************************************************************

         Contact Name(s):

         Phone Number(s):

     Alt-Phone Number(s):

Personal Compuserve Acct:
  Support Advantage Acct:
  Internet Email Address:

         Company Name:
           Address:
    City/State/Zip:
           Country:
     Company Phone:
       Company Fax:
     Company Email:
  Company CIS Acct:


   Number of VxD's planned: __

Number of VxD ID requests enclosed: __

Number of VxD ID's assigned to your company so far: __

   ----- Repeat following section for each VxD -----

  VxD File Name: _______.386     (Avoid using a V____D.386 name if possible.)

  Will this VxD be loaded from a TSR? (Yes/No) : ___

  Will this VxD call out to an MS-DOS device driver or
  TSR using Interrupt 2Fh, Function 1607? (Yes/No) : ___

     Please provide the estimated number of API's/Exports below.
  V86 functions :
  PM functions :
  VxD Services :

  If this VxD replaces a "standard" VxD, which one does it replace : ________


Summarize the purpose of this VxD :


Please provide a technical summary of this VxD below.  (Please include for
example, all interrupts hooked,  I/O ports trapped, and memory trapped.):
```

A4 Verzeichnisstruktur und Dateiliste der Programmdiskette

Aufgrund der erheblichen Menge der vorhandenen Dateien enthält die Diskette ein Archiv mit folgendem Inhalt:

\	
├──WASSER	**Alle Dateien zur Sendersteuerung**
│ SENDERST.HLP	Sendersteuerung-Hilfedatei für Windows
│ SENDERST.EXE	Ausführbares Windows-Programm
│	
│ ├──EXESRC	**Quellen für .EXE**
│ │ USMDDE.PAS	DDE-Server
│ │ USMP.PAS	allgemeine Prozedursammlung
│ │ USMH.PAS	allgemeine Konstanten-, Typ- und Variablensammlung
│ │ ODDE.PAS	objektorientierter Wrapper („Verhüller“) für DDEML
│ │ CONFDLG.PAS	Dialog (Konfiguration)
│ │ FREQDLG.PAS	Dialog (Ansteuerfrequenz-Eingabe)
│ │ PARAMDLG.PAS	Dialog (Wandlerparameter-Eingabe)
│ │ WINKELDL.PAS	Dialog (Abstrahlwinkel-Eingabe)
│ │ SENDERST.PAS	Hauptprogramm
│ │ SENDHW.PAS	Hardware-Steuerung
│ │ SENDERST.RES	binäre Ressourcendatei für WORKSHOP.EXE
│ │	
│ └──HELPSRC	**Quellen für .HLP**
│ SENDERST.RTF	Hilfetext im Rich-Text-Format, erstellt mit WinWord 2.0
│ SENDERST.HPJ	Hilfe-Projektdatei
│ ADJUST.BMP	Bitmap (Einstellregler)
│ DDE.BMP	Bitmap (zwei Fenster mit Blitz)
│ DIPIC.BMP	Bitmap (Schaltkreis)
│ HELP.BMP	Bitmap (gelbes Fragezeichen)
│ RISS.BMP	Bitmap (Kabel mit Riß)
│ TITEL.BMP	Bitmap (Wandlereinheit im Wasser)
│ FREQ.SHG	Hotspot-Bitmap (Frequenz-Dialog)
│ MAIN.SHG	Hotspot-Bitmap (Hauptfenster)
│ WINKEL.SHG	Hotspot-Bitmap (Winkel-Dialog)
│ CONFIG.SHG	Hotspot-Bitmap (Konfigurations-Dialog)
│ PARAM.SHG	Hotspot-Bitmap (Wandlerparameter-Dialog)
│	
├──MOTOR	**Alle Dateien zur Motorsteuerung inklusive Gerätetreiber**
│ MPK3D.386	Motor-Gerätetreiber als VxD für Windows
│ MPK3D.HLP	Referenz für Programmierer zum Treiber als Windows-Hilfe
│ MOTORST.EXE	Ausführbares Windows-Programm zur Motorsteuerung
│ MOTORST.HLP	Motorsteuerungs-Hilfedatei für Windows
│	

├──EXESRC		**Quellen für .EXE**
	BEREICH.PAS	Dialog (Bereich abschreiten)
	GRENZEN.PAS	Dialog (Grenzen festlegen)
	STEPDLG.PAS	Dialog (Bereich abschreiten im Gang)
	MOTORSEL.PAS	Dialog (Motor-Auswahl)
	LOADSAVE.PAS	Routinen zum Laden und Speichern der Konfiguration
	MOTORST.PAS	Hauptprogramm
	ODDE.PAS	objektorientierter Wrapper („Verhüller“) für DDEML
	SMSDDE.PAS	DDE-Server
	SMSH.PAS	allgemeine Konstanten-, Typ- und Variablensammlung
	SMSP.PAS	allgemeine Prozedursammlung
	MOTORST.RES	binäre Ressourcendatei
├──HELPSRC		**Quellen für .HLP zur Motorsteuerung**
	ADJUST.BMP	Bitmap (Einstellregler)
	DDE.BMP	Bitmap (zwei Fenster mit Blitz)
	DIPIC.BMP	Bitmap (Schaltkreis)
	HELP.BMP	Bitmap (gelbes Fragezeichen)
	RISS.BMP	Bitmap (Kabel mit Riß)
	SMS10.BMP	Bitmap (Motor-Symbol)
	MOTORST.HPJ	Hilfe-Projektdatei
	MOTORST.RTF	Hilfetext im Rich-Text-Format, erstellt mit WinWord 2.0
	BEREICH.SHG	Hotspot-Bitmap (Bereich-Dialog)
	GRENZEN.SHG	Hotspot-Bitmap (Grenzen-Dialog)
	MAIN.SHG	Hotspot-Bitmap (Hauptfenster)
	MOTORSEL.SHG	Hotspot-Bitmap (Motorauswahl-Dialog)
├──VXDSRC		**Quellen für .386 (mit Interface-Modulen)**
	MPK3D.ASM	Quelltext zum Virtuellen Gerätetreiber
	MPK3D.DEF	Modul-Definitionsdatei zum .ASM
	MPK3D.C	Quelltext Anwenderschnittstelle für Turbo/Borland C
	MPK3D.H	Header-Datei für Turbo/Borland C
	MPK3D.MAK	MAKE-Datei (Ausführen mit `make -f mpk3d.mak`)
	MPK3D.PAS	Quelltext Anwenderschnittstelle für Turbo/Borland Pascal (als Unit)
└──REFSRC		**Quellen für Motortreiber-Referenz MPK3D.HLP**
	MPK3D.HPJ	Hilfe-Projektdatei
	MPK3D.RTF	Hilfetext im Rich-Text-Format, erstellt mit WinWord 2.0
├──LUFT		**Alle Dateien zur Luftmeßplatz-Steuerung**
	LUFTMP.EXE	Ausführbares Windows-Programm
	LUFTMP.HLP	Hilfedatei zur Luftmeßplatz-Steuerung für Windows
	VCALL0.386	Universell verwendbarer Gerätetreiber für dieses Programm

├──EXESRC	**Quellen für .EXE**
ADCMESS.PAS	Hardware-Steuerung (ADC und Zeitsteuerung)
PLDLGRES.PAS	"PlayDialogResource"-Unterprogramm
APPTOOLS.PAS	Unit für Statuszeile
DDESERV.PAS	DDE-Server (komplexer objektorientierter Wrapper)
DDEH.PAS	Anpassungen zum DDE-Server
HUGEMEM.PAS	Unit zum Ansprechen von Speicher mit mehr als 64 KB
LMPH.PAS	allgemeine Konstanten-, Typ- und Variablensammlung
LMPP.PAS	allgemeine Prozedursammlung
LUFTMP.PAS	Hauptprogramm
MATLAB.PAS	Matlab-Datei-Schnittstelle
MESSREIH.PAS	Alle Fenster: ADC, Motor und Meßreihe
SETUPS.PAS	Alle Dialoge des Programms
LUFTMP.RES	binäre Ressourcendatei
├──HELPSRC	**Quellen für .HLP**
ADJUST.BMP	Bitmap (Einstellregler)
BOOK.BMP	Bitmap (grünes Buch)
DDE.BMP	Bitmap (zwei Fenster mit Blitz)
DIPIC.BMP	Bitmap (Schaltkreis)
HELP.BMP	Bitmap (gelbes Fragezeichen)
LUFTMP.BMP	Bitmap (Schallkeule)
RISS.BMP	Bitmap (Kabel am Stecker mit Riß)
WINDOW.BMP	Bitmap (Programm-Manager-Fenster)
LUFTMP.HPJ	Hilfe-Projektdatei mit umfangreichem [MAP]-Abschnitt
LUFTMP.RTF	Hilfetext im Rich-Text-Format, erstellt mit WinWord 2.0
└──VXDSRC	**Quellen für .386 (mit Interface-Modulen)**
VCALL0.ASM	Quelltext zum Virtuellen Gerätetreiber
VCALL0.DEF	Modul-Definitionsdatei zum .ASM
VCALL0.C	Quelltext Anwenderschnittstelle für Turbo/Borland C
VCALL0.H	Header-Datei für Turbo/Borland C
VCALL0.PAS	Quelltext Anwenderschnittstelle für Turbo/Borland Pascal (als Unit)
VCALL0.TXT	Programm-Dokumentation
VCALL0T.PAS	Einfaches Einsatzbeispiel
├──DDK-LITE	**zusätzliche Werkzeuge zum Übersetzen der VxD's**
├──INCLUDE	**.INC-Dateien zum Einbinden**
DEBUG.INC	Debug-Makro-Sammlung, angepaßt an Turbo Assembler
TVMM.INC	Alle VMM-Funktionen und Makros, angepaßt an Turbo Assembler
VPICD.INC	VPICD-Funktionen und -Typen (zum Interrupt-Controller)
VTD.INC	VTD-Funktionen (zum Timer)
└──BIN	**.EXE-Dateien zum Erstellen**
LINK386.EXE	Linker für Dateien mit Kennung "LE"
ADDHDR.EXE	Patchprogramm, setzt die Versionskennung ein
├──MPK3DRV	**Ergebnisse der Untersuchung der Datei MPK3DRV.EXE**
MPK3DRV.C	Etwaiger C-Quelltext (Teile)
MPK3DRV.SIN	Sinustabelle
MPK3DRV.CAS	Etwaiges Assembler-Listing

├──RCMPLATZ	**Modifikationen am vorhandenen Luftmeßplatz-Programm**
│ RCMHS.EXE	Ausführbares Programm für DOS
│ MESSMOD.C	Veränderte Datei (portable Zeitsteuerung, Zeigergrößen)
│ MAINMOD2.C	Veränderte Datei (Zeigergrößen)
│ INPUT.C	Veränderte Datei (Zeigergrößen)
│	
├──MATLAB	**Matlab-Beispieldateien und Beispielskript**
│ TEST2.M	Beispiel-Skript zur Steuerung von SENDERST.EXE und MOTORST.EXE
│ WE5.MAT	eine Luftmeßplatz-Meßreihe
│ WE6.MAT	noch eine Luftmeßplatz-Meßreihe
│ WE7.MAT	noch eine Luftmeßplatz-Meßreihe
│	
├──VONAHL	**„Datenaustausch“ mit dem Betreuer**
│ DDESERV.OEM	Beschreibung der DDE-Server-Unit
│ DDESERV.PAS	DDE-Server-Unit (leicht modifiziert angewandt bei LUFTMP.EXE)
│ XLTABL~1.TXT	Beschreibung des binären Excel-DDE-Formats
│	
├──DIPARB	**Diplomarbeit**
│ DIPARB.DOC	diese Arbeit als Winword-2.0-Datei
│ LOTUS1.TTF	verwendeter True-Type-Font für OEM-Rahmensymbole
│	
└──GRPSTART	**Sonstige Utilities**
│ GRPSTART.EXE	Gruppen-Starter für Windows
│ SETWIN31.EXE	Setzt erforderliche Windows-Version auf 3.1 (DOS-Programm)
│ WINTMR.EXE	Testprogramm für Timer unter DOS und Windows (Zwitter-Programm)
│ THREADEX.EXE	Beispielprogramm zur Realisierung kooperativen Multithreadings (DOS)
│ LPTDAC.386	Beispiel-VxD zur Virtualisierung nicht existenter Hardware
│	
└──EXESRC	**Quelldateien**
GRPSTART.PAS	Quelldatei für Gruppen-Starter
GRPSTART.RES	binäre Ressourcen-Datei für Gruppen-Starter
SETWIN31.PAS	Quelldatei für Versionsnummern-Setzprogrsmm
WINTMR.PAS	Quelldatei für Timer-Testprogramm
WINTMR.MAK	MAKE-Datei (Ausführen mit `make -f wintmr.mak`)
GLUE.EXE	Kombinierer zur Erstellung von Zwitter-Programmen
THREAD.PAS	Demonstrations-Unit zur Realisierung kooperativen Multithreadings
THREADEX.PAS	Anwendungsbeispiel dieser Unit
LPTDAC.ASM	Quelle zum Beispiel-VxD zur Virtualisierung nicht existenter Hardware